Technical Support

Before You Call For Technical Support...

Xiris prides itself on its technical support. Our technical support people are programmers, not temporary employees. They will do their utmost to help you resolve your programming problems. In the event that you discover a bug or anomaly, Xiris will fix the problem as soon as possible, and deliver a fix in a timely fashion. In the event that you are doing something incorrectly in your code, the technical support team will help you to determine what you are doing wrong.

However, in order to maximize the effectiveness of your contact with Xiris technical support, we ask that you have as much information as possible about you problem ready when you call, so that less valuable time will be wasted.

So, before calling for technical support, FIRST, read the following points thoroughly:

1) **GPF?** If you are getting a GPF (General Protection Fault), write down the information that is displayed when the error occurs. This is invaluable information. Also, make a note of what your code was doing (in general terms).
2) **ISOLATE IT**. Try to isolate the cause of the error. If at all possible, step through your code with F8 and F9. Try to find the one line of code that is causing the error.
3) **SCALE IT DOWN**. If at all possible, try to reproduce the problem in a small test program. Use very little code, so that the technical support team can duplicate the problem without hoarding through piles of irrelevant code.
4) Call Xiris. At this point, you should have a small test program that you can send in. Call the technical support line at (905) 681-8107 to discuss the problem, and possibly arrange the method by which you can send in your test code.
5) IF YOU CANNOT REACH A TECHNICAL SUPPORT PERSON, you can send in you test program via:

Xiris **FAX**: (905) 681-9844 (Faxes must be 2 pages maximum!)

Xiris BBS: (905) 681-6327

Xiris **Email**: xiris@hookup.net

XCaliper Help

XCaliper Tools: Theory Of Operation

Visual Basic Custom Controls

Setup Dialog Boxes

References

Technical Support

XCaliper Overview

XCaliper is a general purpose, non-contact measurement package designed for industrial applications where highly accurate, repetitive measurements are required. It can function as a precision measurement tool, a locating system, or as a feature presence/absence inspection system. When integrated with the appropriate hardware, it can perform measurements on the production line, providing 100% inspection of production volumes.

A typical application for ***XCaliper*** would be gauging of parts on a production line. In such an application, ***XCaliper*** would be integrated into a simple Visual Basic program as part of an automatic inspection system that would use the measurement results from ***XCaliper*** to decide whether to pass or fail the parts as they pass by on the production line.

The package is configured as VBX tools, allowing the user to build an application using Microsoft Visual Basic for Windows, a powerful programming language that allows the integration of numerous standard software packages. Because the language is so easy to use, a professional user interface can be developed for an ***XCaliper*** inspection application with very little effort, allowing the entire application to be up and running after only a few hours of development effort. Output of the applications generated data can be sent to other VBXes or to external programs for statistical analysis, chart generation, or data storage.

XCaliper includes six processing tools: four edge locators, a blob extraction and analysis engine, and a LightMeter. All six tools operate on a Viewport or region of interest within the frame buffer, rather than the full frame buffer, thus increasing both the speed and accuracy of the tool. The Viewport can be positioned either interactively using pick points or through direct program action. The pick points can be suppressed by the designer to provide control over end-user action. ***XCaliper*** includes three additional Image Tools to manage the interaction with the frame buffer and the display.

The typical automatic inspection application will require at least two controls: a frame buffer control, probably the **ImageDevice** in which to store images (or a **MemoryBuffer** if no image acquisition is required), and at least one processing tool to perform the actual image analysis. Normally there will be at least three or four processing tools, while many applications may use a dozen or more. There is no restriction on the number of processing tools used in an application, beyond Windows memory and resource limitations.

Recommended Software Development Cycle

Creating application for Windows in Visual Basic is a significantly different process than the standard development cycle one might be familiar with using packages such as Microsofts Visual C++ or Borlands Turbo C++. The two primary reasons for this difference are the following:

1) Visual Basic provides powerful, graphically oriented, interface building capabilities. By using the controls provided on the Toolbox, and setting their properties appropriately one can layout a comprehensive, professional looking application in a matter of minutes.

2) Visual Basic, like all Windows environments, enforces an *event-driven* programming paradigm. That is, code is executed when an event occurs for a particular control (such as clicking a button, or text being modified in a Text box, or a **Form** being loaded). As controls are added to a project, Visual Basic automatically sets up the event procedures for these controls. It is simply a matter of filling in the code for these routines to define the functionality of the particular control.

For further information on the above two elements, see Chapters 2-6 of the Visual Basic Programmers Guide. The Programmers Guide defines three main steps for creating an application in Visual Basic:

1) Create the user interface.

2) Set the property values for the controls desired.

3) Write the code for the event procedures and any necessary support procedures.

The following description expands on the information presented in the Programmers Guide and attempts to tailor the steps for programmers using the ***XCaliper*** controls.

The first step in building an ***XCaliper*** application is to add the ***XCaliper*** custom controls to Visual Basics Toolbox. While this can be done manually, it can be automated by placing the custom control names into the AUTOLOAD.MAK file. To add custom controls to the Toolbox manually, perform the following:

1) From the **File** menu, choose **Add File**. The Add File dialog box appears.

2) Open the directory which contains the ***XCaliper*** .VBX files. Select a .VBX file (XVBXIMG.VBX for example) containing the custom controls desired and click the **Ok** button. The icons representing the custom controls contained in that file should appear on the Toolbox.

3) Repeat step number two until all the controls required have been added to the Toolbox. The order the files are added doesnt matter. There is no harm in adding controls to the Toolbox that may not be used (provided there is sufficient RAM), as only the controls placed on a Form will be kept with the application.

Before placing the ***XCaliper*** controls on a **Form** you will first have to decide which container to use for the chosen image. The container is the control in which the image will appear. It is recommended that you use a **PictureBox** that is placed within a **Form**. Using a **Form** directly, will cause the entire Form to be used for image display. By employing a **PictureBox**, the image display area can be easily sized and positioned according to the requirements of the application, as well as the constraints created by other controls on the **Form** such as buttons, scroll bars, and text boxes.

Once you have decided on a container and have positioned it on the Form, begin placing ***XCaliper*** controls on it. Either an ImageDevice control, or a MemoryBuffer control must be included in any application in which you intend to make use of the ***XCaliper*** tools.

Assuming a frame grabber has been installed, (using a **MemoryBuffer** control is a very similar process), select the **ImageDevice** icon on the Toolbox. Then, in the **PictureBox** (or whatever control you are using as a container) drag out a small rectangle where you want to place a control. The actual size and location of the rectangle is irrelevant as the control simply appears as an icon where youve placed it and is not visible during run-time. Check to see if the XCALIPER.INI file has the correct name of the driver for your frame grabber type. If ***XCaliper*** has not been configured to operate with your frame grabber, change the DeviceFile property of the **ImageDevice** to the device driver file appropriate for it. Certain drivers may also require special settings to be provided through the XCALIPER.INI file.

Once the correct driver has been loaded through the *DeviceFile* property, the remainder of the **ImageDevice**s properties should be set appropriately. For example, now would be a good time to set the *Name* and Tag properties to something more descriptive. Verify that the **ImageDevice**s GrabMode, InputChannel, InputGain, InputOffset, and SyncMode properties are set appropriately and that the Transparent property setting agrees with your card type. Improper values for any of these properties could result in a poor or nonexistent image display.)

Once the image display is operating correctly, begin placing ***XCaliper*** processing tools on the image container as desired. Once the Viewports are positioned and sized, the next step is to set their properties according to your processing requirements. This will likely be an iterative process: your processing requirements will change and become more detailed once you begin using and experimenting with the tools.

When the tools have been configured in an appropriate fashion for the application, begin adding the required code that will control how the tools process, and analyze their results. Typically, a skeletal version of the code is implemented that allows the programmer to control and possibly configure the tools during run-time (using buttons and scroll bars for example), as well as displaying specific results (using label controls for example). This allows the programmer to experiment with various configurations and settings early in the development cycle and use the insights gained to focus and steer the application development process. This is one of the primary advantages of working in the Visual Basic environment the ability to prototype and reconfigure an application quickly in order to experiment and test various design options with a minimum of programming and debugging time.

The development cycle is completed by creating an executable (.EXE) version of the application when you feel comfortable that it is complete and functioning correctly. This file can be run like any other .EXE file. If you plan to distribute the application you will likely want to use one of the utilities by Visual Basic for creating distribution disks. For more information on creating distribution disks, see Chapter 25 of the Programmers Guide.

Accuracy

The degree of conformance between a measurement of an observable quantity and a recognized standard or specification which indicates the true value of the quantity.   Accuracy normally denotes absolute quality or conformance; PRECISION refers to the level of detail used in reporting the results.

AnyAngle

A type of linear Viewport whose path can be along any angle of inclination. This differs from Horizontal and Vertical Viewports since the angle of inclination of these Viewports are fixed at 0º and 270º respectively.

Apply

The execution of the ***XCaliper*** tool on the region of the image inside the Viewport.

Arc

An open or closed circular or elliptical path. An arc is used to define a path over which edges or edge pairs are searched. The size and curvature of an arc are defined by three points so that the arc passes through each of the points.

ArcCaliper

An ***XCaliper*** Edge Locator tool used to find pairs of edges along a circular or elliptical Path.

ArcEdgeLocator

An ***XCaliper*** Edge Locator tool used to find single edges along a circular or elliptical Path.

AspectRatio

The ratio of the horizontal diameter to the vertical diameter of a closed curve ellipse.

Blob

An arbitrary group of connected pixels in an image.   The group of pixels that make up a blob must share a common or similar light intensity.   A blob often represents a specific real world object for which measurements need to be obtained for inspection.

Blob Analysis

The process of extracting blobs from an image is performed to obtain statistics and other information. This information can then be used to determine presence or absence, location and many other characteristics of the real-world objects which these blobs represent.

BlobAnalyzer

A blob extraction tool used to identify and analyze blobs in an image. Statistics for each blob are produced for inspection and comparison against an ideal object for which the blob represents.

Caliper

The type of measurement tool supplied by ***XCaliper*** as a VBX control used to measure the distance between a pair of edges. Candidate edges are found in a Viewport, then scored based on the active Constraints for the application. Edges are then paired together by finding pairs of edges that match the active Constrains.

Click

To position the cursor on top of a designated item using the mouse and press and release the primary mouse button.

ClosedCurve

A Path which begins at the *StartPoint*, continues through the *MidPoint* and *EndPoint*, and completes an arc back to the *StartPoint*.

Constraint

One of the criteria that ***XCaliper*** uses to evaluate an Edge or an Edge Pair.   Each Constraint has an Edge Evaluation Function and a Weight associated with it.   The constraints are used to remove unwanted edges from the candidate edge list.

Constraint Score

A numeric value within the range [0-1] that indicates how close the actual value of a constraint is to the ideal value.   A value of 1 would indicate a perfect match as determined by the Edge Evaluation Function that has been defined for the constraint.

Container

A Visual Basic object in which you can place other controls. When the container is moved, the controls inside it move with it. Thus, it is possible to group a series of controls together. Standard Visual Basic comes with two containers: the Form and the Picture Box.

Convolution

A process whereby an image is transformed mathematically by applying a kernel which is a set of multipliers applied to the neighborhood of each pixel.   Applications of convolution include sharpening and smoothing.

Cumulative Histogram

Similar to a standard histogram, except the function plotted is the frequency of occurrence of *all intensity values less than or equal to the current intensity value.* As such, a cumulative histogram is always a monotonically increasing function whose maximum value will be the total number of pixels contained within the region of interest.

Difference Filter

A one dimensional array that is convolved with the projection to produce an Edge Intensity Graph. All elements to the left of the center have the value -1, and all elements to the right of the center have the value +1. The default filter size, the distance from the center point to the edge of the filter, is 2.

Discriminator

The first criteria that ***XCaliper*** uses to evaluate an Edge or an Edge Pair. Discriminators act as a very fast evaluation criteria to remove any candidate edges that do not meet basic criteria. The Discriminators available are Minimum Strength and Polarity, as well as Straddle which is only for Edge Pairs.

Drag

The action involved in moving an object on the screen using the mouse. To drag an object is to position the cursor over the object, depress the primary mouse button and hold the button down while moving the mouse. When the desired action is completed, the mouse button is released.

E1 Position

An attribute that defines the location of the first edge of an Edge Pair. For a Viewport with inclination of 0, E1 is the leftmost edge of the Edge Pair.

E1 Strength

An attribute that defines the difference in light intensity across the first edge of an Edge Pair. For a Viewport with inclination of 0, E1 is the leftmost edge of the Edge Pair.

E2 Position

An attribute that defines the location of the second edge of an Edge Pair. For a Viewport with inclination of 0, E2 is the rightmost edge of the Edge Pair.

E2 Strength

An attribute that defines the difference in light intensity across the second edge of an Edge Pair. For a Viewport with inclination of 0, E2 is the rightmost edge of the Edge Pair.

Edge

A rapid change in light intensity that spans several pixels between two adjacent regions of relatively uniform values. Edges correspond to changes in brightness resulting from a discontinuity in surface orientation, reflectance, or illumination.

Edge 1 (or E1)

The first edge of an Edge Pair. For a Viewport with inclination of 0, E1 is the leftmost edge of the Edge Pair.

Edge 2 (or E2)

The second edge of an Edge Pair.   For a Viewport with inclination of 0, E2 is the rightmost edge of the Edge Pair.

Edge Evaluation Function (or EEF)

The Function that defines what Constraint Score is assigned by ***XCaliper*** to an active Constraint based upon the deviation between the actual and ideal value of that Constraint.

Edge Strength

The array of gray scale values derived from the projection, with a width equal to the width of the Viewport.   The value for each element in the array corresponds to the change in light intensity between two adjacent columns in the Viewport.

Edge Intensity Graph

The graph shows the strength of the edges found in the Viewport by plotting the strength across the edges. It is the graph resulting from a filter convolution of the Projection.

EdgeLocator

The type of measurement tool supplied by ***XCaliper*** as a VBX control to measure the location of an edge.   Candidate edges are found in a Viewport, then scored based on the active Constraints for the application.

Edge Pair

Two Edges in an image that are separated by a specific distance.

Edge Score

A numeric value within the range [0-1] that ranks each edge based on the sum of all the weighted scores of active constraints. A value of 1 would indicate a perfect edge. When no constraints are active, edges that pass the Discriminators are defined as having a value of 1.00.

EEF Point

One of the six points that define an Edge Evaluation Function. Three points are located on each side of the Expected Position: one point to represent the cut off for acceptable positions, another to represent the perfect Score area, and the third point to represent the rate of Score decay as the actual position varies from the Expected Position.

EndPoint

The point on an open Path at which measurements terminate.   For a closed Path, measurements continue past this point to the StartPoint.

Expected Position

The ideal value for the Position constraint of an edge or edge pair, relative to the left side of the Viewport.

Expected Size

The ideal value for the Size constraint of an edge or of each edge in an edge pair, relative to the left side of the Viewport.

Feature

A type of statistic relating to a specific attribute of a blob. Examples of features include Area, FormFactor, Orientation, etc. A complete list of features can be found in the BlobAnalyzer User's Guide.

Fiducial

A mark or target defining a datum point or standard of positional reference used as a basis for calculation or measurement.

Filtration

Filtration is the process of eliminating uninteresting blobs from an image. A filter describes a suitable range of acceptable values of a feature for a blob to be accepted. Blobs whose features do not meet the filter requirements are rejected.

Filter Size

The size of the filter that is applied to the projection of the Viewport.   It is defined as the number of elements in the filter to one side of the center element.

Frame Buffer

The memory designed to store a digitized image. Typically, it resides on a Frame Grabber card that plugs into an expansion slot of a computer that is capable of capturing an image and storing it. A window to the Frame Buffer can be identified on a VGA screen by the picture that can be seen in it.

Frame Grabber

A hardware device typically configured as a circuit board that plugs into a slot inside a personal computer with on board circuitry to perform image acquisition, storage and display.

Frame Store

A block of memory in which an image is stored. The memory can be part of a frame grabber or the computers system memory.

Histogram

A function plotting the frequency of occurrence of an intensity value as a function of those intensity values. As such, a histogram illustrates the distribution of intensity values in a given region of interest in an image.

HoleFill

Hole-filling fills holes in blobs so that background areas (and other blobs) which are inside a blob are considered as part of that blob for the purpose of later analysis.

Ideal Value

The value of a constraint that would generate a perfect (1.00) Constraint Score.

Inclination

Describes the angle of inclination of a Viewport. Horizontal Viewports have a fixed inclination of 0º, Vertical Viewports have a fixed inclination of 270º, and the inclination of AnyAngle Viewports can be varied.

Labelization

The process of assigning unique identifiers or names to blobs. Labelization ensures that each blob in an image can be referenced by a blob number.

Left Tail

The left tail is the intensity value below which a pre-defined percentage of intensity values fall. It is sometimes desirable to treat intensity values below the left tail value as unreliable and to be discarded in subsequent processing.

LightMeter

The LightMeter generates statistics of the intensity values inside a Viewport. The tool provides the capability to obtain and manipulate a histogram of the intensity values. It also supports a Stats property that allows the program designer and/or user to obtain a variety of statistical quantities derived from these values.

Line

A straight line segment that is defined by two points. A line is used to define a path over which edges or edge pairs are searched. A line can be positioned between any two points at any angle.

LineCaliper

An ***XCaliper*** Edge Locator tool used to find pairs of edges along a straight line Path. The path can be Horizontal, Vertical or at AnyAngle.

LineEdgeLocator

An ***XCaliper*** Edge Locator tool used to find single edges along a straight line Path.   The path can be Horizontal, Vertical or at AnyAngle.

Mask

A method of segmentation that allows a binary bitmap to specify the foreground and background of an image on a pixel by pixel basis.   The bitmap is set using the BlobAnalyzer's *Mask* property.

MidPoint

The point on a Path between the StartPoint and the EndPoint used to define the radius of curvature.

Minimum Accept Threshold

The minimum acceptable Edge Score for an Edge or Edge Pair to be accepted as valid.   An Edge (in the case of an Edge Locator) or Edge Pair (in the case of a Caliper) with a lower Edge Score than the Minimum Accept Threshold will be discarded.

Minimum Strength

The minimum acceptable Edge Strength for an edge. An edge that has a strength value less than the Minimum Strength will be discarded.

Noise

Irrelevant or meaningless image information resulting from random undesirable video signals, or causes unrelated to the source of data being measured or inspected.

OpenCurve

The Path which begins at the *StartPoint*, and finishes at the *EndPoint*.

Origin

The edge of the Viewport from which all measurements are made. The origin of the Viewport for an edge tool is defined by the StartPoint.

Overlay

A graphics plane that typically exists on top of a frame buffer to display graphics. For example, the Viewport is an overlay on the Frame Buffer image. Generally, an overlay is the video plane that appears on top on a video monitor when two or more video signals are mixed.

Pair Position

The midpoint between the two edges that make up the Edge Pair, measured relative to the origin of the Viewport. For a Viewport with 0 inclination, the origin is the left side of the Viewport.

Pair Size

The distance (in user units) between the two edges that make up the Edge Pair.

Path

The path defines the course which is followed when scanning for edges. A path can be a line that is Horizontal, Vertical, at AnyAngle, and even circular or elliptical. A path and its thickness define a Viewport.

Pixel Jitter

An error introduced into a digitized image caused by Analog to Digital Converters attempting to lock sync with the video camera by picking up the horizontal sync pulse using Phase-Locked-Loop (PLL) circuitry. It arises when the horizontal sync for each scan line drifts slightly from the expected position, causing the position of all the horizontal picture elements to be shifted by an equal amount in the scan line.

Polarity

An attribute of an edge that indicates the type of transition in light intensity across the edge. A light to dark transition is said to have a Negative Polarity, whereas a dark to light transition is said to have a Positive Polarity. The polarity of an edge is viewed as occurring from the origin to the opposite side of a Viewport.

Position

An attribute of an edge that indicates the edges position relative to the origin of the Viewport. For a Viewport with inclination of 0, the origin is the left side of the Viewport.

Precision (of a Path)

The precision of a path defines the number of iterations that are used when finding the projection of a Viewport. Each iteration is performed at a slightly different location in the pixel grid so that a better average of light intensity along a path can be obtained. The precision only applies to AnyAngle linear paths or Arc (circular or elliptical) paths.

Projection

An array of values with a width equal to the width of the Viewport. The value for each element in the array corresponds to the average light intensity for each pseudo column. A pseudo column is the column of pixels which are perpendicular to the path of the Viewport (unless a non-zero Skew is specified) and whose height is defined by the thickness of the Viewport.

Region of Interest

The area of an image inside defined boundaries that encloses all the features that are to be inspected.

Repeatability

The amount by which repeated measurements of the same quantity and same operating conditions vary from their mean.

Right Tail

The right tail is the intensity value above which a pre-defined percentage of intensity values fall. It is sometimes desirable to treat intensity values above the right tail value as unreliable and to be discarded in subsequent processing.

Segmentation

Segmentation is the job of separating of an image into foreground and background, determining which pixels belong to blobs and which belong to the background.

Skew

Normally edge tools find edges that are perpendicular to the Path. Specifying a nonzero skew will cause edges to be detected that are at an angle with respect to the path.

StartPoint

The point on a Path from which all measurements are made. For a Horizontal Line Path, the StartPoint is its leftmost edge.

Statistics

Statistics describe various dimensional and characteristic information about an object they represent in a numeric format. A complete list of statistical types that are supported by the tool can be found in the User Guide.

Straddle

An attribute of an Edge Pair defining the condition where the E1 edge is closer to, and the E2 edge farther away from the origin of the Viewport than the mid point between the two Expected Positions of each edge. Used as a Discriminator to eliminate those Edge Pairs that do not meet this condition.

Strength

A attribute of an edge that indicates the magnitude of the change in light intensity across the edge.

Tail Size

The tail size is set by the programmer and is the percentage of intensity values to be included as part of the left/right tail.

Teach

A Teach must be performed whenever a new path or thickness is defined.   A teach is performed so that ***XCaliper*** can generate an address map for a new path so that all subsequent calls to Apply to find new edges can be performed quickly.   A Teach need only be performed for AnyAngle or Arc Viewports.

Thickness

The thickness of a Viewport defines the number of pixels over which the profile or projection of a Viewport is calculated. For Horizontal and Vertical Viewports the thickness is defined as the height or the width of the Viewport, respectively. For AnyAngle or Arc Viewports the thickness is defined by the Thickness property of the Edge Tool.

Threshold

A threshold value is used in the process of Segmentation to separate an image into foreground and background. Two threshold values, *UpperThreshold* and *LowerThreshold*, are used to identify the foreground and background. Pixels whose light intensities are above the *UpperThreshold* and below the *LowerThreshold* will be assigned to foreground and background based upon the *BlobType* property of the **BlobAnalyzer**. Pixels whose light intensities lie between the *UpperThreshold* and the *LowerThreshold* are considered as transitional pixels.

Underlay

The video plane that appears on a video monitor underneath another plane when two or more video signals are mixed.

VBX (or Visual Basic Extension)

A form of a Dynamic Linked Library (or DLL) that is callable from Microsoft Visual Basic for Windows. ***XCaliper*** is in the form of a VBX to allow programmers to generate complete applications with powerful graphical user interfaces that are easy to program.

Value

The raw result obtained for a constraint from the application of the edge tool.

VGA Pass Through

A type of Frame Buffer that allows images to be displayed on a VGA computer monitor. The image display appears behind the VGA screen, and by writing a specific color to a window on the screen, the VGA will become transparent in that window, displaying a portion of the Frame Buffer.

Viewport

The region of interest of the Frame Buffer that is defined by the window of the ***XCaliper*** Tool. A Viewport can be described by a Path and Thickness. Viewports can be Horizontal, Vertical, AnyAngle, or along an Arc.

Weight

A relative value placed upon a Constraint that is used by ***XCaliper*** to assign the importance given that Constraint during the evaluation of edges or edge pairs.   The weight is expressed as a percentage, where a greater percentage gives a constraint greater influence in the final Edge Score.   The sum of all weights is 100%.

Weighted Score

A numeric value within the range [0-1] that is a product of the Constraint Score and the Weighting for that constraint. This value is used to generate the Edge Score.

WeightTable

An array which specifies the percentage of each pixel in an image to be attributed to foreground and background based on its intensity.

Glossary

A

Accuracy
AnyAngle
Apply
Arc
ArcCaliper
ArcEdgeLocator
AspectRatio

B

Blob
Blob Analysis
BlobAnalyzer

C

Caliper
Click
ClosedCurve
Constraint
Constraint Score
Container
Convolution
Cumulative Histogram

D

Difference Filter
Discriminator
Drag

E

E1 Position
E1 Strength
E2 Position
E2 Strength
Edge
Edge 1 (or E1)
Edge 2 (or E2)
Edge Evaluation Function (or EEF)
Edge Intensity Graph
Edge Locator
Edge Pair
Edge Score
Edge Strength
EEF Point
EndPoint
Expected Position
Expected Size

F

Feature

Clear Command

Applies To

ImageDevice, ImageHook, MemoryBuffer, LightMeter, BlobAnalyzer

Description

Clears the contents of the Frame Store to a specified color in the area covered by this control.

Syntax

[*form*.]*control*.**Command** [= command$]

Command

"Clear *color* "

Remarks

color should be the value which is to be written to the frame store. This is defined as a Visual Basic color exactly like the *BackColor* property. However, for monochrome Frame Buffers, the value should be in the range (0,255) inclusive.

Example

ImgDevice1.command = "Clear &HFF&"

Set the entire frame buffer to the value &HFF& (255 decimal)

Data Type

String

Close Command

Applies To

ImageDevice, ImageHook, MemoryBuffer, LightMeter, BlobAnalyzer

Description

Performs a morphological close on contents of the Frame Store in the area covered by this control.

Syntax

[*form*.]*control*.**Command** [= command$]

Command

"Close [SourceImage] *Size*"

Remarks

A closing of size n is equivalent to an dilation of size n followed by an erosion of size n. As such, it tends to remove small particles and holes within objects. The Size gives the diameter in pixels of the area over which the operation takes places. Size 1 implies that the erosion and dilation take place over a 3 x 3 neighborhood. Size 2 is 5 x 5, and so on.

The optional SourceImage argument corresponds to the *Tag* property of the desired source. Any ***XCaliper*** tool may be used as the source. If not supplied, the source is taken to be the same as the destination.

The area of the image saved includes all that is under the tool in question. For an **ImageDevice** control, this is the area specified by the *InputPan*, *InputScroll*, *InputSizeX* and *InputSizeY* properties. For a **MemoryBuffer**, it is the entire buffer. For **ImageHook** tools, it is restricted to the visible portion of the buffer while for **BlobAnalyzer** and **LightMeter** tools, it is the area covered by the tools Viewport.

Example

ImgDevice1.command = "Close 2"

Apply a size 2 closing to the entire frame buffer.

Data Type

String

Convolve Command

Applies To

ImageDevice, ImageHook, MemoryBuffer, LightMeter, BlobAnalyzer

Description

The Convolve command performs a convolution on the image area using the kernel specified as an argument.

Syntax

[*form*.]*control*.**Command** [= command$]

Command

"Convolve [*sourceImage*] *kernelWidth kernelHeight elements*... "

Remarks

The Convolve command string defines what kernel should be used for the convolution and what the source image should be. The following is a description of each of the elements of the string:

Setting	Description
sourceImage	Corresponds to the contents of the Tag property of the desired source. Any **BlobAnalyzer**, **LightMeter**, **ImageDevice**, **ImageHook**, or **MemoryBuffer** may be used as the source. As always the value of the *Tag* property must be unique. This argument is optional; the default is make the source the same as the destination.
kernelWidth	The width of the kernel in pixels. Must be a positive odd number. Currently, only 3 x 3, 5 x 5, and 7 x 7 kernels are supported.
kernelHeight	The height of the kernel in pixels. Must be a positive odd number. Currently only 3 x 3, 5 x 5, and 7 x 7 kernels are supported.
elements	A list of the kernel multipliers. The list is given in row-major order from top-left to bottom-right. Of course the number of elements must be equal to kernelHeight x kernelWidth.

XCaliper recognizes most common kernels and uses short cuts to speed up the operation. For example, the vertical edge detector shown in the second example below is among those detected. The program will recognize that the zero-entries in the kernel have no effect on the result and that the others can be performed using simple shifts, negations and adds; multiplications are not necessary.

Another consideration regarding speed of the operation is that it is faster to Convolve from one image to another as opposed to convolving an image back to itself. This is because it is not necessary to buffer the source when convolving from one image to another.

Since the result of any convolution is greater than 8 bits, ***XCaliper*** automatically normalizes the result to fit into the frame buffer using the following rules based on the characteristics of the kernel elements:

Kernel format	Assumed Type	Normalization Method
Sum of the kernel is zero	Edge detector	Take absolute value and clip values greater than 255.
Sum of the kernel is one	Sharpening	Clip values less than zero to 0, and values greater than 255 to 255.
Sum of the kernel is greater than one with no negative values	Averaging	Divide the result by the sum of the kernel values.
Sum of the kernel is greater than one with at least one negative value	Gaussian smoothing	Divide the result by the sum of the kernel values and clip values less than zero to 0, and values greater than 255 to 255.

The file, XCALIPER.TXT, contains definitions for several frequently-used kernels. The designer may wish to use these or construct custom kernels as desired. The following kernels are pre-defined:

Kernel Name	Description	Kernel Elements

VerticalSobel	Vertical edge detector using the Sobel kernel	-1 -2 -1
		0 0 0
		1 2 1

HorizontalSobel	Horizontal edge detector using the Sobel kernel	-1 0 1
		-2 0 2
		-1 0 1

Average	3 x 3 averaging (or smoothing)	1 1 1
		1 1 1
		1 1 1

Average5	5 x 5 averaging (or smoothing)	1 1 1 1 1
		1 1 1 1 1
		1 1 1 1 1
		1 1 1 1 1
		1 1 1 1 1

Average7	7 x 7 averaging (or smoothing)	1 1 1 1 1 1 1
		1 1 1 1 1 1 1
		1 1 1 1 1 1 1
		1 1 1 1 1 1 1
		1 1 1 1 1 1 1
		1 1 1 1 1 1 1
		1 1 1 1 1 1 1

SharpLow	3 x 3 sharpening with a Gaussian neighborhood	1 -2 1
		-2 5 -2
		1 -2 1

Sharp5Low	5 x 5 sharpening with a Gaussian neighborhood	-8 -5 -2 -5 -8
		-5 3 9 3 -5
		-2 9 33 9 -2
		-5 3 9 3 -5

		-8 -5 -2 -5 -8

Laplacian	3 x 3 omni-directional edge detector	-1 -1 -1
		-1 8 -1
		-1 -1 -1

Laplacian5	5 x 5 edge detector with a Gaussian neighborhood	-8 -5 -2 -5 -8
		-5 3 9 3 -5
		-2 9 32 9 -2
		-5 3 9 3 -5
		-8 -5 -2 -5 -8

Sharpening	3 x 3 sharpening with an averaging neighborhood	-1 -1 -1
		-1 9 -1
		-1 -1 -1

Sharp5	5 x 5 sharpening with an averaging neighborhood	-1 -1 -1 - 1 -1 -1 -1 -1 -1 -1 -1 -1 25 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
Sharp7	7 x 7 sharpening with an averaging neighborhood	-1 -1 -1 - 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 49 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

Examples

ImgDevice1.command = "Convolve " + *Membuf1*.Tag + *Sharpen*

Convolve the image contained in *Membuf1* using the kernel named in the global constant Sharpen and place the result into *ImgDevice1*.

ImgDevice1.command = "Convolve 3 3 -1 -2 -1 0 0 0 1 2 1"

Convolve the image contained in *ImgDevice1* using the specified 3 x 3 kernel and place the result back into *ImgDevice1*.

Data Type

String

Copy Command

Applies To

ImageDevice, ImageHook, MemoryBuffer, LightMeter, BlobAnalyzer

Description

Copies the contents of another control into the area of the Frame Store covered by the current control.

Syntax

[*form*.]*control*.**Command** [= command$]

Command

"Copy *sourceImage* "

Remarks

sourceImage should be the value of the Tag property of the control which is to be copied.

The area of the image copied includes all that is under the tool in question. For an **ImageDevice** control, this is the area specified by the *InputPan*, *InputScroll*, *InputSizeX* and *InputSizeY* properties. For a **MemoryBuffer**, it is the entire buffer. For **ImageHook** tools, it is restricted to the visible portion of the buffer while for **BlobAnalyzer** and **LightMeter** tools, it is the area covered by the tools Viewport. If copying an arbitrary area within a buffer is desired, it is recommended that a **LightMeter** be used for the task as it uses fewer resources than a **BlobAnalyzer**.

Example

ImgDevice1.command = "Copy " + + *Membuf1*.Tag

Data Type

String

Dilate Command

Applies To

ImageDevice, ImageHook, MemoryBuffer, LightMeter, BlobAnalyzer

Description

Performs a morphological dilation on the contents of the Frame Store in the area covered by this control.

Syntax

[*form*.]*control*.**Command** [= command$]

Command

"Dilate [SourceImage] *Size* "

Remarks

A dilation replaces each pixel in the Frame Buffer with the maximum value of the pixels in its neighborhood. The Size gives the diameter of the neighborhood in pixels. Thus, a Size 1 dilation replaces each pixel by the maximum value over a 3 x 3 neighborhood (itself and the eight pixels which touch it). Size 2 implies that the maximum value over a 5 x 5 neighborhood (25 nearest pixels) is taken.

The optional SourceImage argument corresponds to the *Tag* property of the desired source. Any ***XCaliper*** tool may be used as the source. If not supplied, the source is taken to be the same as the destination.

Example

ImgDevice1.command = "Dilate 2

Apply a size 2 dilation to the entire frame buffer

Data Type

String

Erode Command

Applies To

ImageDevice, ImageHook, MemoryBuffer, LightMeter, BlobAnalyzer

Description

Performs a morphological erosion on the contents of the Frame Store in the area covered by this control.

Syntax

[*form*.]*control*.**Command** [= command$]

Command

"Erode [SourceImage] *Size* "

Remarks

A erosion replaces each pixel in the Frame Buffer with the minimum value of the pixels in its neighborhood. The Size gives the diameter of the neighborhood in pixels. Thus, a Size 1 dilation replaces each pixel by the maximum value over a 3 x 3 neighborhood (itself and the eight pixels which touch it). Size 2 implies that the maximum value over a 5 x 5 neighborhood (25 nearest pixels) is taken.

The optional SourceImage argument corresponds to the *Tag* property of the desired source. Any ***XCaliper*** tool may be used as the source. If not supplied, the source is taken to be the same as the destination.

Example

ImgDevice1.command = "Erode 2

Apply a size 2 erosion to the entire frame buffer

Data Type

String

LoadFile Command

Applies To

ImageDevice, ImageHook, MemoryBuffer, LightMeter, BlobAnalyzer

Description

Loads an image file from a disk or diskette into the image memory. Currently, the only format supported is Windows .BMP.

Syntax

[*form*.]*control*.**Command** [= command$]

Command

"LoadFile *filename* "

Remarks

This command reads an image from storage media (e.g. hard or floppy disks) into the Frame Store of the specified control. The image will be placed in the frame store starting at the point in the Frame Buffer underneath the upper left corner of the control. It will be clipped to the limits of that control (exception: reading an image into a MemoryBuffer with the AutoSize property set to True will cause the size of the buffer to be changed to that of the image).

In a Future release, ***XCaliper*** will support a Viewport tool designed to process an arbitrary window in the Frame Buffer. In the current version, it is suggested that a **LightMeter** be used instead.

The area of the image loaded includes all that is under the tool in question. For an **ImageDevice** control, this is the area specified by the *InputPan*, *InputScroll*, *InputSizeX* and *InputSizeY* properties. For a **MemoryBuffer**, it is the entire buffer. For **ImageHook** tools, it is restricted to the visible portion of the buffer while for **BlobAnalyzer** and **LightMeter** tools, it is the area covered by the tools Viewport. If loading a window within a buffer is desired, it is recommended that a **LightMeter** be used for the task as it uses fewer resources than a **BlobAnalyzer**.

Data Type

String

Median Command

Applies To

ImageDevice, ImageHook, MemoryBuffer, LightMeter, BlobAnalyzer

Description

Performs a median filter on the contents of the Frame Store in the area covered by this control.

Syntax

[*form*.]*control*.**Command** [= command$]

Command

"Median width height "

Remarks

A median filter replaces each pixel in the frame buffer with the median value of the pixels in its neighborhood. A median filter has the desirable effect of decreasing noise in the image without blurring edges.

The width and height specify the neighborhood over which the filter is applied. Both must be odd numbers greater than 3.

This filter uses a proprietary algorithm which is quite fast. Even large filters can be applied in an acceptable amount of time.

The area of the image processed includes all that is under the tool in question. For an **ImageDevice** control, this is the area specified by the *InputPan*, *InputScroll*, *InputSizeX* and *InputSizeY* properties. For a **MemoryBuffer**, it is the entire buffer. For **ImageHook** tools, it is restricted to the visible portion of the buffer while for **BlobAnalyzer** and **LightMeter** tools, it is the area covered by the tools Viewport. If loading a window within a buffer is desired, it is recommended that a **LightMeter** be used for the task as it uses fewer resources than a **BlobAnalyzer**.

Example

ImgDevice1.command = "Median 7 7

Apply a 7 x 7 median to the entire frame buffer

Data Type

String

Open Command

Applies To

ImageDevice, ImageHook, MemoryBuffer, LightMeter, BlobAnalyzer

Description

Performs a morphological opening on the contents of the Frame Store in the area covered by this control.

Syntax

[*form*.]*control*.**Command** [= command$]

Command

"Open [SourceImage] *Size* "

Remarks

A opening of size n is equivalent to an erosion of size n followed by a dilation of size n. As such, it tends to disconnect joined objects and to enlarge holes. The Size gives the diameter in pixels of the area over which the operation takes place. Size 1 implies the erosion and dilation take place over a 3 x 3 neighborhood. Size 2 is 5 x 5, and so on.

The optional SourceImage argument corresponds to the *Tag* property of the desired source. Any ***XCaliper*** tool may be used as the source. If not supplied, the source is taken to be the same as the destination.

The area of the image processed includes all that is under the tool in question. For an **ImageDevice** control, this is the area specified by the *InputPan*, *InputScroll*, *InputSizeX* and *InputSizeY* properties. For a **MemoryBuffer**, it is the entire buffer. For **ImageHook** tools, it is restricted to the visible portion of the buffer while for **BlobAnalyzer** and **LightMeter** tools, it is the area covered by the tools Viewport.

If processing an arbitrary area within a buffer is desired, it is recommended that a **LightMeter** be used for the task as it uses fewer resources than a **BlobAnalyzer**.

Example

ImgDevice1.command = "Open 2

Apply a size 2 opening to the entire frame buffer

Data Type

String

SaveFile Command

Applies To

ImageDevice, ImageHook, MemoryBuffer, LightMeter, BlobAnalyzer

Description

Save an image from the image memory on to a disk or diskette.

Syntax

[*form*.]*control*.**Command** [= command$]

Command

"SaveFile *FileName* "

Remarks

This command saves a copy of the image from the specified control to storage media (e.g. hard or floppy disks).

The area of the image saved includes all that is under the tool in question. For an **ImageDevice** control, this is the area specified by the *InputPan*, *InputScroll*, *InputSizeX* and *InputSizeY* properties. For a **MemoryBuffer**, it is the entire buffer. For **ImageHook** tools, it is restricted to the visible portion of the buffer while for **BlobAnalyzer** and **LightMeter** tools, it is the area covered by the tools Viewport. If loading a window within a buffer is desired, it is recommended that a **LightMeter** be used for the task as it uses fewer resources than a **BlobAnalyzer**.

Data Type

String

TuneInput Command

Applies To

ImageDevice, ImageHook, LightMeter, BlobAnalyzer

Description

Automatically adjusts the gain and offset of the underlying **ImageDevice** in an attempt to obtain an optimal setting.

Syntax

[*form*.]*control*.**Command** [= command$]

Command

"TuneInput"

Remarks

This command attempts to optimize the distribution of image intensity values within the tools Viewport by inspecting its Histogram and then adjusting the gain and offset of the underlying **ImageDevice** appropriately.

Several successive images will be acquired and the information re-examined after each one. Therefore in order to work, the tool must be placed on an **ImageDevice** (not a **MemoryBuffer**) over the grab window.

For the purposes of this command, optimal refers to the histogram range being set as large as possible while ensuring there is no signal saturation.

Data Type

String

Applied Event

Applies To

LightMeter, LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator, BlobAnalyzer

Description

Occurs after a control is applied.

Syntax

Sub *control*_**Applied** (*Index* **As Integer**)

Remarks

The argument *Index* uniquely identifies a control that belongs to a control array. Typically, the *Applied* event is used to detect automatic execution of the tool when moved or sized. However, the event will occur every time the tool is applied whether through programmer or user action.

Click Event

Applies To

LightMeter, LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator, BlobAnalyzer

Description

Occurs when the user presses and then releases a mouse button over an object. It may also occur when the value of a control is changed.

Syntax

Sub *ctlname*_**Click** (*Index* **As Integer**)

Sub BlobAnalyzer_**Click**(*Index* **As Integer**, *BlobNumber* **As Integer**)

Remarks

The argument *Index* uniquely identifies a control if it is in a control array. Typically, you attach a **Click** procedure to a control to carry out commands and command-like actions.

The argument *BlobNumber* supplies the number of the blob over which the action took place in the range 0 to **BlobTool**.**Count** - 1. If the action took place on the background of the image, not over a blob, then the *BlobNumber* will be -1.

Note: To distinguish between the left, right, and middle mouse buttons, use the **MouseDown** and **MouseUp** events.

DblClick Event

Applies To

LightMeter, LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator, BlobAnalyzer

Description

Double-clicking on a tools Viewport will cause a *DblClick* event to occur. For the edge tools and **BlobAnalyzer**, double-clicking on the Viewport will also invoke the appropriate Setup Dialog boxes, if the *UserInterface* property is set to SetupOnly or Enable All.

Syntax

Sub *control*_**DblClick** (*Index* **As Integer**)

Sub BlobAnalyzer_**DblClick**(*Index* **As Integer**, *BlobNumber* **As Integer**)

Remarks

The argument *Index* uniquely identifies a control that belongs to a control array. The *BlobNumber* argument is added in the **BlobAnalyzer** tool. It identifies the number of the blob over which the **DblClick** took place. It will be -1 if the click took place on the background instead.

If the *UserInterface* property is set to cause a dialog box to appear on double-click, then the box will be displayed before the DblClick event occurs. Refer to the Visual Basic Language Reference for a more detailed explanation of the *DblClick* event.

MouseDown, MouseUp Events

Applies To

LightMeter, LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator, BlobAnalyzer,

Description

Occur when the user presses (MouseDown) or releases (MouseUp) a mouse button.

Syntax

Sub *ctlname*_**MouseDown** ([Index **As Integer**,]Button **As Integer**, Shift **As Integer**, X **As Single**, Y **As Single**)

Sub *ctlname*_**MouseUp** ([Index **As Integer**,]Button **As Integer**, Shift **As Integer**, X **As Single**, Y **As Single**)

Sub BlobAnalyzer_MouseDown ([Index As **Integer**,]Button **As Integer**, Shift **As Integer**, X **As Single**, Y **As Single**, *BlobNumber* **As Integer**)

Sub BlobAnalyzer_MouseUp ([Index **As Integer**,]Button **As Integer**, Shift **As Integer**, X **As Single**, Y **As Single**, *BlobNumber* **As Integer**)

Remarks

These events work the same as the standard mouse events except for the case of the **BlobAnalyzer** tool, which has the additional parameter, *BlobNumber.* The *BlobNumber* is an integer that specifies the currently selected blob in the image. If the background is selected, the value of *BlobNumber* is defined to be -1. This can be used to create an interactive application with blob data being reported according to where the user is pointing the mouse.

The **MouseDown** and **MouseUp** events use these arguments:

Argument	**Description**
Index	Uniquely identifies a control if it is in a control array.
Button	The button was pressed (**MouseDown**) or released (**MouseUp**) to cause the event. The *Button* argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2)values 1, 2, and 4, respectively. Only one of the bits is set, indicating which button caused the event.
Shift	The state of the Shift, Ctrl and Alt keys when the button specified in the *Button* argument was pressed or released. A bit is set if the key is down. The *Shift* argument is a bit field, with the least-significant bits corresponding to the Shift key (bit 0), the Ctrl key (bit 1), and the Alt key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. *Shift* indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys is pressed. For example, if both Ctrl and Alt were pressed, the value of *Shift* would be 6.
X, Y	The current location of the mouse pointer. *X* and *Y* are always expressed in terms of the coordinate system set by the ScaleHeight, Scale Width, ScaleLeft, and ScaleTop properties of the container.
BlobNumber	Specifies the index of the blob that was under the mouse pointer when the event occured. This only applies to the **BlobAnalyzer**.

Use a **MouseDown** or **MouseUp** procedure to specify actions to occur when a given mouse button is pressed or released. Unlike the **Click** and **DblClick** events, **MouseDown** and **MouseUp** events allow you to distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the Shift, Ctrl, and Alt keyboard modifiers.

The following applies to both **Click** and **DblClick** events:

If a mouse button is pressed while the pointer is over a form or control, that object "captures" the mouse and receives all mouse events up to and including the last **MouseUp** event. This implies that the **X**, **Y** mouse-pointer coordinates given by a mouse event may not always be in the client area of the object that receives them.

If mouse buttons are pressed in succession, the object that captures the mouse after the first press receives all mouse events until all buttons are released.

If you need to test for the Button or Shift arguments, you can declare constants that define the bits within the argument by loading the CONSTANT.TXT file into a module.

These mouse button constants have the following values:

Constant	**Value**
LEFT_BUTTON	1
RIGHT_BUTTON	2
MIDDLE_BUTTON	4
SHIFT_MASK	1
CTRL_MASK	2
ALT_MASK	4

These constants are in CONSTANT.TXT. The constants then act as bit masks you can use to test for any combination of buttons without having to figure out the unique bit field value for each combination.

Note: You can use a **MouseMove** procedure to respond to an event caused by moving the mouse. The *Button* argument for **MouseDown** and **MouseUp** differs from the *Button* argument used for **MouseMove**. For **MouseDown** or **MouseUp**, the *Button* argument indicates exactly one button per event; for **MouseMove**, it indicates the current state of all buttons.

MouseMove Event

Applies To

LightMeter, LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator, BlobAnalyzer,

Description

Occurs when the user moves the mouse.

Syntax

Sub *ctlname*_**MouseMove** ([*Index* **As Integer**,] *Button* **As Integer**, *Shift* **As Integer**, *X* **As Single**, *Y* **As Single**)

Sub BlobAnalyzer_MouseMove ([*Index* **As Integer**,] *Button* **As Integer**, *Shift* **As Integer**, *X* **As Single**, *Y* **As Single**, *BlobNumber* **As Integer**)

Remarks

The **MouseMove** event uses these arguments:

Argument	**Description**
Index	Uniquely identifies a control if it is in a control array.
Button	The state of the mouse buttons, in which a bit is set if the button is down. The *Button* argument is a bit field, with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2)values 1, 2, and 4, respectively. It indicates the complete state of the mouse buttons; some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons is pressed.
Shift	The state of the Shift, Ctrl and Alt keys. Bit is set if the key is down. The *Shift* argument is a bit field, with the least-significant bits corresponding to the Shift key (bit 0), the Ctrl key (bit 1), and the Alt key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. *Shift* indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys is pressed. For example, if both Ctrl and Alt were pressed, the value of *Shift* would be 6.
X ,Y	The current location of the mouse pointer. *X* and *Y* are always expressed in terms of the coordinate system set by the ScaleHeight, Scale Width, ScaleLeft, and ScaleTop properties of the container.

The **MouseMove** event is generated continually as the mouse pointer moves across objects. Unless another object has captured the mouse, an object recognizes a **MouseMove** event whenever the mouse position is within its borders.

If you need to test for the *Button* or *Shift* arguments, you can declare constants that define the bits within the argument by loading the CONSTANT.TXT file into a module. These mouse button constants have the following values:

Constant	**Value**
LEFT_BUTTON	1
RIGHT_BUTTON	2
MIDDLE_BUTTON	4
SHIFT_MASK	1
CTRL_MASK	2

ALT_MASK	4

These are in CONSTANT.TXT.   The constants act as bit masks you can use to test for any combination of buttons without having to figure out the unique bit field value for each combination.

You test for a condition by first assigning each result to a temporary integer variable and then comparing the Button or Shift arguments to a bit mask.   Use the And operator with each argument to test if the condition is greater than zero, indicating the key or button is pressed. For example:

```
LeftDown = (Button And LEFT_BUTTON) > 0

CtrlDown = (Shift And CTRL_MASK) > 0
```

Then, in a procedure, you can test for any combination of conditionsfor example:

```
If LeftDown And CtrlDown Then
```

Note: You can use **MouseDown** and **MouseUp** event procedures to respond to events caused by pressing and releasing mouse buttons.

The *Button* argument for **MouseMove** differs from the *Button* argument for **MouseDown** and **MouseUp**. For **MouseMove**, the *Button* argument indicates the current state of all buttons; a single **MouseMove** event can indicate that some, all, or no button is pressed.   For **MouseDown** or MouseUp, the Button argument indicates exactly one button per event.

Any time you move a window inside a **MouseMove** event, it can cascade.   **MouseMove** events are generated when the window moves underneath the pointer.   A **MouseMove** event can be generated even if the mouse is perfectly stationary.

XCaliper Version 1.6 Release Notes

PLEASE READ THE CONTENTS OF THIS DOCUMENT PRIOR TO THE USE OF *XCaliper*.

INDEX

Introduction

Welcome to the ***XCaliper*** software library. These release notes present information about new features, enhancements and fixes in this release. It also descibes information which is specific to individual frame grabbers.

Release 1.6 is a minor release of ***XCaliper.*** The main reason for this release is to add support for new board drivers. In addition it incorporates some bug fixes, APIs and an emulation of continuous grabbing on the VGA monitor.

Contents of the *XCaliper* Package

The complete ***XCaliper*** package includes four 3.5" diskettes, a software key, and a manual.   The ***XCaliper*** distribution diskettes include the following list of files:

- ***XCaliper*** Visual Basic extensions (VBXs)
- Supported frame grabber drivers
- Visual Basic source files for the tutorial
- Visual Basic source files for a sample application
- Blob and edge tools demonstrations
- Visual Basic source files for the demonstrations
- ***Xiris*** *Image Processor (**XIP)*** software
- Bit mapped images
- Help files

New Features

XIP

XIP, Xiris Image Processor, is an application built using the ***XCaliper*** image processing library. **XIP** is not intended to solve any applications. Instead, it has two purposes:

1) a demonstration tool to illustrate the capabilities of the VBX library and the power of applications built around a combination of ***XCaliper*** and other Visual Basic Extensions;
2) a prototyping tool used to exercise the tool kit, and to see how it reacts to images of actual vision problems.

XIP can be used to acquire images of actual parts, to analyze them and to see how the ***XCaliper*** tools can be used to extract measurements. Any supported frame grabber can be used. It is also possible to work without a frame grabber, using system memory and the system display instead.

Continuous Grab Emulation

It is now possible to perform continuous grabbing even when the hook's *Transparent* property is set to NonTransparent. This is done by repeatedly copying from the frame buffer into the VGA memory.

The way that the system reacts to a request for continuous grab in non-transparent mode depends on the setting of the **ImageDevice** *ContGrabType* property. See the manual for further details.

Note that in order to support this new mode, the Bitflow Raptor driver has been updated. This is because this driver incorporated its own emulation of continuous grabbing. This internal mode has been discarded. Other drivers are not affected.

New Commands

XCaliper has significantly expanded the list of image processing algorithms supported. As well as convolutions, it now supports median filters and the morphological operations of opening, closing, erosion and dilation. The median filter uses a proprietary algorithm which is extremely fast. Do not be afraid to use it even with very large neighborhoods (say, 20 x 20).

New Properties

Property	**Description**
ActiveControl	Returns the tag value of the currently selected ***XCaliper*** control.
ContGrabType	Selects how the system will react to an attempt to enable continuous grabbing in non-transparent mode.

New APIs

Four new APIs are included in this version. They are:

API	**Description**
ImgOpGetDib	obtain a Device-independent Bitmap representation of a region of interest in a

	frame buffer
ImgOpReleaseDib	release access to a DIB obtained through a call to ImgOpGetDib
ImgOpGetDib	obtain a Device-independent Bitmap representation of a region of interest in a frame buffer
ImgOpGetDib	obtain a Device-independent Bitmap representation of a region of interest in a frame buffer

Known Bugs and Unimplemented Features

Controls Don't Zoom

The controls in a container do not zoom when the underlying device has its Zoom properities change. This is largely a visual representation problem, hence switching between zoom and non-zoom modes at different points in the application may be required.

Driver Notes

XCaliper drivers are kept in DLLs which conform to a special interface specification and which are loaded by the ***XCaliper*** image manager (xlibimg.dll). Xiris is in the process of expanding the list of supported hardware. As a consequence new drivers may be available at any given time. This section gives some information about special considerations associated with each of the drivers currently supported.

The following list is ordered alphabetically by company name and board name.

Bitflow
Coreco
Dipix
Imaging Technologies
Matrox
Sharp

Bitflow

Bitflow
21-G Olympia Avenue, #80
Woburn, MA
01801
Tel: (617) 932-2900
FAX (617) 933-9965

Raptor -------- xdrvrap.dll

There are no special considerations to using the Raptor with ***XCaliper.*** Assuming that the Bitflow demo programs work and that the installation instructions on pages 13-14 of the Bitflow installation and User Guide have been followed, the system should be ready to use with ***XCaliper***.

The Bitflow card has seven inputs. The *InputChannel* property of the **ImageDevice** maps these as follows:

0	Analog Input 0
1	Analog Input 1
2	Analog Input 2
3	Analog Input 3
4	8-bit Digital Input
5	16-bit Digital Input
6	32-bit Digital Input

This property must match the value selected in the Bitflow syscon program. For example, if you are using a digital camera, then a digital input must be selected.

Coreco

Coreco
6969 Trans Canada Highway, Suite 113
Ville St-Laurent, Quebec
H4T 1V8
Tel: (514) 333-1301
US Toll free: 1-800-361-4997

This company provides a DOS TSR (Terminate and Stay Resident) program which interfaces to the board. In order to use XCaliper with Coreco hardware, the TSR must be installed.

Use of memory-mapped access to the frame buffer is recommended as it is significantly faster. If this is done, the memory must be enabled on the bus using the Coreco setup program. ***XCaliper*** will automatically recognize that the memory has been enabled and where. Your system.ini should contain an EMMExclude statement which prevents Windows from using the associated bus addresses. If you are using emm386 or an equivalent memory manager, it should suppress usage of these addresses as well.

Coreco claims that their device drivers provide a consistent interface across their entire family. In practice they don't quite achieve this although they come close. As a consequence, our driver may work with any card in the product line. It has been tested with the TCX, MX and F-64. It is not likely to work with other boards (although this is possible) but a port should be simple and easy.

F-64 ---------- xdrvtcx.dll
Use with a 640 x 480, 800 x 600 or 1024 x 768 Super VGA in either 16-color or 256-color mode.

The F-64 supports an assortment of special cameras. See the Coreco documentation for a complete list. The correct camera is chosen using the setup program. The driver will simply use the camera chosen by the setup program. However there may be some incompatibilities and we do not guarantee that the driver will work with untested cameras although it should. Currently only the Kodak MegaPlus 1.4 and 4.2 have been tested.

MX ------------ xdrvtcx.dll
Use with a 640 x 480, 800 x 600 or 1024 x 768 Super VGA in either 16-color or 256-color mode. Ensure that the frequencies set up by your Super VGA setup program are compatible with those of the frame grabber. Compare the values specified by your Oculus setup program with those of the VGA card.

This card does not corectly support output lookup tables in VGA overlay mode. By default, Coreco turns them off. You should leave the settings this way and not access the output lut in code.

TCX ----------- xdrvtcx.dll
The driver assumes that if you are using a TCX, it is because you want to grab color images. The driver does not work correctly with monochrome input, Use with a 640 x 480, 800 x 600 or 1024 x 768 Super VGA in either 16-color or 256-color mode.

This card has no output LUTs. Any attempt to change the LUTs in ***XCaliper*** will thus fail.

Dipix

Dipix
1051 Baxter Road
Ottawa Ontario
K2C 3P1
Tel: (613) 596-4942
FAX: (613) 596-4914
US Toll Free: 1-800-724-5929

P360F Power Grabber ----------- xdrvp360.dll

The P360 is memory-mapped and the memory must be enabled on the bus. Your system.ini should contain an EMMExclude statement which prevents Windows from using the associated bus addresses. If you are using emm386 or an equivalent memory manager, it should suppress usage of these addresses as well.

Dipix has the notion of a p360 directory which is set up using the DIPP360F environment variable as, for example, : DIPP360F=c:\p360. An assortment of board-specific files are kept in this directory. The ***XCaliper*** driver follows this convention. Board-specific files should be kept in the places that DIPIX expects them to be.

The P360 comes with an on-board DSP capable of high-speed calculations. ***XCaliper*** uses this DSP to perform histograms and a portion of the edge extraction code.

XCaliper.ini has the following entries for the Dipix card described in the following paragraphs:

```
    MemoryBase
    IOBase
    CameraType
    DpaFile
    TagFile
    IsTransparent
```

This card does not have any configuration program so you must supply the base addresses to the program as well. Two entries in the [DipixP360] section of the file, MemoryBase and IOBase are used to supply these numbers.

The P360 supports a variety of special cameras. You specify the camera you wish to use by naming its .cdt file in xcaliper.ini as:

```
            [DipixP360]
            CameraType = RS170.CDT
```

RS170.CDT is the default.

It is also possible to name the tag and display parameters files (.tag and .dpa files) used with entries in the [DipixP360] section. The defaults are:

```
            TagFile = A.TAG
            DpaFile = WIND103.DPA
```

The .TAG file contains the firmware which is downloaded to the board's DSP. It is unlikely that you will want to change this but it is possible if you are developing your own DSP code.

The .DPA file is used to describe the characteristics of the VGA card in your system. This file is only needed if you are using the video mixer daughter board It must be generated using the DIPIX Windows demo program. To do this perform the following:

1. Modify the file DIPP360F\ DEMOWIND\DEMOWIND. DAT to reflect your system configuration.
2. Start the demo. When it asks you to wait, push the OK button instead.
3. If the image looks wrong, execute the <GenerateDpa> item in the <Setup> menu.
4. If it is not already on the screen, Execute the <TestImage> entry in the <Setup> to put up the Dipix test image .
5. Use the scroll bars on the window to align the VGA and the image screens. The image is correct when you see a single line across the left and top side of the image with tick marks emanating from it.
6. Whether or not you saved a DPA file at step 3, do it now.
7. Enter the name of the file generated here in XCaliper.ini.

The final entry in the ini file is IsTransparent. If set to False, ***XCaliper*** will refuse to allow use the video mixer card even if one is present. By default, ***XCaliper*** will allow use of the daughter board (eg. will allow the Transparent property to be set to LockedWindow and FloatingWindow) if one is present. It will not allow the Transparent property to be set this way otherwise.

Imaging Technologies

Imaging Technology Incorporated
55 Middlesex Turnpike
Bedford MA
01730-1421
Tel: (617) 275-2700
FAX: (617) 275-9590

IC-PCI ---- xdrvicp.dll
IC-VL ---- xdrvicvl.dll

The drivers for these two boards work in the same way. Start by installing the board and its software according to ITI's instructions. Currently only the AM-VS acquisition module is supported.

The ITI AMVS module has a unique idea of how the input channel mechanisms work which may be useful in certain applications. It is possible to set different gains, offsets and even cameras on different input ports. ***XCaliper*** supports this notion. When the gain or offset is set, it only takes effect on the current input channel. Hence if you want the same gain on all channels, you will have to set it four times, switching channels between settings.

The IC-PCI (or IC-VL) section of xcaliper.ini contains entries used to select the input camera for each of the ports. These are: AmvsCamera, AmvsCamera0, AmvsCamera1, AmvsCamera2, and AmvsCamera3. AmvsCamera gives the default camera type used for the board. The numbered entries specify the camera to use for a specific input port and will override the setting in AmvsCamera. The camera names used in this section are taken from Table B-1 in the AM-VS software manual Appendix B. At this writing, the possible values are:

DEF_RS512P	RS-170	512 x 482
DEF_RS640P	RS-170	640 x 482
DEF_RS752P	RS-170	752 x 482
DEF_CC512P	CCIR	512 x 572
DEF_CC768P	CCIR	768 x 572
DEF_CC736P	CCIR	732 x 572

Should ITI decide to add additional camera types, these will automatically be supported by ***XCaliper*** as it does not interpret these names; it simply passes them on to the ITI software.

The default value for a camera is DEF_RS640P as:

```
AmvsCamera=DEF_RS640P
```

Matrox

MATROX
1055 St Regis Blvd
Dorval Quebec
H9T 2T4
Tel: (514) 685-2603
FAX (514) 685-2853
US Toll Free: 1-800-4MATROX

PIP series ---- xdrvpip.dll
This family contains three cards: the PIP-512, PIP-1024, and the PIP-640. The driver has only been tested with the PIP-640 but should work with all three cards. This is an older card and as a consequence the functionality is limited. Some of the limitations are:

The board only zooms by 1 or 2. Furthermore all four zooms (InputZoomX, InputZoomY, OutputZoomX, and OutputZoomY) are locked together. Changing one will change all four. Pan is limited to the nearest 8 pixels; scroll to the nearest 16. The input and output properties are locked together. It is not possible to change any of the input or output window sizes or positions.

The PIP series is a dream for compatibility. The card never conflicts with anything.

IP-8 ---------- xdrvip8.dll

The IP-8 is memory-mapped and the memory must be enabled on the bus. Your system.ini should contain an EMMExclude statement which prevents Windows from using the associated bus addresses. If you are using emm386 or an equivalent memory manager, it should suppress usage of these addresses as well.

There is a DOS device driver, ip8drv.sys,. The driver cannot know whether the board is being used in VGA overlay mode or not. Given that, an entry is required in xcaliper.ini to set the overlay mode. We assume that it is normal to ***XCaliper*** in overlay mode so this is the default. If you are running in non-transparent mode the following should be
added to your xcaliper.ini:

```
        [IP8]
        IsTransparent=False
```

In order to run the board in non-transparent mode, the MATROX windows display driver, ip8_640.drv, should be installed. It is unlikely that the CCIR version will work but this has never been tested and maybe we could get a surprise.

Sharp

Sharp
16841 Armstrong Ave.
Irving, CA 92714
Tel: (714) 261-6224
FAX: (714) 261-9321

GPB-1---------- xdrvgpb1.dll

The GPB-1 is I/O-mapped with a single entry in the 10-bit I/O space. As a consequence there is almost no way to have a conflict using this card. The only consideration is the board-select address which is given by the BaseAddress entry in the [GPB] section of your xcaliper.ini file. The default is 0 and there should be no reason to change this.

The GPB-1 requires use of the vgapal.drv Windows display driver to order to run in pass-through mode. This driver can be found on any Microsoft bulletin board. Alternately it can be had through either Sharp or Xiris. Because of the limitations of this driver, it is not possible to mix pass-through and VGA-only windows on the same screen. As a consequence, you cannot display the contents of a MemoryBuffer either, except by copying it on to the GPB-1.

Entries in the [GPB] section of xcaliper.ini include:

IsTransparent Specifies the mode of operation of the frame grabber. Valid entries are True or False. The Sharp GPB1 must have the single monitor auxiliary card installed to operate in transparent mode. The ImageHook VBX can only be used in the transparent mode specified by this setting due to limitations of the Sharp GPB. Default value is True.

BaseAddress Specifies the canotical base address of the frame grabber. Valid entries are in the range of 0 to 7. Default value is 0.

Sync Specifies the video multiplexer used for sync input. Valid entries are in the range of 0 to 3. Default value is 0.

SyncThres Specifies the sync threshold level. Valid entries include: 50, 75, 100 and 125. Default value is 100.

ColorCode Specifies the color code that is used for transparency keying when the frame grabber is used in transparent mode. A color code of 9 will be used as the default if this item is not specified.

About Property

Applies To

ImageDevice ImageHook, MemoryBuffer, LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator, BlobAnalyzer, LightMeter tool

Description

Displays information about the tool in a dialog box.

Remarks

The property is available at design time only. It is invoked by double-clicking on the property in the Properties window. Among the most important information displayed is the version number of the software and serial number of the protection key.

AcceptColor Property

Applies To

BlobAnalyzer

Description

Specifies the display color for blobs that pass subsequent filtering operations.

Syntax

[*form*.]*blobtool*.**AcceptColor** [= &Hcolorvalue&]

Remarks

This property, along with the *AcceptDisplay* property, determines how blobs that pass subsequent apply filtering operation will be displayed on the image display. If *AcceptDisplay* is set appropriately, *AcceptColor* sets the color used to display the acceptable blobs in the image. Setting the property is identical to setting other standard Visual Basic color properties. Refer to the Visual Basic Programmer's Guide for more information on color properties and how to set them.

Data Type

Long Integer

AcceptDisplay Property

Applies To

BlobAnalyzer

Description

Specifies how the **BlobAnalyzer** alters the overlay when displaying blobs that pass subsequent Filtering operations (invoked through the Apply property).

Syntax

[*form*.]*blobtool*.**AcceptDisplay** [= displaytype%]

Remarks

The following are the possible values for the *AcceptDisplay* property:

Setting	**Description**
0 (`NO_CHANGE`)	The color of the blobs remains unchanged from what it was before the application of the tool.
1 (`BD_CLEAR`)	The blobs' overlay display is cleared such that only the underlying original image is visible.
2 (`BD_COLOR`)	The blobs are set to the color specified in the *AcceptColor* property.

Data Type

Integer (Enumerated)

ActiveControl Property

Applies To

ImageDevice, ImageHook, MemoryBuffer

Description

Returns the tag of the currently selected control.

Syntax

[*form*.]*control*.**ActiveControl**

Remarks

The value of the property is not available at design time and is read-only at run time.   The result is the contents of the *Tag* property of the ***XCaliper*** control which has the input focus.   This is the control which is displayed on the screen with picl points highlighted.

Data Type

String

Angle Property

Applies To

ArcEdgeLocator

Description

Specifies the angle of the edge indicated by the *ResultIndex* property.

Syntax

[*form*.]*control*.**Angle**

Remarks

The *Angle* property is not available at design time and is read-only at run time. The result is the angle of the vector, from the center of the closed curve to the point on the curve where the edge was detected, with repect to the positive x-axis. The value is reported in Engineering degrees (0 is at 3 oclock and values increase in a counter-clockwise direction).

Data Type

Single

Apply Property

Applies To

LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator, BlobAnalyzer, LightMeter tool

Description

Causes the control to be applied on the image.

Syntax

[*form*.]*control*.**Apply** [=value%]

Remarks

The *Apply* property is not available at design time but can be read from or written to at run time; either operation results in the tool processing the image data in its Viewport. After processing is complete, the *ElapsedTime* property is changed to reflect the amount of time taken to perform the operation. For a detailed description of the processing performed by each tool, see the appropriate section below. The exact meaning of the return value of the property and the value written to the property, depend on the type of tool being employed.

LightMeter

The **LightMeter** ignores any values passed to it as a parameter and returns the area of its Viewport in pixels.

Caliper and EdgeLocator Tools

The edge tools (**Calipers** and **EdgeLocators**) ignore any value passed to them. If the *TeachMode* property for an arbitrary-angle edge tool is set to `TM_DONT_TEACH`, and the current state of the tool does not correspond to that which was last taught, then any explicit access to the *Apply* property will result in an error message. In this situation, implicit accesses caused by moving the tool or when *ApplyOnChange* is set to True are ignored.

BlobAnalyzer

The value written to the tools *Apply* property is used to determine the type of operation to perform. There are 3 major steps in a complete *Apply* operation that are user selectable: image segmentation; labelization; and extracting the blob features. (A fourth step, filtering, is not user-selectable as it always happens). Precisely which steps are actually performed by the tool depends on the value passed to the *Apply* property.

The following table lists the possible values for the **BlobAnalyzer**'s *Apply* property:

Setting	Description
0 (`AM_SEGMENT_NO_LABEL`)	Segment the image, do not perform any labeling, then analyze and filter the resulting blob list. This technique, also known as whole image analysis, treats all foreground pixels as belonging to a single blob.
1 (`AM_SEGMENT_LABEL`)	Segment the image, label the foreground pixels, analyze and filter the resulting blob list.
2 (`AM_FILTER_ORIGINAL`)	Refilter the blob list obtained from the most recent

	labeling operation. In other words, discard all previous filters results. Do not segment, label or analyze.
3 (`AM_FILTER_CURRENT`)	Filter the current blob list keeping the results of any previous filters. Effectively ANDs the result of the current filter list with that of the previsous filter list.
4 (`AM_MASK_LABEL`)	Use the existing image mask to segment the image, label foreground pixels, analyze and filter the resulting blob list.
5 (`AM_MASK_NO_LABEL`)	Use the existing image mask to segment the image, do not label (treat the entire foreground as a single blob), then analyze and filter the resulting blob list.

If the property is read, as opposed to written, then the operation last performed will be repeated and the active *Apply* mode will be returned. Note that this behavior differs from previous versions of ***XCaliper*** where the number of blobs found was returned. The default mode is `AM_SEGMENT_LABEL`. It is used if the property is read without ever having been written.

Data Type

Integer

ApplyOnChange Property

Applies To

LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator, BlobAnalyzer, LightMeter tool

Description

Determines whether the tool automatically applies itself when the tools Viewport is resized or moved or the underlying frame buffer is changed.

Syntax

[*form*.]*Control*.**ApplyOnChange** [= {True | False}]

Remarks

When *ApplyOnChange* is set to True, any modification in the position or size of the Viewport, during design time or run time, will cause the tool to Apply itself to the image. Therefore if processing speed is of primary importance, or when an apply requires significant amount of processing time, the programmer may wish to set this to False. As such, the tool will only apply itself when explicitly asked to through the *Apply* property.

The default mode for this property depends on the type of tool as follows:

Tool	**Default Value**
ArcCaliper	True
LineCaliper	True
ArcEdgeLocator	True
LineEdgeLocator	True
LightMeter	True
BlobAnalyzer	False

Data Type

Integer (Boolean)

Area Property

Applies To

LightMeter tool

Description

Returns the area, in pixels, of the LightMeter's Viewport.

Syntax

[*form*.]*lightmeter*.**Area**

Remarks

The *Area* property is not available at design time and is read-only at run time.   The property returns the area of the Viewport in pixels - in effect, Width * Height.

Data Type

Integer

AspectRatio Property

Applies To

ArcCaliper, ArcEdgeLocator

Description

Determines the ratio of the horizontal diameter to the Vertical diameter of the curve.

Syntax

[*form*.]*Control*.**AspectRatio** [= value!]

Remarks

XCaliper Arc edge tools are restricted to those elliptical arcs which have an orthogonal orientation with respect to the major axes. As such, the shape of the ellipse can be defined simply as a ratio between the diameter in the x-direction and the diameter in the y-direction. For an *AspectRatio* of 1.0, a circle is drawn.

This property specifies the ratio of the diameters of the closed curve ellipse, not of the arc subtended by the start and end point properties.

By default, the *AspectRatio* is set to 1 (a circle).

Data Type

Single

AutoSize Property

Applies To

MemoryBuffer

Description

Determines whether the buffer is automatically resized to fit its contents.

Syntax

[*form*.]*Membuf*.**AutoSize** [= {True **|** False}]

Remarks

When *AutoSize* is set to True, the buffer size is changed whenever a new image is copied into it. If *AutoSize* is False, then the image will be clipped to fit before the copy.

Data Type

Integer (Boolean)

BlobType Property

Applies To

BlobAnalyzer

Description

Describes the kind of objects which are expected to be found in an image. Thi is used in combination with the UpperThreshold and LowerThreshold properties as a simple method for writing common binary and edge-oriented weighting schemes into the *WeightTable* to segment an image.

Syntax

[*form*.]*blobtool*.**Blobtype** [= type%]

Remarks

The following are the possible values for the *BlobType* property:

Setting	Description
0 (`BT_DARK`)	The foreground is darker than the background. Pixels below or equal to the LowerThreshold are foreground pixels; those above or equal to the UpperThreshold are full background. A linear ramp is used for values in between.
1 (`BT_LIGHT`)	The foreground is lighter than the background. Pixels below or equal to the LowerThreshold are foreground pixels; those above or equal to the UpperThreshold are full background. A linear ramp is used for values in between.
2 (`BT_USERDEFINED`)	The threshold are ignored and the definitions are taken directly from the WeightTable.
3 (`BT_BINARY_DARK`)	Like Dark except that the UpperThreshold and LowerThreshold are locked together such that the UpperThreshold is always kept equal to the LowerThreshold plus one. A modification to one threshold will cause the other to be changed to keep this relationship true. Since the size of the ramp is thus 0, this is equivalent to binary analysis.
4 (`BT_BINARY_LIGHT`)	Like BinaryDark except that the foreground is lighter than the background.
5 (`BT_MASK_ONLY`)	All pixels, no matter what their intensity, are to be considered full foreground. This mode is only useful when using a mask instead of segmentation. Otherwise, the entire image would be considered as a single blob.

Note that in all of these modes except `BT_USERDEFINED`, the *LowerThreshold* must always be kept at least one less than the *UpperThreshold*. In effect, the *LowerThreshold* value gives the highest intensity which is to be considered fully black while the *UpperThreshold* gives the lowest intensity which is fully white.

Directly changing any entry in the *WeightTable* will cause the *BlobType* property to be set to `BT_USERDEFINED`.

Data Type

Integer (Enumerated)

BufferSizeX, BufferSizeY Properties

Applies To

ImageDevice**,** MemoryBuffer

Description

BufferSizeX and *BufferSizeY* specify the width and height of the frame buffer in pixels.

Syntax

[*form*.]*control*.**BufferSizeX** [= width%]
[*form*.]*control*.**BufferSizeY** [= height%]

Remarks

For most image devices, the *BufferSizeX* and *BufferSizeY* properties are set by the driver and are read-only from the application. This restriction does not apply to **MemoryBuffers** although changing the height or width of this kind of buffer will destroy its contents.

Data Type

Integer

CenterX and CenterY Properties

Applies To

ArcCaliper, ArcEdgeLocator

Description

The *CenterX* and *CenterY* properties specify the x and y-coordinates, respectively, of the center point of the arc.

Syntax

[*form*.]*Control*.**CenterX** [= point!]

[*form*.]*Control*.**CenterY** [= point!]

Remarks

These properties define the center point of the ellipse from which the arc is derived. This point is not used to define the shape of the arc it is provided as a convenient positioning method.

Data Type

Single

ClosedCurve Property

Applies To

ArcCaliper, ArcEdgeLocator

Description

Determines whether to treat the arc as a closed curve (a complete ellipse) or an open curve (the arc between the start and end points).

Syntax

[*form*.]*Control*.**ClosedCurve** [= {True **|** False}]

Remarks

By default, the *ClosedCurve* property is set to False (open arc).

Data Type

Integer (Boolean)

Command Property

Commands

Applies To

ImageDevice, ImageHook, MemoryBuffer, BlobAnalyzer, LightMeter tool

Description

Specifies an action to be performed using the control.

Syntax

[*form*.]*control*.**Command** [= command$]

Remarks

This property is used to send command strings to the control which in turn interprets the strings and acts upon them. After execution, the ElapsedTime property is modified to indicate the amount of time taken to perform the action. If the *Command* property is read, it returns the last command executed.

When a command is executed on an **ImageDevice** or **MemoryBuffer**, it takes effect over the entire area of the frame buffer, unless the command has a source for the operation. In this case, the source supplies a size.

When executed on an **ImageHook**, **BlobAnalyzer**, or **LightMeter**, the command takes effect on the area of the buffer covered by the tool. In the case of an **ImageHook**, this means the visible area of the buffer. It is possible therefore, to set up a command over an arbitrary rectangular area of a buffer by using a **BlobAnalyzer** or **LightMeter** for this purpose. While either could be used, a **LightMeter** is the recommended tool in this circumstance because it requires fewer system resources.

Data Type

String

ContGrabType Property

Applies To

ImageDevice

Description

Selects the way that the system will react to a request for continuous grabbing of images.

Syntax

[*form*.]*control*.**ContGrabType** [= type%]

Remarks

This property selects the way that the system will react when continuous grabbing is enabled when using non-transparent mode.

The following are the possible values for the *ContGrabType* property.

Setting	Description
0 (`GT_SECONDARY`)	Do not use the VGA. Simply enable continuous grabbing on the board. This only makes sense if there is a secondary monitor attached to the card.
1 (`GT_CLEAN`)	Perform continuous grabbing into VGA memory. Favor clean display of images over rapid display of images. This is the default.
2 (`GT_FAST`)	Perform continuous grabbing into VGA memory. Favor rapid display of images over clean display of images.

When this propery is set to 1 (`GT_CLEAN`) or 2 (`GT_FAST`), images will be copied from the digitizer board to VGA memory as quickly as possible. The exact speed of transfer will depend on the type of digitizer used, the number of hooks attached to the **ImageDevice** and the size of the containers.

For this property to take effect, either the value of the *Transparent* property must be set to `TM_NONTRANSPARENT` or there must be at least one hook attached to the **ImageDevice** whose *Transparent* property is set this way. If this is not the case, then an error code will be generated.

Notes:

- Running continuous grab emulation in design mode is significantly faster when the Properties window is closed. When this window is open, Visual Basic will re-read the values of all properties listed at every image data transfer.
- If you exit Visual Basic design mode with continuous grab emulation enabled, the system may appear to hang. This is because Visual Basic has popped up a Dialog Box behind the grabbing window and is waiting for a response. To see the Dialog Box, touch the <ALT> key and continue normally.

Data Type

Integer (Enumerated)

Count Property

Applies To

BlobAnalyzer

Description

Returns the number of blobs found that passed the most recently applied filtering criteria.

Syntax

[*form*.]*blobtool*.**Count**

Remarks

This property is read-only. After each invocation of the Apply property, *Count* is set to the number of blobs found that passed the filter criteria. The maximum number of blobs that can be counted in an image is approximately 8000.

Data Type

Integer

CurAutoInc Property

Applies To

BlobAnalyzer

Description

Determines if and how the **BlobAnalyzer** automatically increments through the blob list when the CurValue property is read. It also determines the order in which the QuickData property returns the table of result data.

Syntax

[*form*.]*control*.**CurAutoInc** [= AutoIncType%]

Remarks

The following are the possible values for the *CurAutoInc* property:

Setting	Description
0 (`AI_BY_NONE`)	Don't auto-increment. Repeated reads of the *CurValue* property will return the same value.
1 (`AI_BY_FEATURE`)	Auto-incrementing is done in "blob-major" order. That is, the tool increments CurFeature through all the features of the current blob before incrementing CurBlob and resetting *CurFeature*, repeating the process for the next blob.
2 (`AI_BY_BLOB`)	Auto-incrementing is done in "feature-major" order. That is, the tool increments *CurBlob* through all the passed blobs before incrementing *CurFeature* and resetting *CurBlob*, repeating the process for the next feature.

If you need to access multiple features and/or multiple blobs then using auto-increment mode is more efficient than setting the *CurFeature* and *CurBlob* properties yourself.

Warning: when Auto_increment is turned on, the value returned by the *CurValue*, *CurFeature*, and *CurBlob* properties may vary in an apparently random manner in design mode. This is because Visual Basic frequently reads the values of properties itself. If it decides to read the *CurValue* property, then the *CurFeature* and *CurBlob* properties will be updated in the above manner. This problem does not affect Run mode.

Data Type

Integer (Enumerated)

CurBlob Property

Applies To

BlobAnalyzer

Description

The "Current Blob". *CurBlob* determines which blob's feature values are to be reported through the CurValue property.

Syntax

[*form*.]*blobtool*.**CurBlob** [= BlobNum%]

Remarks

Along with the CurFeature property, this property determines exactly what the value of *CurValue* represents. *CurValue* reports the value of the feature indicated by *CurFeature*, for the blob specified by *CurBlob*.

Data Type

Integer

CurFeature Property

Applies To

BlobAnalyzer

Description

The "Current Feature". *CurFeature* determines which feature is to be reported through the CurValue property.

Syntax

[*form*.]*blobtool*.**CurFeature** [= FeatureNum%]

Remarks

Along with the CurBlob property, this property determines exactly what the value of *CurValue* represents. *CurValue* reports the value of the feature indicated by *CurFeature*, for the blob specified by *CurBlob*.

Data Type

Integer (Enumerated)

CurValue Property

Applies To

BlobAnalyzer

Description

Returns the value of the feature indicated by CurFeature for the blob specified by the value of CurBlob.

Syntax

[*form*.]*blobtool*.**CurValue**

Remarks

This property is read-only. The *CurValue* property allows the user to access the results of the most recent invocation of the Apply property. Through it, the user can obtain the value of any feature (as long as it was previously selected) for any blob in the blob list by setting the *CurBlob* and *CurFeature* properties appropriately.

Data Type

Single

DeviceFile Property

Applies To

ImageDevice

Description

Specifies the type of frame grabber being used. This is the name of a device driver file to be loaded and used.

Syntax

[*form*.]*control*.**DeviceFile** [= file$]

Remarks

XCaliper supports dynamic loading of device drivers. This makes it simple to create applications which can be moved easily from one frame grabber to another or to support multiple frame grabbers in a single application.

XCaliper will attempt to load the driver using the exact name specified. If this fails, it will use the path variable to attempt to find the file and if that fails an error condition will be generated.

The *DeviceFile* property defaults to the value found in the LibName entry in your XCALIPER.INI file. This provides an easy method of specifying the type of frame grabber found in your system. It should not, therefore, be necessary to change this property in your application unless you need to be able to select frame grabbers dynamically.

Data Type

String

DeviceName Property

Applies To

ImageHook

Description

Specifies the name of the ImageDevice or MemoryBuffer to be hooked.

Syntax

[*form*.]*ImageHook*.**DeviceName** [= name$]

Remarks

This property allows the buffer which the **ImageHook** is displaying to be changed. For example, a **MemoryBuffer** can be displayed instead of the active **ImageDevice**.

This property should be set to the *Tag* property of the **MemoryBuffer** or **ImageDevice** which is to be displayed through the hook. Note that the *Tag* property is used, not the *Name* property. While ***XCaliper*** copies the *Name* property into the *Tag* property when a control is first created, later programmer action may cause the two properties to have different values.

The name "StdImageDev" is reserved. If *DeviceName* is set to StdImageDev, then the active image device will be hooked. It is also legal to hook to the active **ImageDevice** by setting the *DeviceName* to the *Tag* property of an appropriate **ImageDevice** control. Using the *Tag* property is the only way to set the **ImageHook** to display a **MemoryBuffer**.

As is true for Visual Basic in general, it is the programmers responsibility to ensure that the *Tag* property is given a different value in all controls. Note that the *Name* property is not guaranteed to be unique (controls in an arrary or in different forms can have the same *Name*) so relying on the default *Tag* value is not necessarily sufficient. ***XCaliper*** will work correctly if the *Tag* is unique across all frame stores but the general rule should be observed where possible.

The default value of this property is "StdImageDev".

Data Type

String

DisplayMode Property

Applies To

BlobAnalyzer

Description

Specifies how blobs are drawn on the display.

Syntax

[*form*.]*blobtool*.**DisplayMode** [= displayMode%]

Remarks

The settting of this property applies to all blobs. The following are the possible values for the *DisplayMode* property:

Setting	Description
0 (`DM_NONE`)	Blobs are not displayed.
1 (`DM_FILLED`)	Blobs are filled in according to the color selected using the AcceptColor, AcceptDisplay, RejectColor, and RejectDisplay properties. Any pixel in the blob having a non-zero weight will be set to this color.
2 (`DM_OUTLINED`)	Blobs are outlined with the color chosen according to the rule described above.
3 (`DM_TRANSLUCENT`)	Blobs are drawn in a see-through fashion. Every fourth pixel in the interior is colored, thus permitting display of both the blob found and the image underneath.

Data Type

Integer (Enumerated)

DrawColor Property

Applies To

LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator, BlobAnalyzer, LightMeter tool

Description

Specifies the color of the Viewport bounding box.

Syntax

[*form*.]*Control*.**DrawColor** [= &Hcolor&]

Remarks

This property is used to set the color of the rectangle drawn around the Viewport placed on the image. Setting the property is identical to setting other standard Visual Basic color properties. Refer to the Visual Basic Programmers Guide for more information on color properties.

Data Type

Long

Edge1Angle, Edge2Angle Properties

Applies To

ArcCaliper, ArcEdgeLocator

Description

Edge1Angle returns the angle of the first edge in the edge pair indicated by the ResultIndex property. This will be the edge closest to the start point along the line or curve.

Edge2Angle returns the angle of the second edge in the edge pair specified by the *ResultIndex* property. This will be the edge furthest from the start point along the line or curve.

Syntax

[*form*.]*Control*.**Edge1Angle**

[*form*.]*Control*.**Edge2Angle**

Remarks

The values of the properties are not available at design time and are read-only at run time. The results are the angles of the vectors, from the center of the closed curve to the points on the arc where the edges were detected, with respect to the positive x-axis. The values are reported in Engineering degrees (0 is at 3 oclock and values increase in a counter-clockwise direction).

Data Type

Single

Edge1Position, Edge2Position Properties

Applies To

LineCaliper, ArcCaliper

Description

Edge1Position returns the position of the first edge in the edge pair indicated by the ResultIndex property. This will be the edge closest to the start point along the line or curve.

Edge2Position returns the position of the second edge in the edge pair indicated by the *ResultIndex* property. This will be the edge furthest from the start point along the line or curve.

Syntax

[*form*.]*Control*.**Edge1Position**
[*form*.]*Control*.**Edge2Position**

Remarks

The values of the properties are not available at design time and are read-only at run time. If the Mode property for a **LineCaliper** is set to `MD_ANYANGLE`, the results are reported relative to the start point and scaled according to the ScaleX, ScaleY and ScaleType properties. The RefType property is then ignored. If the *Mode* property for a **LineCaliper** is set to `MD_HORIZONTAL` or `MD_VERTICAL`, the results are reported relative to the origin specified by the *RefType* property and scaled according to the *ScaleX*, *ScaleY* and *ScaleType* properties.

For **ArcCalipers**, the results are always reported in pixels along the curve relative to the start point because arc tools do not have scaling properties.

Data Type

Single

Edge1X, Edge2X, Edge1Y, Edge2Y Properties

Applies To

LineCaliper, ArcCaliper

Description

Edge1X and *Edge1Y* specify the coordinates of the first edge of the edge pair indicated by the ResultIndex property.

Edge2X and *Edge2Y* specify the coordinates of the second edge of the edge pair indicated by the *ResultIndex* property.

Syntax

[*form*.]*Control*.**Edge1X**

[*form*.]*Control*.**Edge1Y**

[*form*.]*Control*.**Edge2X**

[*form*.]*Control*.**Edge2Y**

Remarks

The values of the properties are not available at design time and are read-only at run time.   The results are reported relative to the containers origin and scaled according to the containers scale.

Data Type

Single

EdgeDisplay Property

Applies To

LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator

Description

Determines the way in which edges or edge pairsare displayed in the Viewport when they are found.

Syntax

[*form*.]*control*.**EdgeDisplay** [= mode%]

Remarks

The following table lists the possible values for the EdgeDisplay mode.

Setting	Description
0 (`ED_BEST`)	Only the edge or edge pair which best matches the constraint criteria is displayed.
1 (`ED_PASSFAIL`)	The edge or edge pair which best matches the constraint criteria is displayed in green if it passes and in red if it fails (not currently implemented equivalent to Best).
2 (`ED_ALL`)	All edges or edge pairs found are displayed.
3 (`ED_NONE`)	No edges or edge pairs are displayed under any circumstances. Note, however, that the Viewport will continue to be displayed.

Data Type

Integer (Enumerated)

ElapsedTime Property

Applies To

ImageDevice, MemoryBuffer, LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator, BlobAnalyzer, LightMeter tool

Description

Gives the amount of time, in milliseconds, taken to execute the last operation on the control. The timer is accurate to the nearest millisecond.

Syntax

[*form*.]*control*.**ElapsedTime**

Remarks

This property is not available at design time, and is read-only at run time. The only action measured for **ImageDevices** and **MemoryBuffers** is the amount of time taken to execute the operation specified in the Command property. The only action measured for **Calipers** and **EdgeLocators** is the amount of time required to execute the Apply property. For a **BlobAnalyzer** or a **LightMeter**, this property is updated after the execution of the *Apply* and the *Command* properties.

Data Type

Integer

EndPointX, EndPointY, StartPointX, StartPointY Properties

Applies To

LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator

Description

The *EndPointX* and *EndPointY* properties specify the coordinates of the end point of the line or arc. The *StartPointX* and *StartPointY* properties specify the coordinates of the start point of the line or arc.

Syntax

[*form*.]*control*.**EndPointX** [= value!]

[*form*.]*control*.**EndPointY** [= value!]

[*form*.]*control*.**StartPointX** [= value!]

[*form*.]*control*.**StartPointY** [= value!]

Remarks

The values of these properties are specified relative to the frame buffer origin and are scaled according to the ScaleX, ScaleY and ScaleType properties.

Data Type

Single

Features Property

Applies To

BlobAnalyzer

Description

This property determines which features are to be calculated and whether or not the current filter for each feature should be applied.

Syntax

[*form*.]*control*.**Features**(*Index*) [= value%]

Remarks

The *Features* property is an array where each feature has been assigned a number that is used as an index into the array when setting or reading that feature's value. By including the file XCAIPER.TXT, which contains various Visual Basic declarations, the programmer is able to use the much more intuitive constant name for the feature as the index, avoiding the need to memorize the actual numerical values. This property is not directly available at design time but it can be accessed and modified through the Setup property's dialog box. The following table lists the possible values for the propery:

Setting	Description
0 (`FE_DONT_EXTRACT`)	Do not compute the indexed feature on subsequent applies.
1 (`FE_EXTRACT`)	On subsequent applications of the tool, compute the indexed feature for each blob found, but ignore any filter setting for this feature.
2 (`FE_EXTRACT_AND_FILTER`)	On subsequent applications of the tool, compute this feature for each blob found and use the current filter setting for this feature to filter out blobs accordingly.

The set of features returned through the QuickData and CurValue properties are also controlled by the setting of the *Features* property. Features which have this property set to either `FE_EXTRACT` or `FE_EXTRACT_AND_FILTER` will be returned in the *QuickData* property and are accessible through the *CurValue* property. Features which have this property set to `FE_DONT_EXTRACT` are not available through the *CurValue* property and will not be returned through the *QuickData* property. An attempt to set the CurFeature property to such a feature will generate a Visual Basic error.

This behavior may cause problems when a feature is turned on(e.g. when the setting of this property is changed from `FE_DONT_EXTRACT` to another value). It would thus be possible to extract features which were never analyzed. ***XCaliper*** resolves this problem by assuming that if a feature is turned on, it must be because an Apply is about to happen. Therefore it discards the entire previous analysis. Any attempt to access any feature will fail until the next Apply occurs. The *QuickData* property will return an empty string and the *CurValue* property will return 0. Nothing in this prevents the designer from turning off features after an Apply has been invoked, and turning them back on later. Enabling a new feature causes the analysis to be discarded rather than re-enabling an old one.

Data Type

Array of Integer (Enumerated)

Filters Property

Applies To

BlobAnalyzer

Description

An array of strings specifying the filter bounds to use for each feature when filtering the blob list.

Syntax

[*form*.]*control*.**Filters**(*Index*) [= string$]

Remarks

The *Filters* property is an array where each feature has been assigned a number that is used as an index into the array when setting or reading that feature's filter string. By including the file XCALIPER.TXT, which contains various Visual Basic declarations, the programmer is able to use the much more intutive constant name for the filter as the index, avoiding the need to memorize the actual numerical values. This property is not directly available at design time but it can be accessed and modified through the Setup property's dialog box.

The syntax for the filter strings is as follows:

"[NOT] { (| [} [*min*], [*max*] {) |] }"

min and *max* represent the minimum and maximum filter limits respectively and are both optional. If not included, the limit is ignored. The filter limits pair may be enclosed in square brackets, rounded brackets, or a combination of the two. A square bracket indicates the interval includes the limit, while a round bracket indicates that the interval does not include the limit. The optional NOT reverses the filtering logic - acceptable values are those outside the interval. Some typical strings and what they mean are listed below:

String	Description
"[20, 30)"	Values in the range, $20 \pounds x < 30$ are considered acceptable, all others are rejected.
"[24,]	values greater than or equal to 24 are considered acceptable, while all values less than 24 are rejected.
NOT (13, 34]	Values less than or equal to 13 and values greater than 34 are considered acceptable.

Data Type

Array of String

GrabMode Property

Applies To

ImageDevice

Description

Determines the type of frame grab being used.

Syntax

[*form*.]*ImageDevice*.**GrabMode** [= mode%]

Remarks

The following table lists the possible values for the *GrabMode* property of the **ImageDevice** control.

Setting	**Description**
0 (`GT_GRAB_OFF`)	Frame grabbing is turned off. Also known as freeze frame mode.
1 (`GT_GRAB_ON`)	Frame grabbing is turned on and left on until reset. This mode is also known as continuous grab mode.
2 (`GT_SINGLE_FRAME`)	A single frame is acquired from the camera and placed in the frame buffer. After the acquisition is completed, the *GrabMode* property will revert to 0 (Off). This provides a simple method of determining that the acquisition is finished.
3 (`GT_SINGLE_FIELD`)	A single field is acquired from the camera and placed in the frame buffer. After the acquisition is completed, the *GrabMode* property will revert to 0 (Off). This provides a simple method of determining that the acquisition is finished.

Some device drivers do not support SingleField grabbing. If so, a single frame will be acquired instead. Continuous grab mode (setting 1) will use the field or frame mode selected by the previous setting of the *GrabMode* property, namely either SingleFrame or SingleField. For example, to ensure that continuous grab takes place in field mode, set the *GrabMode* property to `GT_SINGLE_FIELD`, then set it to `GT_GRAB_ON`.

Data Type

Integer (Enumerated)

Histogram Property

Applies To

LightMeter tool

Description

Returns the frequency of occurrence of the specified intensity value.

Syntax

[*form*.]*lightmeter*.**Histogram**(*Index*)

Remarks

This property is not available at design time and is read-only at run time. By reading this property, the programmer is able to access the raw histogram data. The value of *Index* specifies the intensity value to return and must be between 0 and 255.

Data Type

Array of Long Integer

HoleFill Property

Applies To

BlobAnalyzer

Description

Sets or returns a Boolean value that determines whether or not automatic hole filling should occur when the image is segmented.

Syntax

[*form*.]*blobtool*.**HoleFill** [= {True | False}]

Remarks

Holes are regions within blobs whose pixel values have a weight of less than 1.0.   In order to be considered a hole, the region must be entirely surrounded by pixels with weightings of 1.0.   If HoleFill is set to True, then all regions are automatically filled in   that is, their weights are automatically set to 1.0 during image segmentation.

Data Type

Integer (Boolean)

Inclination Property

Applies To

LineCaliper, LineEdgeLocator

Description

Sets the angle at which the **LineCaliper** or **LineEdgeLocator** is executed.

Syntax

[*form*.]*control*.**Inclination** [= angle!]

Remarks

This property gives the angle of the vector from the start point of the edge tool to its end point with respect to the positive x-axis. This value is supplied and reported in Engineering degrees (0 is at 3 o'clock and positive numbers increase in a counter-clockwise direction).

Setting this property to some other value than 0º or 270º while the Mode property is set to `MD_HORIZONTAL` or `MD_VERTICAL` will change the mode to `MD_ANYANGLE`.

When the Mode property is set to 0 (`MD_HORIZONTAL)`, the *Inclination* is automatically changed to 0º. When the *Mode* property is set to 1 (`MD_VERTICAL)`, the *Inclination* is automatically changed to 270º (***XCaliper*** defines vertical as downwards).

Data Type

Single

InputChannel Property

Applies To

ImageDevice

Description

Determines the channel to be used for frame acquisition.

Syntax

[*form*.]*ImageDevice*.**InputChannel** [= channel%]

Remarks

Channel numbers follow the numbering scheme used by the frame grabber.

Data Type

Integer

InputGain Property

Applies To

ImageDevice

Description

Specifies the level of the input gain to the frame grabber card.

Syntax

[*form*.]*ImageDevice*.**InputGain** [= gain%]

Remarks

The gain is represented by a number between 0 and 255. The smaller the number, the smaller the slope of the gain function. The exact minimum and maximum values as well as the shape of the gain function depend on the digitizer being used. Furthermore, some digitizers may define this property as a white-level cut-off instead of as a gain. See the frame grabber documentation for further information.

Data Type

Integer

InputLut Property

Applies To

ImageDevice

Description

This property is an array of 256 integers containing a copy of the input look up table (LUT).

Syntax

[*form*.]*control*.**InputLut** (index%) [= intensity%]

Remarks

This property is not available at design time. The *InputLut* propety is an array of 256 integers which is used to read and write the input LUT. Each element is organized as a 16-bit number where the least significant eight bits represent the gray level written to a frame buffer; the high byte is ignored.

Data Type

Array of Integers

InputOffset Property

Applies To

ImageDevice

Description

Specifies the level of the input offset to the frame grabber card.

Syntax

[*form*.]*ImageDevice*.**InputOffset** [= offset%]

Remarks

The offset is a number between 0 and 255 which specifies the reference input voltage to be digitized as pure black. Lower numbers indicate a lower black reference level and therefore a brighter image overall. The exact voltage given by any particular number depends on the frame grabber being used.

Data Type

Integer

InputOriginX, InputOriginY Properties

Applies To

ImageDevice

Description

InputOriginX and *InputOriginY* determine the X and Y-positions, respectively, in the input video signal at which grabbing starts.

Syntax

[*form*.]*ImageDevice*.**InputOriginX** [= position%]
[*form*.]*ImageDevice*.**InputOriginY** [= position%]

Remarks

Together, these properties determine the location in the camera signal at which digitization is started. For example, if the video signal is 640 X 480 and we are acquiring 512 X 480 images, then if the x-origin is set to 64, the signal will be blanked for 64 pixels; the image will be acquired in the next 512 pixels and the last 64 pixels will also be blanked. Setting the X-origin to 0 would left-justify the acquisition while setting it to 128 would right-justify it.

Note that for standard RS-170 cameras, the Y-origin should be left at its default value of zero. However, certain windowing acquisition systems may support other values.

Different frame grabbers may place various hardware restrictions on the value of these properties such as locking it to the OutputOriginX, or the OutputOrginY. Due to this, the value read may not be the same as the value written.

Data Type

Integer

InputPan Property

Applies To

ImageDevice

Description

Determines the x-position in the frame buffer at which grabbing starts.

Syntax

[*form*.]*ImageDevice*.**InputPan** [= position%]

Remarks

Together with the InputScroll property, this property determines the location in the frame buffer at which pixels are stored.

Different frame grabbers may place various hardware restrictions on the value of this property such as locking it to the OutputPan property and rounding it to some power of two.   As a result, the value read may not be the same as the value written.

Data Type

Integer

InputScroll Property

Applies To

ImageDevice

Description

Determines the y-position in the frame buffer at which grabbing starts.

Usage

[*form*.]*ImageDevice*.**InputScroll** [= position%]

Remarks

Together with the InputPan property, this property determines the location in the frame buffer at which pixels are stored.

Different frame grabbers may place various hardware restrictions on the value of this property such as locking it to the OutputScroll property and rounding it to some power of two. Due to this, the value read may not be the same as the value written.

Data Type

Integer

InputSizeX, InputSixeY Properties

Applies To

ImageDevice

Description

InputSizeX and *InputSizeY* determine the X and Y-size resolutions, respectively, of the grabbed image.

Syntax

[*form*.]*ImageDevice*.**InputSizeX** [= size%]
[*form*.]*ImageDevice*.**InputSizeY** [= size%]

Remarks

InputSizeX determines the number of pixels digitized from each line in the incoming video signal. The *InputSizeY* determines the number of lines digitized from the incoming video signal.

These fields are set at initialization to reasonable defaults (such as 480 and 640) which depend on the frame grabber being used. Different frame grabbers may place various hardware restrictions on the values of these properties such as locking them to other input and output properties and restricting their possible range of values. As a result, the values read may not be the same as the values written.

It is an error to set these properties to values which would cause more than 100% of the incoming signal to be digitized. The exact value which is illegal for the *InputSizeX* depends on the current value of the InputOriginX property. The sum of the values of the *InputOriginX* and *InputSizeX* cannot exceed the number of pixels in a video line (normally 512 or 640). The restrictions on the *InputSizeY* work identical to those of *InputSizeX*.

Data Type

Integer

InputZoomX, InputZoomY Properties

Applies To

ImageDevice

Description

InputZoomX and *InputZoomY* determine the zoom factors to be applied at digitization in the X and Y-directions, respectively.

Syntax

[*form*.]*ImageDevice*.**InputZoomX** [= value!]
[*form*.]*ImageDevice*.**InputZoomY** [= value!]

Remarks

These properties determine the magnification factors applied when an image is digitized.

Different frame grabbers may place various hardware restrictions on the values of these properties such as restricting them to integral values or powers of two. As a result, the values read may not be the same as the values written.

Data Type

Single

LeftTailSize Property

Applies To

LightMeter tool

Description

Specifies the percentage area of the histogram to be considered part of the Left tail.

Syntax

[*form*.]*lightmeter*.**LeftTailSize**(*Index*)

Remarks

This property specifies what proportion of the total number of pixels, at the low end of the histogram, should be considered outliers and part of the Left tail. The value entered is treated as a percentage and can be any valid floating point number between 0 and 100.

Data Type

Single

LockAngle Property

Applies To

LineCaliper, LineEdgeLocator

Description

Locks the inclination of the tool to the value set in the Inclination property

Syntax

[*form*.]*control*.**LockAngle** [= {True **|** False}]

Remarks

When the property is set to True, any interactive changes of the Viewport will maintain the inclination set in the Inclination property. However, any changes to the Viewport via Visual Basic code or the properties window is permitted.

Data Type

Integer (Boolean)

LowerLimit and UpperLimit Properties

Applies To

LightMeter tool

Description

The *LowerLimit* and *UpperLimit* properties specify the minimum and maximum pixel values, respectively, used by the **LightMeter** when generating the various statistical quantities computed from the histogram of pixel values contained within the **LightMeter**s Viewport.

Syntax

[*form*.]*control*.**LowerLimit** [= value%]

[*form*.]*control*.**UpperLimit** [= value%]

Remarks

These properties can be used to specify to the LightMeter a range of pixel values to use when computing the various statistical quantities.   For example, if you wanted to know the mean of the pixel values between 145 and 225 you would set the *LowerLimit* and *UpperLimit* accordingly and then access the mean (or any other quantity you might be interested in) through the *Stats()* property.

Data Type

Integer (Range 0-254)

LowerThreshold, UpperThreshold Properties

Applies To

BlobAnalyzer

Description

LowerThreshold and *UpperThreshold* specify the lower and the upper thresholds respectively. These properties are used to determine the *WeightTable* constructed when BlobType is set to a value other than `BT_USERDEFINED`.

Syntax

[*form*.]*control*.**LowerThreshold =** [= value%]

[*form*.]*control*.**UpperThreshold =** [= value%]

Remarks

The *LowerThreshold* must never be set greater than or equal to the *UpperThreshold*. In either of the binary *BlobType* modes, any attempt to change the *LowerThreshold* will cause the *UpperThreshold* to change as well. In any non-binary *BlobType* mode, any attempt to set the *LowerThreshold* greater than or equal to the *UpperThreshold* will generate an error.

Data Type

Integer (Range 0-254)

Mask Property

Applies To

BlobAnalyzer

Description

Contains the mask used to determine which pixels in the frame buffer are to be considered foreground and which are background.

Syntax

[*form*.]*control*.**Mask** [= picture%]

Remarks

This is a standard Visual Basic picture in .bmp format. It may be read and written using picture-related mechanisms like *LoadPicture* and *SavePicture*.

The bitmap is binary. Pixels with index 0 are background and others are foreground. Currently, there is no method to supply a labelled mask nor to read the labelled blobs. If using PaintBrush or a similar program, pixels drawn in white will be background and those drawn in black will be foreground. It is not possible to save the pixel weighting information to the mask as the mask is binary only.

This property exists primarily to allow the programmer to apply the same mask to multiple images as in the following example:

```
Blobtool1.Mask = Blobtool2.Mask

Blobtool1.Apply = AM_MASK_LABEL
```

If HoleFill is enabled when the mask is written, holes will be filled in at this point.

When analysis is performed using a mask, it defines the area of the frame buffer where pixels are to be analyzed. Inside this area the WeightTable is still applied to features extracted. Thus if the intention is that only the mask should be taken into account, the BlobType should be set to `BT_MASK_ONLY`.

Data Type

Integer

MaxResults Property

Applies To

LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator

Description

Specifies the maximum number of edges or edge pairs which will be returned from an edge tool.

Syntax

[*form*.]*control*.**MaxResults** [= value%]

Remarks

The default value is 25 edges or edge pairs. It is guaranteed that the best matching results will be returned where the meaning of best depends on the setting of the tools SortOrder property.

Data Type

Integer

MidPointX, MidPointY Properties

Applies To

ArcCaliper, ArcEdgeLocator

Description

MidPointX and *MidPointY* specify the x and y-coordinates, respectively, of the middle pick point.

Syntax

[*form*.]*control*.**MidPointX** [= value!]

[*form*.]*control*.**MidPointY** [= value!]

Remarks

The values of these properties are specified relative to the frame buffer origin and are scaled according to the ScaleX, ScaleY and ScaleType properties.

Data Type

Single

Mode Property

Applies To

LineCaliper, LineEdgeLocator

Description

Specifies the mode of operation of the Line Edge tools.

Syntax

[*form*.]*control*.**Mode =** [= value%]

Remarks

The following table lists the possible values for the *Mode* property.

Setting	Description
0 (`MD_HORIZONTAL`)	The Inclination property is fixed at 0. This mode is optimized for speed. Only vertical edges will be found (perpendicular to the Viewport).
1 (`MD_VERTICAL`)	The *Inclination* property is fixed at 270. This mode is optimized for speed. Only horizontal edges will be found (perpendicular to the Viewport).
2 (`MD_ANYANGLE`)	The tool may be rotated to assume any inclination. Edges will be found at the angle which is perpendicular to the angle specified by the value of the *Inclination* property minus that of the Skew property.

When the mode is changed, the tool attempts to keep the same shape, at the expense of the same position if one of the two must change. For example, if the *Mode* is modified from `MD_HORIZONTAL` to `MD_VERTICAL` , then the *Height* and *Width* properties will change. The PathLength and Thickness properties will be unchanged.

Data Type

Integer (Enumerated)

NumResults Property

Applies To

LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator

Description

This property returns the number of single edges for the **ArcEdgeLocator** and **LineEdgeLocator** tools, or edge pairs for **ArcCaliper** and **LineCaliper** tools found when the edge tool was last applied.

Syntax

[*form*.]*control*.**NumResults**

Remarks

The value of the property is not available at design time and is read-only at run time. The value of hte property will be zero if the edge tool has never been applied.

Data Type

Integer

OutputLut Property

Applies To

ImageDevice, MemoryBuffer

Description

This property is an array of 256 Integers containing a copy of the output look up table (LUT). This property is not available at design time.

Syntax

[*form*.]*control*.**OutputLut** (index%) [= position%]

Remarks

This is an array of 256 Long Integers which are used to read and write the output LUT. Each element is organized as a 24-bit number representing one color with Red in the least significant byte, Green in the next byte and Blue in the most significant byte. This is the same order used by Visual Basic and Windows to represent color values.

Note that the LUT is also used when the buffer is displayed through a non-transparent hook. In this case, though, the quality of the LUT mapping is limited by the Windows graphics display. A graphics card supporting at least 256 colors should therefore be used. Please see the README.TXT file for information about how this property works with certain frame grabbers.

Data Type

Array of Long

OutputOriginX, OutputOrginY Properties

Applies To

ImageDevice

Description

OutputOriginX and *OutputOriginY* determine the X-position and the Y-position, respectively, in the output video signal at which frame buffer display starts.

Syntax

[*form*.]*control*.**OutputOriginX** [= position%]

[form.]*control*.**OutputOriginY** [= position%]

Remarks

Together, these properties determine the location in the video signal at which the image is displayed. For example, if the video signal is 640 X 480 and we are displaying 512 X 480 images, then if the X origin is set to 64, the signal will be blanked for first 64 columns; the image will be displayed in the next 512 pixels and the last 64 pixels will also be blanked. Setting the x-origin to 0 would left-justify the acquisition while setting it to 128 would right-justify it.

Similarly, if the video signal is 640 X 480 and 640 X 400 pixel images are being displayed, then if the Y origin is set to 40, the signal will be blanked for the first 40 rows of pixels, the image will be displayed in the next 400 rows and the last 40 rows will also be blanked. Setting the Y origin to 0 would top-justify the display while setting it to 80 would bottom-justify it.

Different frame grabbers may place various hardware restrictions on the values of these properties such as locking them to the *InputOriginX* and *InputOriginY*. As a result, the values read may not be the same as the values written.

The values of these properties should not be changed if the *Transparent* property has been set to 2 (`TM_LOCKEDWINDOW`). LockedWindow transparency and setting the origin through the OutputOrigin properties serve the same purpose and therefore do not work together.

Data Type

Integer

OutputPan Property

Applies To

ImageDevice, ImageHook, MemoryBuffer

Description

Determines the X-position in the frame buffer at which display starts.

Syntax

[*form*.]*control*.**OutputPan**[= position%]

Remarks

Together with the *OutputScroll* property, this property determines the location in the frame buffer which corresponds to the *OutputOriginX*.

When applied to an unhooked or transparent **ImageDevice** or **ImageHook**, the value of this property specifies where the display starts on the imaging monitor in dual monitor mode or on the underlay in single monitor mode. The value may be restricted by a variety of hardware limitations. Some of these include: locking it to the *InputPan* property and rounding it to a power of two. When this happens, the value read may not be the same as the value written.

For non-transparent image tools, this property specifies where the display starts on the VGA screen and has no effect on the underlay if any.

Data Type

Integer

OutputScroll Property

Applies To

ImageDevice, ImageHook, MemoryBuffer

Description

Determines the Y-position in the frame buffer at which display starts.

Syntax

[form.]*control*.**OutputScroll**[= position%]

Remarks

Together with the *OutputPan* property, this property determines the location in the frame buffer which corresponds to the *OutputOriginY*.

When applied to an unhooked or transparent **ImageDevice** or **ImageHook**, the value of this property specifies where the display starts on the imaging monitor in dual monitor display or on the underlay in single monitor mode. The value may be restricted by a variety of hardware limitations. Some of these include: locking it to the *InputScroll* property and rounding it to some power of two. As a result, the value read may not be the same as the value written.

For non-transparent image tools, this property specifies where the display starts on the VGA screen and has no effect on the underlay if any.

Data Type

Integer

OutputSizeX OutputSizeY Properties

Applies To

ImageDevice

Description

OutputSizeX and *OutputSizeY* determine the resolution of the display in the X-direction and the Y-direction, respectively.

Syntax

[form.]*ImageDevice*.**OutputSizeX** [= position%]
[form.]*ImageDevice*.**OutputSizeY** [= position%]

Remarks

These properties control the number of pixels displayed on the screen. If the number of pixels displayed is less than the size of the display, the rest will be blanked.

Different frame grabbers may place various hardware restrictions on the values of these properties such as locking them to the *InputSizeX* and *InputSizeY* properties. As a result, the values read may not be the same as the values written.

Data Type

Integer

OutputZoomX, OutputZoomY Properties

Applies To

ImageDevice, ImageHook, MemoryBuffer

Description

OutputZoomX and *OutputZoomY* determine the zoom factors to be applied at display in the X-direction and Y-direction, respectively.

Syntax

[form.]*control*.**OutputZoomX** [= value!]
[form.]*control*.**OutputZoomY** [= value!]

Remarks

This property determines the magnification factor applied when an image is displayed.

Different frame grabbers may place various hardware restrictions on the values of these properties such as restricting them to integral values or powers of two. As a result, the values read may not be the same as the values written. However, no restrictions are imposed on their values for non-transparent image devices. Any positive zoom factor is legal, including values less than 1.

Note: Tools placed on the **ImageHook** or **ImageDevice** are not zoomed and therefore will be misplaced on the display. Their location in the frame buffer is still correct, however. It is suggested that tools sitting on a zoomed image be made invisible.

Data Type

Single

PathLength Property

Applies To

LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator

Description

Specifies the number of points on the line or arc.

Syntax

[*form*.]*control*.**PathLength**

Remarks

this property returns the length, in pixels, of the edge tools Viewport along its axis. It is a read-only property except for line tools in Horizontal or Vertical mode. For line tools in these specific modes the property becomes a convenient way to adjust the size of the Viewport.

Data Type

Single

PathSize Property

Applies To

ArcCaliper, ArcEdgeLocator

Description

Specifies the maximum number of points permitted on an arc.

Syntax

[*form*.]*control*.**PathSize** [= value%]

Remarks

The *PathSize* property is used to limit the amount of memory that is allocated, and the number of points to teach on an arc. Any attempts to increase the size of the arc beyond the limit set by the *PathSize* property will result in an error and the arc will return to the shape and size it had before the attempt. The *PathSize* property must be in the range 0-15000.

Data Type

Integer

Position Property

Applies To

LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator

Description

Specifies the position of an edge or edge pair. The position of an edge pair is defined as the midpoint of the position of the two edges in the pair.

Syntax

[*form*.]*control*.**Position**

Remarks

The value of the property is not available at design time and is read-only at run time. For **LineCaliper** and **LineEdgeLoctor** tools only, the result is scaled according to the ScaleX, ScaleY, and ScaleType properties. If the Mode property for a **LineCaliper** or a **LineEdgeLocator** is set to `MD_ANYANGLE`, the result is reported relative to the start point and the RefType is ignored. If the *Mode* property for a **LineCaliper** or **LineEdgeLocator** is set to `MD_HORIZONTAL` or `MD_VERTICAL`, the result is reported relative to the origin specified in the *RefType* property.

For **ArcCaliper** and **ArcEdgeLocator** tools, the result is always reported in pixels along the arc relative to the start point.

Data Type

Single

QuickData Property

Applies To

BlobAnalyzer, LightMeter tool

Description

Returns all the tools results in a single string.

Syntax

[*form*.]*control*.**QuickData**

Remarks

This property is read-only. It is sometimes desirable to be able to obtain and manipulate all the data points in a single operation - perhaps to pass the data to another tool such as a database, spreadsheet, or graphing control. This can be achieved by assigning the return value of *QuickData* to a programmer defined string, or similar property such as the **Graph** control's *QuickData* property.

The data is in the form of a string with each data point separated by a tab character (Chr$(9)), and each data set by a CR+LF (Chr$(13) + Chr$(10)). For example, a string consisting of two data sets and only a few data points may look like the following:

```
27<TAB>32<TAB>12<TAB>8<TAB>42<CRLF>
9<TAB>7<TAB>76<TAB>65<TAB>73
```

The first set consists of the first five numers while the remaining six numbers make up the second set. Note that the number of data points on each line is always the same.

BlobAnalyzer

All of the values of the features extracted are returned in a two-dimensional array. If the CurAutoInc property is set to `AI_BY_FEATURE` then the features will be placed along the horizontal axis and blobs along the vertical axis. If *CurAutoInc* is set to `AI_BY_BLOB` this order is reversed. If it is set to None, the result is undefined. If the QuickLabels property is set to True, then blob number and feature name labels will automatically be added along both axes. If the amount of information to be returned exceeds the maximum length of a string (slightly less than 64K bytes), then the result returned by the property is undefined. It is suggested that a loop over all values be used instead if this is a serious possibility.

LightMeter

Two sets of data are provided - the standard histogram and the cumulative histogram. Through the use of the QuickDataMode property, the programmer can request one or the other, or both. If both are requested, the first 256 values represent the standard data, followed by a CR+LF and then another 256 values representing the cumulative data.

Data Type

String

QuickDataMode Property

Applies To

LightMeter

Description

Determines the nature of the data returned in the QuickData property.

Syntax

[*form*.]*lightmeter*.**QuickDataMode** [= mode%]

Remarks

The following are the possible values for the *QuickDataMode* property:

Setting	**Description**
0 (LM_HISTOGRAMONLY)	The *QuickData* property will return only standard histogram data.
1 (LM_CUMULATIVEONLY)	The *QuickData* property will return only cumulative histogram data.
2 (LM_BOTH)	The *QuickData* property will return both standard and cumulative histogram data, in that order.

Data Type

Integer (Enumerated)

QuickLabels Property

Applies To

BlobAnalyzer

Description

Specifies whether labels are to be added to the string returned through the QuickData property.

Syntax

[*form*.]*control*.**QuickLabels** [= {True | False}]

Remarks

The *QuickLabels* property, when set to True, adds the blob number and feature name labels along both axes of the two dimensional string returned from the *QuickData* property. If the CurAutoInc property is set to `AI_BY_FEATURE` then the features will be placed along the horizontal axis and blobs along the vertical axis. If *CurAutoInc* is set to `AI_BY_BLOB` this order is reversed. If the *QuickLabels* property is set to False, then no labels are added, regardless of the setting of the *CurAutoInc* property.

Data Type

Integer (Boolean)

RefType Property

Applies To

LineCaliper, LineEdgeLocator, BlobAnalyzer

Description

Specifies how to calculate the reference point for reporting results.

Syntax

[*form*.]*control*.**RefType** [= value%]

Remarks

Results are reported with respect to a reference point. The *RefType* property applies to a **LineCaliper** or a **LineEdgeLocator** only if the Mode property is set to `MD_HORIZONTAL` or `MD_VERTICAL`. The following are the possible values for the *RefType* property:

Setting	Description
0 (`REF_ABSOLUTE`)	Values are reported with respect to the upper left hand corner of the frame store.
1 (`REF_RELATIVE`)	Values are reported with respect to the origin of the tool. Since an edge tool is a directed vector, this origin will be located at the start point of the edge tool. This is the default value of the edge tools. In the **BlobAnalyzer** the origin is always at the upper-left corner of the Viewport.

Data Type

Integer (Enumerated)

RejectColor Property

Applies To

BlobAnalyzer

Description

Specifies the display color for blobs that fail subsequent filtering operations.

Syntax

[*form*.]*control*.**RejectColor** [= &Hcolorvalue%]

Remarks

This property, along with the RejectDisplay property, determines how blobs that fail subsequent filtering operations will be displayed on the image display. If *RejectDisplay* is set appropriately, *RejectColor* sets the color used to display the rejected blobs in the image. Setting the property is identical to setting other standard Visual Basic color properties. Refer to the Visual Basic Programmer's Guide for more information on color properties and how to set them.

Data Type

Long Integer

RejectDisplay Property

Applies To

BlobAnalyzer

Description

Specifies how the **BlobAnalyzer** alters the overlay when displaying blobs that fail subsequent filtering operations (invoked through the Apply property).

Syntax

[*form*.]*control*.**RejectDisplay** [= displaytype%]

Remarks

The following are the possible values for the *RejectDisplay* property:

Setting	Description
0 (`NO_CHANGE`)	The color of the blobs remains unchanged from what it was before the application of the tool.
1 (`BD_CLEAR`)	The blobs' overlay display is cleared such that only the underlying original image is visible.
2 (`BD_COLOR`)	The blobs are set to the color specified in the *RejectColor* property.

Data Type

Integer (Enumerated)

ResultIndex Property

Applies To

LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator

Description

Specifies the edge or edge pair from which information is reported in other properties.

Syntax

[*form*.]*control*.**ResultIndex** [= value%]

Remarks

An edge tool stores information about all edges or edge pairs which pass the minimum accept criteria specified in the setup box. While only the best edge or edge pair may normally be of interest, it may be desired to look at others. This property is used to select the edge or edge pair for edge reporting properties, which include: Position, X, and Y, Edge1Position, Edge2Position, Edge1X , Edge1Y , Edge2X , Edge2Y , and Size.

Setting	**Description**
0	First edge/edge pair returned.
1	Second edge/edge pair returned.
...	...
N	The (N-1)th edge/edge pair returned.

Example

```
Dim NumResults As Integer
Dim Positions(20) As Single
Dim I As Integer
NumResults = Caliper1.Apply
If NumResults = 0 Then
     MsgBox "No results found!"
ElseIf NumResults >= 20 Then
     MsgBox "Too many results to process!"
Else
     For I = 0 To NumResults - 1
           Caliper1.ResultIndex = I
           Positions(I) = Caliper1.Position
     Next I
```

```
End If
....
```

Data Type

Integer

RightTailSize Property

Applies To

LightMeterVBX_LIGHTMETER

Description

Specifies the percentage area of the histogram to be considered part of the right tail.

Syntax

[*form*.]*lightmeter*.**RightTailSize** [= tailsize!]

Remarks

This property specifies what proportion of the total number of pixels, at the high end of the histogram, should be considered outliers and part of the right tail.   The value entered is treated as a percentage and can be any valid floating point number between 0 and 100.

Data Type

Single

ScaleType Property

Applies To

ImageDevice, MemoryBuffer, LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator, BlobAnalyzer

Description

Specifies the type of scaling to be used when reporting back the results of the tool acting on the frame store.

Syntax

[*form*.]*control*.**ScaleType** [= value%]

Remarks

Setting the ScaleX, or ScaleY property automatically forces the value of the *ScaleType* property to be set to 0 (`ST_USER`). Setting the value of the *ScaleType* property to 1 (`ST_PIXELS`) automatically forces the value of the *ScaleX* and *ScaleY* properties to 1. For **Edge Tools** or **BlobAnalyzers** only, setting the value of the *ScaleType* property to 2 (`ST_FROM_DEVICE`) causes the *ScaleX* and *ScaleY* properties to reflect the coordinate system of the frame store on which the tool acts; consequently, changing these properties in the frame strore automatically causes them to change in the tools whose *ScaleType* property is set to `ST_FROM_DEVICE` as well.

The following table lists the possible values for the *ScaleType* property.

Setting	Description
0 (`ST_USER`)	The translation performed is as specified in the *ScaleX* and *ScaleY* properties.
1 (`ST_PIXELS`)	No translation is performed.
2 (`ST_FROM_DEVICE`)	The translation performed is as specified in the *ScaleX* and *ScaleY* properties of the frame store on which the tool acts. Verical edge tools scale according to the framestore's *ScaleY* property; horizontal edge tools follow its *ScaleX*, any angle edge tools scale according to a combination of *ScaleX* and *ScaleY*.

MemoryBuffer and **ImageDevice** controls, naturally enough, do not have the `ST_FROM_DEVICE` setting.

Data Type

Integer (Enumerated)

ScaleX, ScaleY Properties

Applies To

ImageDevice, MemoryBuffer, LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator, BlobAnalyzer

Description

Specifies the scaling factor to be used in the x and y-direction to translate from pixels to user units.

Syntax

[*form*.]*control*.**ScaleX** [= value!]

[*form*.]*control*.**ScaleY** [= value!]

Remarks

Setting the *ScaleX* or *ScaleY* properties automatically forces the value of the ScaleType property to be set to 0 (`ST_USER`). Setting the value of the *ScaleType* property to 1 (`ST_PIXELS`) automatically forces the values of the *ScaleX* and *ScaleY* properties to 1. Setting the value of the *ScaleType* property to 2 (`ST_FROM_DEVICE`) forces the tool to inherit its *ScaleX* and *ScaleY* properties from the frame store.

The *ScaleX* and *ScaleY* properties of an **ImageDevice** or a **MemoryBuffer** may be inherited by tools such as a **LineCaliper** which are acting on that frame store, depending on the setting of the *ScaleType* property.

Data Type

Single

Score Property

Applies To

LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator

Description

Specifies how well the edge or edge pair indicated by the *ResultIndex* property matches the edge evaluation criteria specified by the *Setup* property.

Syntax

[*form*.]*control*.**Score**

Remarks

The value of this property is not available at design time and is read-only at run time. The score will be greater than 0 or less than or equal to 1.0. A perfect match is given a score of 1.0. Only those edges or edge pairs that score higher than the minimum accept level, defined in Edges Dialog Box, will be returned.

Data Type

Single

Setup Property

Applies To

ImageDevice, ImageHook, LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator, BlobAnalyzer, LightMeter

Description

The *Setup* property specifies the complete state of the control.

Syntax

[*form*.]*control*.**Setup** [= value$]

Remarks

The setup string is not available at design time and is not intended to be modified directly. It is a binary string containing values of the properties that define the tools state. The state of a tool includes such parameters as its Viewport position and size and the value of the properties. At run time, the value may be copied to or from string variables, allowing it to be passed from one tool to another or to be saved to a file.

The edge tools and the **BlobAnalyzer** have the Setup property present in the Properties list at design time. If the mouse is double-clicked on the property it will open a dialog box which allows the designer to change some of the attributes of the tool. During run time, setting the *Setup* property to PopUpDialog will result in the setup dialog box being displayed.

These strings are in an internal binary format. If reading or writing files in Visual Basic containing these strings, use `Get` and `Put` rather than `Print`. When using binary file access, the length of the save string must be declared explicitly. The required size varies from one tool to the next according to the following table:

Tool	**Size Required**
ImageHook	96 bytes
LightMeter	128 bytes
ArcEdgeLocator, ArcCaliper, LineEdgeLocator, LineCaliper	512 bytes
BlobAnalyzer	1024 bytes
ImageDevice, MemoryBuffer	4096 bytes

Example

```
Sub SaveData()
     Dim BlobBuffer As String * 1024
     Open(setup.dat) For Binary Access Write As 1
     BlobBuffer = BlobTool1.Setup
     Put #1, , BlobBuffer
     Close #1
End Sub
```

```
Sub LoadData()
     Dim BlobBuffer As String * 1024
     Open (setup.dat) For Binary Access Read As 1
     Get #1, , BlobBuffer
     BlobTool1.Setup = BlobBuffer
     Close #1
End Sub
```

Data Type

String

Size Property

Applies To

LineCaliper, ArcCaliper

Description

Specifies the size of the edge pair indicated by the *ResultIndex* property in user units.

Syntax

[*form*.]*Control*.**Size**

Remarks

The value of this property is not available at design time and is read-only at run time. For **LineCalipers**, this value is scaled according to the current settings of the *ScaleX*, and *ScaleY* properties.

Data Type

Single

Skew Property

Applies To

LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator

Description

Specifies the angle at which edges should be detected and displayed in a Viewport.

Syntax

[*form*.]*control*.**Skew** [= value!]

Remarks

While the *Skew* property can be used with most edge tools, it is most useful for **LineCalipers** and **LineEdgeLocators**. However, it only applies to **LineCalipers** and **LineEdgeLocators** when their Mode property is set to 2 (`MD_ANYANGLE`). Any other value of *Mode* will cause the tool to dtect only perpendicular edges. A *Skew* of 0 (default) means edges will be found perpendicular to the Viewport. The *Skew* property must be in the range [-60,60], where 0 is perpendicular to the Viewport and positive angles slope in a clockwise direction.

Data Type

Single

SortFeature Property

Applies To

BlobAnalyzer

Description

Determines which feature to sort on when sorting the list of blobs resulting from an Apply operation.

Syntax

[*form*.]*control*.**SortFeature** [= feature%]

Remarks

If the SortOrder propety is set accordingly, the tool will, after performing an *Apply*, automatically sort the resulting blob list using the values of the feature indicated by the *SortFeature* property. The Visual Basic constants included in the file XCALIPER.TXT can be used to describe the feature, avoiding the need to memorize the actual numerical values.

Data Type

Integer (Enumerated)

SortOrder Property

Applies To

LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator, BlobAnalyzer

Description

Specifies the method by which the list of image objects generated by the tool are to be sorted.

Syntax

[*form*.]*control*.**SortOrder** [= value%]

Remarks

The information that is returned by a tool is unique to each class of tools: **EdgeLocators** and **Calipers** generate edge and edge pai positions, respectively, while the **BlobAnalyzer** generates a list of blobs with associated feature values. As such, the possible values for the *SortOrder* property is also unique for each class of tools.

The following table lists the possible values for the *SortOrder* property of the **Caliper** and **EdgeLocator** tools:

Setting	Description
0 (`ERO_BYSCORE`)	Results are reported in decreasing order of the Score property. E.g., those edges (or edge pairs) scoring 1.00 will be first, those edges (or edge pairs) scoring 0.00 will be last.
1 (`ERO_BYPOSITION`)	Results are reported in increasing order of the Position property. E.g., the edge (or edge pair) appearing closest to the orgin will be reported first.
2 (`ERO_BYINVERTEDPOSITION`)	Similar to the `ERO_BYPOSITION` mode except the report order is reversed.

For the **BlobAnalyzer** tool, the possible values are as follows:

Setting	Description
0 (`ST_NONE`)	The blob list is not sorted. The blobs appear in the list according to the order in which they were extracted from the image. This will be in scan order from the upper left to the lower right corner of the Viewport.
1 (`ST_ASCENDING`)	The blob list is sorted in ascending order of the feature indicated by the *SortFeature* property.
2 (`ST_DESCENDING`)	The blob list is sorted in descending order of the feature indicated by the *SortFeature* property.

DataType

Integer (Enumerated)

Stats Property

Applies To

BlobAnalyzer, LightMeter

Description

Returns the current value of the specified statistical quantity.

Syntax

[*form*.]*control*.**Stats**(*Index*)

Remarks

This property is not available at design time and is read-only at run time. The *Stats* property is an array containing a number of useful statistical quantities computed from the data obtained from the most recent invocation of the Apply property. Note that each quantity is computed only when actually accessed through the *Stats* property. In the case of the **LightMeter** the data used are the pixel intensity values contained within the tools Viewport. In the case of a **BlobAnalyzer**, the data used are the values of the feature indicated by the CurFeature property. Note that the **BlobAnalyzer** does not support the ST_MODE, ST_LEFTTAIL, and ST_RIGHTTAIL statistics.

The following table lists the statistical measures stored in the array, and their associated index value. By including the Visual Basic header file XCALIPER.TXT in the project, the programmer can use a descriptive constant name for each index that provides easy access to the various measures.

Setting	Description
0 (`ST_MEAN`)	The arithmetic mean of the values in the data set.
1 (`ST_MEDIAN`)	The value that divides the data set into two equally sized subsets - those with values below the median and those with values above the median.
2 (`ST_MODE`) (**LightMeter** only)	The value that occurs most often within the set of values. If there is more than one value that occurs most often, the smallest value is returned.
3 (`ST_MINVAL`)	The smallest value in the data set.
4 (`ST_MAXVAL`)	The largest value in the data set.
5 (`ST_RANGE`)	The range of the values in the data set. In the **LightMeter** this is calculated as Max - Min + 1; while in the **BlobAnalyzer** it is Max - Min. This is because the **LightMeter** deals with integer values while the **BlobAnalyzer** works in floating point.
6 (`ST_SUM`)	The sum of all the values in the data set.
7 (`ST_SUMOFSQUARES`)	The sum of the square of all the values in the data set.
8 (`ST_VARIANCE`)	The variance, or mean square deviation. Note that the deviations are assumed to be computed from the true or population mean and as such the formula for the true variance is used.
9 (`ST_STDDEV`)	The true standard deviation computed as the square root of the variance.

10 (`ST_LEFTTAIL`) (**LightMeter** Only)	The smallest intensity value such that one or more (and possibly all) of its pixels are not part of the Left tail as determined by the LeftTailSize property.
11 (`ST_RIGHTTAIL`) (**LightMeter** Only)	The largest intensity value such that one or more (and possibly all) of its pixels are not part of the right tail as determined by the RightTailSize property.

DataType

Array of Single

SubPixel Property

Applies To

LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator

Description

Specifies the accuracy supported when an edge tool is translated from the point at which it is taught and the point at which it is applied.

Syntax

[*form*.]*control*.**SubPixel** [= value%]

Remarks

The property only applies to **LineCalipers** and **LineEdgeLocators** when the Mode property is set to 2 (`MD_ANYANGLE`). Otherwise, values are calculated at a one pixel resolution.

It should rarely be necessary to set this property to a value other than one but it is possible especially in arc tools. A higher resolution setting will require more teach time and more resources. The actual execution time of the tool is about the same. Increasing the accuracy of the sub-pixel translation may improve detection of arcs, especially those with a high eccentricity. However, it will not increase the accuracy of the result.

The following table lists the possible values for the *SubPixel* property:

Setting	**Description**
0 (`SP_ONE_PIXEL`)	Translation is made to the nearest pixel. This is the default.
1 (`SP_HALP_PIXEL`)	Translation is made to the nearest half of a pixel.
2 (`SP_THIRD_PIXEL`)	Translation is made to the nearest third of a pixel.
3 (`SP_QUARTER_PIXELS`)	Translation is made to the nearest quarter of a pixel.
4 (`SP_FIFTH_PIXELS`)	Translation is made to the nearest fifth of a pixel.

DataType

Integer (Enumerated)

SyncMode Property

Applies To

ImageDevice

Description

The *SyncMode* property specifies the source of the input sync signal.

Syntax

[*form*.]*ImageDevice*.**SyncMode** [= mode%]

Remarks

The following table lists the possible values for the *SyncMode* property of the **ImageDevice** control.

Setting	Description
0 (`ST_INTERNAL`)	Sync is generated internally, normally from a crystal.
1 (`ST_GENLOCK`)	Sync is overlaid on the input video signal by the camera and and stripped by a phase-locked loop. This is the normal method for synchronizing to a camera, known as composite sync.
2 (`ST_DEFAULT`)	Use the sync type selected by the driver.

Generally, a driver will either support switching between internal or external (genlock) synchronization, or it will accept the `ST_DEFAULT` mode only. If a driver uses the `ST_DEFAULT` type, camera selection and sync selection will be performed using some other mechanism. The usual possibilities are an external camera configuration program and an entry in the driver-specific section of XCALIPER.INI.

Data Type

Integer (Enumerated)

Tag Property

Applies To

ImageDevice, MemoryBuffer, ImageHook, LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator, BlobAnalyzer, LightMeter

Description

Stores a string which is intended as a unique identifier for a control.

Syntax

[*form*.]*control*.**Tag** [= string$]

Remarks

The *Tag* property is used to uniquely identify a control. Standard Visual Basic controls do not use this property in anyway; it exists simply for the convenience of the programmer.

XCaliper uses the *Tag* property in two ways: to identify controls which other controls need to access and as an informational string displayed in the title bar of Setup dialog boxes. In the latter role, it affects the Setup boxes of all the edge tools and the **BlobAnalyzer**.

The *Tag* property of an **ImageHook** or **MemoryBuffer** is used by the **ImageHook** to identify the frame store to display. The DeviceName property should contain the string stored in the *Tag* property of the frame store.

The *Tag* property is also used to specify the source for a *Command* property, such as *Copy*, which takes a source argument.

Note: When a control is created, the *Tag* property is set to be the same as the *Name* property. This can result in multiple controls having the same *Tag*. It is the responsibility of the programmer to ensure that *Tag* values are unique. ***XCaliper*** will supply a warning if a project is loaded with duplicate *Tag* values.

DataType

String

TeachMode Property

Applies To

LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator

Description

Specifies when to teach or re-teach a tool based on the size and position of the Viewport.

Syntax

[*form*.]*control*.**TeachMode** [= value%]

Remarks

The property is only applicable to **LineCalipers** and **LineEdgeLocators** when the Mode property is set to 2 (`MD_ANYANGLE`). Arc edge tools and AnyAngle line edge tools work in two distinct modes: first the tool is taught the characteristics of the region around the tool and later it is applied. Teaching the tool can take several seconds. This property determines when teaching takes place.

The following table lists the possible values for the *TeachMode* property:

Setting	Description
0 (`TM_DONT_TEACH`)	The tool is never taught except by explicit programmer action. If the taught state should become invalid, the tool will not Apply automatically and an error will be generated on an explicit Apply.
1 (`TM_TEACH_NOW`)	The edge tool is taught based on the current Viewport and property settings. After the operation is complete, the property is set to 0 (`TM_DONT_TEACH`).
2 (`TM_AUTO_TEACH`)	The edge tool is automatically re-taught whenever the state becomes invalid through a change in the size of the Viewport or modification of a property.

The time required to teach increases as the size of the Viewport increases and can reach many seconds. Hence, this property is ver useful when setting many properties and/or sizing the Viewport.

DataType

Integer (Enumerated)

TeachState Property

Applies To

LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator

Description

Specifies whether the tool has learnt its Viewport and can therefore be applied safely. This property is read-only and is only available at run-time.

Syntax

state% = [*form*.]*control*.**TeachState**

Remarks

Changes to certain properties of an arc tool, or an AnyAngle line tool can cause its internal map of the Viewport to become invalid. That is, the Viewport on which it was last taught is sufficiently different from the current one so as to cause an *Apply* of the tool to return invalid data. In fact, any attempt to apply the tool with an invalid map will result in an error being thrown.

The internal map will become invalid if a change is made to a property defining the shape, size or thickness of the Viewport. Changes to properties affecting position, colors and edge discrimination do not affect the map.

Setting	**Description**
True	The Viewport map matches the current Viewport and the tool may safely be applied
False	The Viewport map does not match the current Viewport. An attempt to apply the tool will cause an error to be generated.

This property will always be True if the *TeachMode* is set to `TM_AUTOTEACH` or if the *Mode* of a line tool is Horizontal or Vertical.

DataType

Integer (Boolean)

Thickness Property

Applies To

LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator

Description

Specifies the thickness of a Viewport, i.e. the distance across the Viewport perpendicular to the projection.

Syntax

[*form*.]*control*.**Thickness** [= value!]

Remarks

The property specifies the thickness of the Viewport to use when generating a projection. The value of the *Thickness* property is always an odd number.

Data Type

Single

Translate Property

Applies To

LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator

Description

Specifies whether to resize or translate the edge tool when any of the start, end, or mid point coordinates are set through the properties.

Syntax

[*form*.]*control*.**Translate** [= {True | False}]

Remarks

The value of this property is ignored in Line Edge tools whose Mode property is set to `MD_HORIZONTAL` or `MD_VERTICAL`.

When this property is set to True, any changes to the start, end, or mid point coordinates through the properties will cause the Viewport to be translated to the new location in the frame buffer while maintaining the size and shape. This avoids having to re-teach the tool.

When set to False, the other pick points are kept at their current position. The effect is therefore to change the shape of an arc tool Viewport, or the angle and/or size of a line tool Viewport. This will cause the tool to become untaught and will initiate a re-teach if the TeachMode property is set to `TM_AUTO_TEACH`.

Data Type

Integer (Boolean)

Transparent Property

Applies To

ImageHook, ImageDevice, MemoryBuffer

Description

Selects the way the image is displayed on the VGA screen.

Syntax

[*form*.]*control*.**Transparent** [= displayMethod%]

Remarks

This property chooses the mechanism used to display an image in a Visual Basic container. The following are the possible values for the *Transparent* property:

Setting	**Description**
0 (TM_NONTRANSPARENT)	The image is simply drawn into the VGA memory. This makes it work like the *Picture* property of the container.
1 (TM_FLOATINGWINDOW)	The container is treated as a window which looks through to the frame buffer underneath. Thus the frame buffer and Windows must be displayed on a single monitor with the hardware switching from one to the other on a pixel-by-pixel basis. Changing the location of the hooked container changes the part of the frame buffer being displayed.
2 (TM_LOCKEDWINDOW)	The container is treated as a window which looks through to the frame buffer underneath. Thus the frame buffer and Windows must be displayed on a single monitor with the hardware switching from one to the other on a pixel-by-pixel basis. The same portion of the frame buffer continuous to be displayed even if the location of the hooked container changes.
2 (TM_NOTHOOKED)	No image is displayed. This mode exists primarily to allow multiple tools to co-exist inside the same container.

Locked and Floating windows require hardware overlay. If the associated device does not support this mode, the tool will refuse to enter the requested mode. While some frame grabber cards may support overlay, **MemoryBuffers** will not. Some frame grabbers are limited in their ability to align the image buffer and VGA overlay, especially in Locked mode. If this is the case, the overlay may be shifted slightly (typically 1-3 pixels) to their left or the right. The location of edges or the position of blobs may appear slightly wrong. However, this is only a display effect. They are, in fact detected correctly.

Data Type

Integer (Enumerated)

UpdateMode Property

Applies To

ImageDevice, MemoryBuffer, ImageHook

Description

Controls when an image tool is changed to reflect new states. Not available at design time.

Syntax

[*form*.]*control*.**UpdateMode** [= {True **|** False}]

Remarks

It is sometimes desirable to update a series of frame grabber properties simultaneously, instead of one at time. This is particularly true of lookup table values. As long as the *UpdateMode* property is False, properties changed in the **ImageDevice, MemoryBuffer** or **ImageHook** are stored but no changes happen. As soon as the value becomes True, all of the deferred operations are written simultaneously thus giving the user the impression that they all happened at once.

Data Type

Integer (Boolean)

UserInterface Property

Applies To

LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator, BlobAnalyzer, LightMeter

Description

Determines to what extent the end user is allowed to modify the characteristics of a tool at run time.

Syntax

[*form*.]*control*.**UserInterface**[= mode%]

Remarks

The property allows the programmer to control whether or not the end user is able to alter, through mouse control, certain characteristics of the tool at run time. These characteristics include the position and size of the controls Viewport, as well as any information presented in the tools Setup dialog box (provided the tool has one).

The following table lists the possible values for the *UserInterface* property.

Setting	Description
0 (`UIF_FIXED`)	The tool will neither display its Setup dialog box nor allow any modifications to its Viewport.
1 (`UIF_SETUPONLY`)	The user is able to open the tools Setup dialog box, but cannot modify the tools Viewport. For controls without a Setup dialog box, this option is identical to the previous value. This setting is useful in ensuring that a tools Viewport remains stationary when the user double-clicks on the controls Viewport to open the Setup dialog box.
2 (`UIF_MOVEONLY`)	The user is allowed to change the position (but not the size) of the tools Viewport, and is not able to access its Setup dialog box.
3 (`UIF_ENABLEALL`)	Complete mouse control is enabled. The user is able to alter the Viewports position and size as well as open the Setup dialog box.
4 (`UIF_MOVEANDSIZEONLY`)	The user is allowed to change the position and size of the tools Viewport, but is not able to access its Setup dialog box. If the tool does not have a Setup box, this is equivalent to `EnableAll`.

Data Type

Integer (Enumerated)

WeightTable Property

Applies To

BlobAnalyzer

Description

Specifies the weighting applied to each pixel based on its gray level.

Syntax

[*form*.]*Control*.**WeightTable** (Index%) [= value!]

Remarks

The *WeightTable* is an array of 256 floating point values which specify the percentage of foreground and background for a pixel with a given intensity. The value for an entry into the *WeightTable* will vary from 0.0 to 1.0, where 0.0 represents pure background and 1.0 represents pure foreground.

For example, if the 120th entry of the *WeightTable* is 0.6, then a pixel with a gray level intensity of 120 will be allocated 60 percent foreground and 40 percent background. This allows the programmer to specify transition values as partially foreground and background instead of arbitrarily pick a threshold point. This leads to a more accurate calculation of blob parameters.

Common *WeightTables* can be generated using the *LowerThreshold*, *UpperThreshold* and *BlobType* properties. Any change to an entry to a *WeightTable* will cause the *BlobType* to be set to `BT_USERDEFINED`.

Data Type

Array of Single

X, Y Properties

Applies To

LineEdgeLocator, ArcEdgeLocator

Description

Specify the coordinates of the edge indicated by the ResultIndex property.

Syntax

[*form*.]*Control*.**X**
[*form*.]*Control*.**Y**

Remarks

The values of the properties are not available at design time and are read-only at run time. The results are reported relative to the containers origin and scaled according to the containers scale.

Data Type

Single

Properties Reference

All of the properties are listed below. Properties that are not standard or that require special consideration when used with this control are marked with an asterisk (*). For information on standard Visual Basic properties, please see the Visual Basic Programmer's Guide or the Visual Basic on-line Help.

A

About*
AcceptColor*
AcceptDisplay*
ActiveControl
Angle*
Apply*
ApplyOnChange*
Area*
AspectRatio*
AutoSize*

B

BlobType*
BufferSizeX*
BufferSizeY*

C

CenterX*
CenterY*
ClosedCurve*
Command*
Count*
CurAutoInc*
CurBlob*
CurFeature*
CurValue*

D

DeviceFile*
DeviceName*
DisplayMode*
DrawColor*

E

Edge1Angle*
Edge1Position*
Edge1X*
Edge1Y*
Edge2Angle*
Edge2Position*
Edge2X*
Edge2Y*

UpdateMode*
UpperLimit*
UpperThreshold*
UserInterface*

V - Z

Visible
WeightTable*
Width
X*
Y*

Commands Reference

All of the commands are listed in the following table. Commands can be executed either during design time or run time. To execute a command during design time, enter the command into Command property of the control.

Clear*

Close

Convolve*

Copy*

Dilate

Erode

LoadFile*

Median

Open

SaveFile*

TuneInput*

Event Reference

Applied*

Click*

DblClick*

MouseDown*

MouseMove*

MouseUp*

Methods Reference

All of the methods supported by ***XCaliper*** are listed in the following table.

Move*

Refresh*

ZOrder*

Error Messages Reference

The following tables list the error codes that can be produced by the Xiris VBXes:

Error Number	Error Description
32004	Attempt to set value of read-only property.
32005	Attempt to set property to illegal value.
32006	Cannot Genlock to channel; no sync available.
32007	Cannot obtain handle to container.
32008	Attempt to set property to illegal value.
32014	Color bitmaps not supported.
32015	Compressed files not supported.
32016	Too many blobs to process.
32017	No imaging control with specified tag could be found.
32018	Window initialization failure.
32019	Result index out of range.
32020	Either no results found at last apply or no apply ever performed.
32021	*Command* property used without a properly constructed verb.
32022	Comnmand verb unknown.
32023	Wrong number of arguments supplied to Command.
32031	Value must be greater than zero.
32032	Call not supported for this kind of object.
32033	Device is not a Frame Grabber.
32034	Processing region does not overlay input region.
32035	Call requires UpdateMode to be True.
32036	Too many entries used in color table; cannot add new one.
32037	Internal Logic Error.
32038	Attempt to access a blob feature which has not been calculated.
32039	The bit width of this image is not supported.
32040	The kernel requested was not found in the .INI file or there is no .INI file.
32041	Convolution kernel format is wrong.
32042	The sum of the elements of a convolution kernel may not be negative.
32043	An attempt to open the specified device driver failed; DLL may not exist.
32044	Attempt to initialize the specified driver failed.
32045	Attempt to use color processing on a card which does not support it.
32046	Attempt to use color processing on an image which does not support it.
32047	Attempt to load an **ImageDevice** with an invalid checksum.
32048	Attempt to load an unexpected version of an ImageDevice.
32049	Attempt to access an image with zero size.
32050	Attempt to load an **ImageHook** with an invalid checksum.
32051	Attempt to load an **ImageHook** from a setup from a different version.
32052	Attempt to load an unexpected version of an edge tool.
32053	Attempt to load an edge tool with an invalid checksum.
32054	Attempt to use an XCaliper control in design mode without a protection key.
32055	The software protection key does not authorize use in design mode.
32056	Attempt to load a LightMeter with an invalid checksum.
32057	Attempt to load an unexpected version of the LightMeter.
32058	Image Access generates Viewport into color image; not supported.
32059	Attempt to load an unexpected version of a **MemoryBuffer**.
32060	Attempt to load a **MemoryBuffer** with an invalid checksum.
32061	Container already has an enabled hook. Only one is allowed.
32062	Attempt to load an unexpected version of a **BlobAnalyzer** tool.
32063	Attempt to load a **BlobAnalyzer** tool with an invalid checksum.
32064	Attempt to load a **BlobAnalyzer** setup string with the wrong length.

32065	Attempt to load an **ImageDevice** setup string with the wrong length.
32066	Attempt to load an **ImageHook** setup string with the wrong length.
32067	Attempt to load an **EdgeLocator** setup string with the wrong length.
32068	Attempt to load an **LightMeter** setup string with the wrong length.
32069	Attempt to use continuous grab emulation without a non-transparent hook.
32070	Attempt to use continuous grab on a board without a secondary display.
32076	Property is not allowed to be set to this value in design mode.
32077	Filtering operation kernel is larger than viewport on which it is to be applied.
32078	Cannot apply too when TeachState is False.

Standard Visual Basic Error Codes

Error Number	**Error Description**
7	Out of memory
52	Bad file name or number
53	File not found
54	Bad file mode
58	File already exists
61	Disk Full
64	Bad file name
67	Too many files open
68	Device Unavailable
70	Permission denied
76	Path not found
481	Invalid Picture Type

Factors Affecting Performance: Overview

There are many factors which can affect the Speed and Accuracy of ***XCaliper*** in a measurement application.   A thorough understanding of these factors and how they relate can help the programmer to obtain maximum performance from ***XCaliper*** in any measurement application.

Factors Affecting Performance: Speed

Speed is defined as the run time required to execute a tool and obtain a result inside an application. To be able to understand how to optimize the execution speed, it is necessary to be aware of the various steps of the processing and measuring process.

Edge Tools

Once an image has been acquired and is resident in a frame buffer, a region of interest must be read to generate a projection of gray level intensities in the region. When this curve has been generated, the system CPU processes the projection to find the locations of maximum edge strength, or edges. Finally, the list of edges is evaluated to find those that match a set of input criteria. For a more complete discussion of the process steps of an edge tool, refer to the Edge Tools Theory of Operation.

Blob Tools

Once an image has been acquired and is resident in a frame buffer, a region of interest is accessed, pixel by pixel by the system CPU, with the result being placed in the computer's system memory. From this point on, all the processing is done using the personal computer's CPU, namely the steps of segmentation, labelization, feature extraction, filtering, and display. For a more complete discussion of the process steps of a blob tool, refer to the BlobAnalyzer Theory of Operation.

A number of additional considerations are necessary to optimize the execution speed of ***XCaliper*** tools. These include:

Hardware Considerations
Setup Considerations

Factors Affecting Performance: Hardware Considerations - Speed

The process of generating a projection is by far the largest portion of the cycle time of an edge tool, taking up to 90% of the total time required when running the tool in software. This is governed by the access time of reading the pixels in the frame buffer, and the speed of the processor performing the projection operation. To maximize the speed of both these steps, a frame grabber should be chosen that can perform profiling in hardware, and can then send the projection result to the system CPU. Should the frame grabber not have on-board processing, the system CPU will be used to perform the projection operation which must access the frame grabber's memory to process the image.

The system CPU is usually used exclusively for the BlobAnalyzer and the LightMeter tools. In each case, the rate of pixel access is dependent on the system's bus: an EISA based frame grabber will allow the CPU to access the frame buffer at EISA bus clock speeds of up to 33 MHz, whereas an ISA based frame grabber is restricted to 8 MHz bus speed. Other frame grabber boards that are based on VL or PCI local bus standards can operate even faster. Other than access time, the speed of the CPU directly affects the time taken to generate the projection curve of an edge tool, or to perform the labelization of the image for the **BlobAnalyzer**. In any case, if processing is to be done in software, the execution speed of any tool can be improved using the fastest CPU possible with a cache of at least 128 KBytes.

Factors Affecting Performance: Setup Considerations - Speed

Because generating a projection an edge tool or performing a segmentation using the BlobAnalyzer is the most time consuming step in performing a measurement, the speed of execution of ***XCaliper*** is directly proportional to the shape, size and inclination of its Viewport.   For an edge tool, the first concern should be to minimize the Viewport width (or height of columns which are totaled to calculate the projection).   Secondly, the length of the Viewport, while not as significant, can help to reduce the overall cycle time.   In other words, a Viewport 50 pixels wide and 100 pixels long will be faster to projection than a Viewport 100 pixels wide and 50 pixels long.   Finally, inclination of the Viewport is also significant to the execution speed of XCaliper: a Viewport that has its major axes along one of the cardinal axes (e.g. 0, 90, 180, 270) will operate much faster than one with its axes along an intermediate angle because no interpolation will have to occur.

The edge tools can also be made to run slightly faster by fine tuning the parameters for inspection when designing the application: when numerous edges are found in the Viewport, the tool will take slightly longer to evaluate all the edges to find the ideal edge or edge pair.   By activating the discriminators such as minimum strength, pair straddle, etc., the Edge Evaluation Functions will not have to operate on as many edges.

For the **BlobAnalyzer**, the smaller the Viewport is around the desired blob, the faster the tool will run. As the processing time varies with the size of the Viewport, leaving extra space around a blob will add extra processing time.   In addition, if repeated filtering operations or feature extraction is to be done on the same image, the tool should be set up to avoid the processing steps that are not required over and over, e.g. segmentation or labelization.

Factors Affecting Performance: Accuracy

Accuracy and repeatability are the two main metrics by which any measurement tool is rated.   Accuracy and repeatability are closely related: the accuracy of a process cannot be correctly calculated without first determining the repeatability of the process.

A number of considerations are necessary to ensure accuracy of an ***XCaliper*** tool.   These include:

Hardware Considerations

Optical Considerations

Lighting Considerations

Setup Considerations

Factors Affecting Performance: Hardware Considerations - Accuracy

The analog video signal produced by a video camera is a voltage proportional to the light intensity found on the camera's sensor. Additional information exists in the video signal to define the timing of the scan line, known as the horizontal sync pulse. When the video signal enters a frame grabber, it is digitized by flash Analog-to-Digital converters (ADCs) at pixel rates (approx. 14.3 Megapixels/sec). The ADCs attempt to lock sync with the video camera by picking up the horizontal sync pulse using Phase-Locked-Loop (PLL) circuitry. Because of the nature of this process, an error is introduced into the digitized image known as pixel jitter.

Each time the location of the horizontal sync differs from the locked sync, the position of all the horizontal picture elements is shifted by an equal amount in the scan line. A large error in the horizontal sync could cause the location of a single row of pixels to be off by one or more pixels in one scan, and zero in the next, hence the name pixel jitter. Significant pixel jitter can affect an edge tool that has been placed in the horizontal direction. When selecting a frame grabber, careful attention should be made of the published pixel jitter. When pixel jitter does exist, vertically oriented edge tools will have higher accuracy than horizontal edge tools.

Once the video signal is digitized, the resulting digital data is stored in video RAM and can be accessed by either an on board processor or the computer's CPU. Each picture point in the image is referred to as a pixel. The number of pixels in an image defines the resolution of an image and the number of bits used to store the brightness information defines the depth of the image. Each pixel has a numerical value that represents the average brightness of the image over the area that the pixel occupies in the original analog image, where 0 represents black, 255 represents white, and the values in-between all the shades of gray (for an 8 bit memory buffer). Although the camera has discrete physical pixels, the exact timing between pixels is lost due to the time delay between transmitting that pixel and digitizing its value, causing a temporal error.

By increasing the resolution of the image by using a larger frame buffer, a measurement can be made more accurate. However, this has a practical restriction in that there is no point to digitize an image into a frame buffer whose resolution is greater than that of the camera sensor from whom the image was derived. To represent a perfect edge in an image, at least two pixels are required on a scan line. A perfect edge can only be generated by a perfect alignment of the image with the pixel array using a perfect optical system, which happens rarely. Based on Nyquist's sampling theory, we can say that a 768 X 493 pixel camera can display up to 383 X 246 distinct lines, or objects in the image. A typical image has much less information content than this number.

Because physical pixels are not the same sizes horizontally and vertically, and the digitization process does not occur on pixel boundaries, any pixel based measurements do not give the same calibration value horizontally and vertically. Therefore, the correlation between the photosites (camera pixels), and the digitally stored pixels can only be approximated by calibrating the pixel size in both the horizontal and vertical directions.

Factors Affecting Performance: Optical Considerations - Accuracy

Optics, the study of light and behavior of optical elements, is a very important part of machine vision. The input to a machine vision system is almost exclusively via optics. A scene or object is imaged onto a camera sensor. Well designed optics will improve the chances of creating a successful system, and will significantly reduce the software effort required to bring the project to completion. Conversely, many machine vision projects fail, or are late because of poor optical design. Software cannot compensate for poor lighting, out of focus images, or incorrectly chosen perspectives. For the above reasons, optics will be discussed in detail with emphasis placed on lenses.

The camera sensor, or CCD, typically come in sizes of 1/3", 1/2", and 1". A common mistake is to assume a 1/2" CCD has 1/2" diagonal sensing distance. In reality, about half of the chip is used for storage, and the rest for image acquisition. For a 1/2" CCD, the usable diagonal area for image acquisition is approximately 0.31" (8mm).

The optical path between the camera's sensing device and the component under inspection is critical for an accurate measurement system. The physical set up of the camera and component under inspection can greatly affect accuracy: should the camera not be perpendicular to the component under inspection for example, a linear distortion will occur across the image. Similarly, should the distance between the component and the camera from one inspection to the next change significantly relative to the focal length of the application, a further process inaccuracy will develop. The choice of optic components to use in a gauging application, such as lenses, filters and mirrors must be carefully made to avoid introducing optical aberrations to the system. Careful selection of the optical components must be made to avoid optical aberrations such as:

Spherical - When rays from the center and edges of the lens focus at different distances.

Chromatic - When different gray level intensities focus at different distances or depths. Lens may have different sensitivities to specific light wavelengths.

Coma - When an off-axis image appears to have asymmetric blurring which is comet-like in shape, such as an uneven spot.

Astigmatism - When horizontal and vertical features focus at different depths.

Distortion - When there is a difference in lateral magnification, usually appearing as a Barrel or Pincushion image. Can be minimized by avoiding wide angle lenses.

There are four fundamental equations that need to be considered when designing lens systems. Two of the equations only apply when the optical system is focused. The equations are derived for simple lens systems, but will give a reasonable starting point when used on a complex one.

1) **Basic lens equation:**

$$1/f = 1/s + 1/s'' \quad \text{(when in focus)}$$

where f is the focal length of the lens

s is lens to object distance

s'' is lens to sensor distance

2) **Magnification:**

$$m = s''/s \quad \text{(when in focus)}$$

where m is the magnification of the lens s, s" are the same as above

3) Refraction index of a beam at a glass/air boundary:

$n = \sin(i) / \sin(r)$

where i is the angle to the incoming beam

r is the angle of the refracted beam

n is the refractive index of the glass (typically 1.52 for optical glass.)

The refractive index is the ratio of the speed of light in a vacuum to the same glass. Light slows down in glass.

4) Compound lens equation:

(This applies when two simple lenses are combined.)

$1/f = 1/f1 + 1/f2 - d/f1*f2$

where f is the resultant focal length

f1 is the focal length of lens #1

f2 is the focal length of lens #2

d is the distance between the principal points on the lenses

The above equations will allow you to calculate most optical systems involving simple lenses, glass plates, windows, and prisms. Equations (1) and (2) can be used to calculate the kind of camera lens to be selected for a given job and the length of the extension tube to be applied with the lens. Frequently, solving two simple simultaneous equations will be required for extension tube calculations.

The light gathering power of lenses depends on the ratio of active diameter to focal length. This is called f-number. The active diameter of the lens is called aperture, which is adjustable in camera lenses. The aperture scale engraved on the lens is a series of values: 2, 2.8, 4, 5.6, 8, 11, 16, etc. These numbers represent the inverse of the actual f-number and really mean: 1/2, 1/2.8, etc. and are part of a series based on the square root of 2. Each increment in the series halves the light transmitted by the lens. The aperture of the lens, which is also referred to as f-stop, is adjusted by two or more thin blades inside the compound lens. The aperture works on the entire image, so setting the aperture to a small opening does not just cut the light on the perimeter of the image, but dims the entire field of view. Reducing the aperture also increases the **depth of field**. So there is a trade-off between brightness of a scene and the depth of field.

Camera Lenses

The use of standard camera lenses has several advantages over the use of simple lenses in a video imaging system:

1) Spherical and chromatic aberrations are corrected for in the design

2) Built-in focusing adjustment

3) Built-in Adjustable aperture

4) Standard screw-mount supplied, usually 'C'-mount

5) C'-mount is a recognized standard: 1" dia X 32 TPI, with flange to sensor distance of 17.53 mm

6) Video lenses are available in standard focal lengths: 16 mm, 25 mm, 50 mm, 75 mm, although for most machine vision applications 25mm or 50mm lenses are used due to their superior optics.

Zoom (variable focal length) lenses are impractical for precise machine vision applications because of their design: with loose fitting internal adjustable lens elements, the lens axis can be thrown off its ideal center line; and the scale factor of the lens/camera system can never be guaranteed due to the moving zoom action of the lens. A further limitation of zoom lenses is that they are generally not designed for close focus work.

Microscope Objectives

For large magnifications ordinary camera lenses are not suitable, as they are not designed for such applications. A better solution would be to use commercially available microscope objectives from large microscope vendors. They do not have adjustable focus and aperture, but will produce sharp images at high magnifications. Some, such as the Nikon measuring microscope objective, will allow the designer to work at a significant distance from the object and yet obtain high magnifications. This feature is achieved through a very complex optical train inside the lens barrel.

Other Optical Devices

Mirrors, flat or curved, come in two versions: front surfaced and rear surfaced. In all applications where imaging is done through a mirror, front coating should be chosen for optimum image quality. When looking through a glass plate 4% of the light is reflected off the front face of the glass. In a rear coated mirror two such reflections occur, one on entry, the other on exit. All these extra reflections contribute to image degradation. Front faced mirrors do not exhibit the same behavior. Rear surfaced mirrors are generally more robust and can be used in optically less demanding applications, such as reflectors for light sources

Beam-splitters are generally made with glass plate that has one side coated with a partial light transmitting layer, while the other side is anti-reflection coated. Beam-splitters are useful in combining two or more optical systems to create complex systems such as light source and image travelling in the same optical path.

Windows are generally anti-reflection coated glass plates and are useful in keeping dirt, etc. out of an optical system. For very abrasive environments they may be made of synthetic sapphire at a reasonable cost.

Prisms are generally used to replace mirrors in special applications. Because of their well known dispersive properties (color separating) they are also used in spectral analysis.

Factors Affecting Performance: Lighting Considerations - Accuracy

The selection of an appropriate lighting technique for a machine vision project will greatly simplify the software effort and enhances the success of the entire project. There are no rules in selecting the proper lighting for the job, the best way is often found by experimenting with a variety of techniques. The goal is to highlight the feature of interest without illuminating areas of no concern.

An image typically consists of surfaces and edges, where each surface in the image has a different reflectance and creates a unique brightness level. Lighting is one of the most important considerations when setting up any machine vision system, particularly a gauging application. The way that light bends around and reflects off objects is extremely variable between any two parts. The main objective when choosing a lighting method for gauging is to illuminate the target in such a way as to generate edges in the image that are as sharp as possible. Sharp edges are ideal for highly accurate measurements because the edge locating algorithm will find the maximum rate of change of strength across an edge more consistently when the edge is sharper. The steeper the edge gradient, the less the algorithm is influenced by minor intensity changes across the edge.

To achieve the best measurement results, a consistent light source should be used that can provide as close as possible to constant intensity. Light sources that are driven by DC, strobe or high frequency AC power supplies are recommended. By minimizing the variation in light intensity, the strength projection across an edge will be more consistent, thereby improving both the repeatability and accuracy of the measurement. The ideal light source to use for gauging is a backlight with a collimated cover to generate only parallel light rays with a minimum amount of distortion by the time they hit the camera's sensor. By using such a light source, more accurate edges can be obtained, and image aberrations such as flare (hot spots or uneven illumination caused by reflections from part of the optical equipment) and vignetting (uneven illuminated image, such as a bright center spot with dark edges) can be minimized. The lighting and optics to be used for a measurement application should be designed so that no pixels in the image are at their maximum level as this could cause saturation or blooming (when charge from one pixel spills over to the adjacent pixels, causing a smearing of the region) on the camera pick up sensor.

When choosing a lighting method, there are a few basic considerations:

1) Low angle lighting enhances contrast exhibited by the surface structure of an object.

2) High angle lighting enables more precise feature measurements.

3) Certain 3-dimensional features can be enhanced using selective lighting.

4) Highly reflective, mirror like objects can be best viewed using collinear or co-axial lighting normal to the surface.

5) 3-Dimensional contours show up well under structured (zebra-stripe) lighting.

6) Polarizers may eliminate unwanted reflections at the cost of loss of light.

7) One polarizer in front of the light source, the other in front of the lens at 90 degrees to the first will allow otherwise impossible situations to work. This method is called Polarizer/Analyzer technique.

8) Ambient light can be excluded by the use of monochromatic light, such as LEDs combined with optical filters of the same spectral band.

Factors Affecting Performance: Setup Considerations - Accuracy

The way in which a Viewport is set up during the design phase can also have an impact on the accuracy of the measurement.   Because a single edge tool can be subject to noise and local discontinuities in the image, the edge tool was designed with a two dimensional window, or Viewport, from which measurements can be made.   This provides the programmer some flexibility in that the Viewport can be sized to suit: by making the Viewport several lines in width, an averaging effect can be achieved to provide a better, overall edge location.   However, this can be a problem if the Viewport is made too wide because the resulting edge projection may be too weak for proper detection or accurate location of the edge.

The size of the filter used to locate edges in the projection can also influence accuracy of the tool: a larger filter size will tend to smooth the projection curve, taking into account information about the edge farther away from the actual edge location and therefore slightly change the location of the edge found. In addition, a larger filter size may smooth the projection so much that it may desired edges are removed from the image.

Methods

All of the methods are listed in the following table.   For information on standard Visual Basic methods, please see the Visual Basic Programmer's Guide or the Visual Basic on-line Help.

Move*

Refresh*

ZOrder*

Move Method

Applies To

LightMeter, LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator, BlobAnalyzer

Description

This method is used to move and size a control.

Syntax

[object.]**Move** left[, top[, width[, height]]]

Remarks

The Move method has these parts:

Part	**Description**
object	Form or control to move. May be any control except timers and menus.
left	Single-precision value indicating the horizontal coordinate for the left edge of object.
top	Single-precision value indicating the vertical coordinate for the top edge of object.
width	Single-precision value indicating the new width of object.
height	Single-precision value indicating the new height of object.

Only the left argument is required. However, to specify any other arguments, you must specify all arguments that appear in the syntax before the argument you want to specify. For example, you cannot specify width without specifying left and top. Any trailing arguments that are unspecified remain unchanged.

For forms and for controls within frames, the coordinate system is always twips. Moving a form on the screen or moving a control within a frame is always relative to the origin (0,0), which is the upper-left corner. When moving controls on a form or in a picture control, the coordinate system of the object is used. The coordinate system is set using ScaleHeight, ScaleWidth, ScaleLeft and ScaleRight properties.

Refresh Method

Applies To

LightMeter, LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator, BlobAnalyzer

Description

This method is used to refresh a control.

Syntax

[formname. **|** controlname. **|** objectname.]**Refresh**

Remarks

Use this method to force a complete repaint of a form or control. This is useful when you want a form to display completely while another form is loading, or when you want to update the contents of a system list box, such as a file list box, or the data structures of a data control.

Generally, painting a form or control is handled automatically while no events are occurring. However, there may be situations where you want the form or control updated immediately. For example, if you use a file list box, a directory list box, or a drive list box to show the current status of the file system, you can use Refresh to update the list whenever a change is made to the directory structure.

ZOrder Method

Applies To

LightMeter, LineCaliper, LineEdgeLocator, ArcCaliper, ArcEdgeLocator, BlobAnalyzer

Description

This method is used to change the z-order of a control.

Syntax

[object.]**ZOrder** [position]

Remarks

Integer indicating the position of the control relative to other controls. If position is 0 or omitted, the control is positioned at the front of the z-order. If position is 1, the control appears at the back of the z-order.

The z-order of controls can be set at design time using Bring To Front and Send To Back choices on the Edit menu.

Note: There are three graphical layers associated with forms and containers. The back layer is the drawing space, where the results of the graphics methods appear. Next is the middle layer where graphical controls and labels appear. The front layer is where all non-graphical controls like command buttons, check boxes, and file controls appear. Anything in one layer covers anything in the layer behind. The **ZOrder** method arranges controls only within the layer where the control appears.

LineCaliper

See Also Properties Events Methods

Description

The **LineCaliper** detects Edge Pairs inside a Viewport. The Viewport can be oriented at various angles to suit the application. The program designer and/or user provides a series of constraints which describe the ideal edge pair to be found and the tool reports back the position and size of each edge pair. Constraints are modified using the Setup Dialog Boxes.

File Name

XVBXELOC.VBX

Remarks

The control is represented on the Visual Basic toolbox by an icon depicting an actual Caliper (the mechanical device used to measure the size of an object). When the control is placed in a container, it is represented by a rectangular box. The search for an edge pair is performed inside this box.

Properties

All of the properties are listed in the following table. Properties that are not standard or that require special consideration when used with this control are marked with an asterisk (*). For information on standard Visual Basic properties, please see the Visual Basic Programmer's Guide or the Visual Basic on-line Help.

About*
Apply*
ApplyOnChange*
DrawColor
Edge1Position*
Edge1X*
Edge1Y*
Edge2Position*
Edge2X*
Edge2Y*
EdgeDisplay*
ElapsedTime*
EndPointX*
EndPointY*
Height
Inclination*
Index
Left
LockAngle*
MaxResults*
Mode*
Name
NumResults*
PathLength*
Position*
RefType*
ResultIndex*
ScaleType*
ScaleX*
ScaleY*
Score*
Setup*
Size*
Skew*
SortOrder*
StartPointX*
StartPointY*
SubPixel*
Tag*
TeachMode*
TeachState*
Thickness*
Top
Translate*
UserInterface*
Visible
Width

See Also

LineEdgeLocator

ArcCaliper

See Also Properties Events Methods

Description

The **ArcCaliper** detects Edge Pairs inside a Viewport. The Viewport can be configured as an open or closed (e.g. a circle) arc for a range of Aspect Ratios. The program designer and/or user provides a series of Constraints which describe the edge pair to be found and the tool reports back the Position and Size of each edge pair. Constraints are modified using the Setup Dialog Boxes.

File Name

XVBXELOC.VBX

Remarks

The control is represented on the Visual Basic toolbox by an icon depicting an actual Caliper (the mechanical device used to measure the size of an object) superimposed over a circle. When placed in a container, it is represented by an arc. The search for an edge pair is performed along this arc. Also, in the case of closed curve Viewports, the edge extraction process wraps around the start point such that edges and edge pairs that span the start point will be detected.

Properties

All of the properties are listed in the following table. Properties that are not standard or that require special consideration when used with this control are marked with an asterisk (*). For information on standard Visual Basic properties, please see the Visual Basic Programmer's Guide or the Visual Basic on-line Help.

About*
Apply*
ApplyOnChange*
AspectRatio*
CenterX*
CenterY*
ClosedCurve*
DrawColor
Edge1Angle*
Edge1Position*
Edge1X*
Edge1Y*
Edge2Angle*
Edge2Position*
Edge2X*
Edge2Y*
EdgeDisplay*
ElapsedTime*
EndPointX*
EndPointY*
Height
Index
Left
MaxResults*
MidPointY*
MidPointX*
Name
NumResults*
PathLength*
PathSize*
Position*
ResultIndex*
Score*
Setup*
Size*
Skew*
SortOrder*
StartPointX*
StartPointY*
SubPixel*
Tag*
TeachMode*
TeachState*
Thickness*
Top
Translate*
UserInterface*
Visible
Width

See Also

ArcEdgeLocator

LineEdgeLocator

See Also Properties Events Methods

Description

The **LineEdgeLocator** detects one or more Edges in a Viewport. The Viewport can be oriented at various angles to suit the application. The program designer and/or user provides a series of constraints which describe characteristics of the edge(s) to be found and the tool reports back the position of each edge. Constraints are modified using the Setup Dialog Boxes.

File Name

XVBXELOC.VBX

Remarks

The control is represented in the Visual Basic toolbox by an icon depicting a box in the shape of a line with edge markers appearing at points of change of contrast. When the control is placed in a container, it is represented by a rectangular box. The search for single edges is performed inside this box.

Properties

All of the properties are listed in the following table. Properties that are not standard or that require special consideration when used with this control are marked with an asterisk (*). For information on standard Visual Basic properties, please see the Visual Basic Programmer's Guide or the Visual Basic on-line Help.

About*
Apply*
ApplyOnChange*
DrawColor
EdgeDisplay*
ElapsedTime*
EndPointX*
EndPointY*
Height
Inclination*
Index
Left
LockAngle*
MaxResults*
Mode*
Name
NumResults*
PathLength*
Position*
RefType*
ResultIndex*
ScaleType*
ScaleX*
ScaleY*
Score*
Setup*
Skew*
SortOrder*
StartPointX*
StartPointY*
SubPixel*
Tag*
TeachMode*
TeachState*
Thickness*
Top
Translate*
UserInterface*
Visible
Width
X*
Y*

See Also

LineCaliper

ArcEdgeLocator

See Also Properties Events Methods

Description

The **ArcEdgeLocator** detects one or more Edges in a Viewport. The Viewport can be configured as an open or closed (e.g. a circle) arc for a range of Aspect Ratios. The program designer and/or user provides a series of Constraints which describe characteristics of the edge(s) to be found and the tool reports back the Position of each edge. Constraints are modified using the Setup Dialog Boxes.

File Name

XVBXELOC.VBX

Remarks

The control is represented in the Visual Basic toolbox by an icon depicting a box in the shape of an arc with edge markers appearing at points of change of contrast, as shown above. When the control is placed in a container, it is represented by an arc. The search for single edges is performed along this arc. Also, in the case of a closed curve Viewport, the edge extraction process wraps around the start point such that edges on or near the start are detected regardless of the edge filter size.

Properties

All of the properties are listed in the following table. Properties that are not standard or that require special consideration when used with this control are marked with an asterisk (*). For information on standard Visual Basic properties, please see the Visual Basic Programmer's Guide or the Visual Basic on-line Help.

About*
Angle*
Apply*
ApplyOnChange*
AspectRatio*
CenterX*
CenterY*
ClosedCurve*
DrawColor
EdgeDisplay*
ElapsedTime*
EndPointX*
EndPointY*
Height
Index
Left
MaxResults*
MidPointX*
MidPointY*
Name
NumResults*
PathLength*
PathSize*
Position*
ResultIndex*
Score*
Setup*
Size*
Skew*
SortOrder*
StartPointX*
StartPointY*
SubPixel*
Tag*
TeachMode*
TeachState*
Thickness*
Top
Translate*
Visible
Width
X*
Y*

See Also

ArcCaliper

Events

All of the properties are listed in the following table. Properties that are not standard or that require special consideration when used with this control are marked with an asterisk (*). For information on standard Visual Basic properties, please see the Visual Basic Programmer's Guide or the Visual Basic on-line Help.

Applied*
Click*
DblClick*
GotFocus
KeyDown
KeyPress
KeyUp
LostFocus
MouseDown*
MouseMove*
MouseUp*

Edge Tools Theory: Overview

XCaliper provides four types of edge tools: a Line EdgeLocator, a Line Caliper, an Arc EdgeLocator, and an Arc Caliper. Each of these ***XCaliper*** edge tools operate on a region of an image called a Viewport. The Viewport, defined by the program or the user, contains the Path along which the edge tools locate and measure edges.

Linear edge tools operate in one of three modes: Horizontal, Vertical, and AnyAngle. This distinction is made because a horizontal or vertical Viewport is orthogonal with respect to the Frame Buffer axes, meaning there will be a one-to-one correspondence between the pixels in the frame buffer and those in the Viewport. They therefore are optimized versions that bypass the interpolation techniques required for non-orthogonal tools. ***XCaliper*** defines Horizontal as left-to-right and Vertical as top-to-bottom (0 and 270 respectively). The Viewport can be moved and sized anywhere on the image until it contains all of the edges that are to be analyzed.

The edge tools take advantage of the fact that differences in brightness will be most rapid around an Edge in an image. If rapid changes in brightness can be found, the edges within a Viewport can also be found. To minimize the effects of Noise in an image, the gray level information of a *region* is used, rather than of one line.

A series of steps are used by the ***XCaliper*** edge tools to find and process edges:

i) Generating a Projection
ii) Generating an Edge Strength Array
iii) Determining Edge Location
iv) Applying Discriminators
v) Applying the Scoring Engine

Edge Tools Theory: Generating a Projection

After the Viewport has been defined, it is processed into a one dimensional array of light intensities, known as a Projection. The number of elements in this array corresponds to the length of the Viewport in pixels. Each element in the projection array represents the average brightness of a line of pixel intensities at a specified angle relative to the Viewport. The Viewport should be sized so that its thickness is not greater than the length of the edge(s) to be measured. In so doing, the average brightness of a line of pixel intensities will be a true representation of the brightness across the edge(s) desired. The projection of the Viewport can be viewed in the Projection Window of the Edges Dialog Box.

To generate a projection from an orthogonal Viewport, the Frame Buffer is simply scanned in the appropriate direction and the results stored into the projection. However, non-orthogonal Viewports require use of a mapping technique between the pixels in the Viewport and the points in the projection. To improve performance, generation of this mapping is separated out into a distinct Teach operation which is a long, computationally intensive process. By separating the Teach operation where not required, edges can be extracted from a non-orthogonal Viewport almost as fast as from an orthogonal one.

To simplify the distinction between generating the map and extracting the edges, the edge tools support a *TeachMode* property as well as a *TeachState* property. When *TeachMode* is set to `TM_AUTO_TEACH`, an edge tool will re-generate the mapping whenever a modification to its state causes the mapping to become invalid. Otherwise, the mapping is only generated by explicit programmer action. The mapping becomes invalid whenever the shape of the Viewport or the Skew angle changes. The *TeachState* property is a read-only Boolean that permits the developer to query the tool as to whether or not the mapping is currently valid. If the property returns True then the tools mapping is valid and can be safely applied. If it returns False, the tool must be retaught (using the *TeachMode* property) before being applied. An attempt to apply the tool without reteaching will generate an error.

The angle at which a projection is generated is defined by the *Skew* property which ranges from (-60º,60º). A skew of 0º indicates that pixel intensities should be averaged at an angle perpendicular to the Viewport. Skews increase in a clockwise direction and decrease in a counter-clockwise direction. The *Skew* property is ignored in orthogonal Viewports because if it is set to something other than 0º, the one-to-one correspondence to the frame buffer is lost. Although the *Skew* property is supported in Arc tools as well, non-zero skews are more commonly used in linear tools.

Before searching the edges in a non-orthogonal Viewport, the edge tool position must be translated from the point in the buffer at which it was taught to the point in the buffer at which it will be applied. In order to avoid invalidating the mapping, the translation must be of a whole number of pixels. That is, if a map was generated at location (1.2, 3.7), then a translation to (3.2, 4.7) is OK but a translation to (3.3, 4.7) is not. In order to avoid constantly re-teaching the part, ***XCaliper*** limits the precision of the translation. The *SubPixel* property selects this limit, the default being to round the translation to the nearest pixel. Increasing the precision causes additional mappings to be generated, effectively allowing more precise translations.

The effect of a more precise translation is not to increase the Accuracy of edge location but rather the probability that the edge will be detected at all. However, there is rarely any reason to change the default because the tool is very good at detecting edges unless dealing with extremely narrow edges and/or highly elliptical arcs. Increasing the sub-pixel precision is costly; CPU and memory resources used for generating maps and the time taken to Teach increase in proportion to the square of its value.

Edge Tools Theory: Generating an Edge Strength Array

Once the Projection is generated, a Differential Filter, whose size is defined in the Edges Dialog Box, is applied to the projection to create an array of new values that indicate the change in light intensity over a number of pixels. This is done by performing a Convolution of a one dimensional differential filter across the projection. The result is an Edge Strength Array. In the convolution, for each element in the projection array, the elements of the filter are multiplied by their corresponding elements in the projection array, and the products are summed together to create an element in the Edge Strength Array.

The absolute value of the resulting array is taken to allow all peaks to be displayed in the Edge Strength Window of the Edges Dialog Box. The sign of each value in the array yields the polarity of the edge. The values in this array are known as the edge strength, where peaks in the graph of the array correspond to locations of edges in the Viewport, and the height of the peaks in the graph of the array corresponds to the strength across the edge. A peak of greater height therefore corresponds to an edge of greater strength.

A filter of a larger size has the effect of smoothing the edge strength graph since the change in light intensity is found over a greater number of pixels. The effect of changing the filter size can be seen in real time by viewing the graph of the edge strength located in the Edge Strength Window of the Edges Dialog Box. The filter size should be chosen so that the total width of the filter is roughly equal to the width of the edge in pixels. The edge finding algorithm defines the filter size to be = 1/2(filter width - 1). Choosing a small filter size will result in finding edges whose change in light intensity span just a few pixels, conversely choosing a larger filter size will result in finding edges whose change in light intensity spans a larger number of pixels. If the change in light intensity of the pixels surrounding an ideal edge spans a large number of pixels, and a small filter size is used, it is likely that the ideal edge will be found along with several other edges found as a result of noise across the edge. Generally speaking, a filter size of 2 or 3 is ideal for most applications.

Edge Tools Theory: Determining Edge Location

The algorithm for locating Edges finds the local maxima of the edge strength array to sub-pixel precision.   Like the output of any Difference Filter, peaks of the difference array are found at the same locations of the most rapid rate of change of the original curve.   Consequently, by finding the peaks in the edge strength array, the points of maximum rate of change (i.e. edges) can be found.   The technique used is capable of finding the edge to a high degree of precision, 1/8 pixel or more, based on the type of application.

Once the edges have been found in an image, the next step is to evaluate the edges to determine which, if any, are suitable for measurement.   This is done through the Edge Evaluation Criteria.

Edge Tools Theory: Edge Evaluation Criteria

The Edge Evaluation Criteria evaluates Edges which have been found in an image to determine if any are suitable for measurement.   This is done by comparing the edges with the parameters established for the tool during design time.   The evaluation is a two step process: first a set of discriminators, and then a set of constraints, are applied to the list of edges to eliminate those edges which do not meet the design requirements.   Discriminators are a set of boolean edge evaluation criteria that eliminate edges that do not possess certain basic attributes, such as Minimum Strength, Edge Polarity and Pair Straddle. Edges that pass the discriminators are then evaluated by a more comprehensive set of edge evaluation criteria, known as constraints.   Constraints are weighted functions that assign a constraint score to various edge attributes, such as Position and Size.   Finally, an edge score is generated as a sum of all the weighted scores for each active constraint.   Should an edge score be higher than the minimum accept level, the edge will be accepted.

Edge Tools Theory: Applying Discriminators

To help minimize processing time, a set of boolean discriminators are applied against the Edges found from the edge strength array.   These discriminators are set up by the designer during design mode as a method of eliminating edges that are too far outside the design criteria to be acceptable.

Discriminators for Calipers:

Pair Straddle	*Minimum Edge Strength*
E1 Polarity	*E2 Polarity*

Discriminators for EdgeLocators:

*Polarity*POLARITY	*Minimum Edge Strength*

The *Minimum Strength* discriminator can be graphically displayed and edited in the Edges Dialog Box. The graphical representation of the minimum strength is made relative to the strength of the edges viewed.   Only edges whose strength is greater than or equal to the minimum strength will be accepted for further analysis, all other edges will be eliminated.   This effect can be graphically demonstrated in the Edge Strength Window by marking only those edges whose peaks in the edge strength curve are higher than the minimum strength line.   The minimum strength discriminator is useful for inspecting a noisy image.   Raising the minimum strength above the largest noise peak will cause the scoring engine to ignore the noise in the image and only use stronger edges for inspection.   The default value for the minimum strength is 20.

Edge Polarity is another type of discriminator that will eliminate edges or edge pairs.   The behavior of this discriminator varies depending on the mode of operation.   In an EdgeLocator tool, the *Polarity* discriminator will throw out edges that did meet the *Minimum Strength* criteria.   In a Caliper tool, the *Polarity* discriminator is applied to each edge in the edge pair (E1 being the first edge) and will throw out any combination of edge pairs where either edge does not meet the polarity discriminators for that edge pair.

Pair Straddle is the third discriminator, only applicable to a Caliper tool.   When this discriminator is enabled, a Caliper tool will throw out a combination of two edges if either the left edge is not to the left of, or the right edge is not to the right of, the midpoint of the ideal position of the edges.   The ideal position of an edge is the location at which an edge is expected to be found.

Edge Tools Theory: Applying the Scoring Engine

Once the Edges have been evaluated by the Discriminators, the remaining edges are evaluated using a Scoring Engine to rank how well each edge matches a set of Constraints.   A constraint is a method establishing by the designer to further evaluate the edges in the field of view.   A Constraint Score is generated for each active constraint applied to each edge.   As in the case of discriminators, some constraints are applied to single edges for both an EdgeLocator and a Caliper tool, while others are applied only on edge pairs in a Caliper tool.   In both cases a comparison is made between the Ideal Value of a constraint and the actual value of an edge or edge pair.   For non ideal edges, a constraint score is generated for each active constraint based on the Edge Evaluation Function (EEF) for that constraint.

Constraints for Calipers:

Pair Position

E1 Position

E2 Position

Pair Size

E1 Strength

E2 Strength

Constraints for EdgeLocators:

Position

Strength

The set of constraints that are applied to edge pairs act to filter out those edge pair combinations that are not valid.   Constraints are activated by selecting the corresponding option button in the Setup Dialog Box.   *Pair Position* behaves in a fashion similar to the *Position* constraint for a single edge, although it operates on two edges rather than one.   In some applications of a Caliper tool, it may be advantageous to score on the position of one or both edges (e.g. measuring the length of an object that is justified to one side), rather than scoring on the position of the pair together.   Because scoring on *Pair Position* or scoring on the *Position* of each edge are mutually exclusive, enabling *Pair Position* will disable the *Position* constraint for edge 1 and edge 2.   Similarly, enabling the *Position* constraint for either edge 1 or edge 2 will disable the *Pair Position* constraint.

The *Pair Size* constraint is useful when searching for an object of a particular known size in the Viewport.   This is quite useful when looking for an object that could vary in position from one image to another, such as a Fiducial on a printed circuit board.   Using the *Pair Size* constraint with the *Pair Position* constraint will search for an object of a particular size that is in some known location in the Viewport.   Both the *Pair Position* and the *Pair Size* constraints will return a constraint score reflecting how close the attribute of the edge pair matched the ideal value.

A Constraint score is generated for each edge based on the shape of the Edge Evaluation Function for each constraint.   The EEF for each constraint is centered around an ideal value, or the value at which an ideal edge (with a perfect constraint score) would exist.   For example, the ideal value for the *Position* constraint describes the expected position of an edge or edge pair, the expected size of an edge pair for the *Size* constraint, or the ideal strength across an edge for the *Strength* constraint.   Should the value of an edge or edge pair constraint be evaluated to match the ideal value, the constraint score for that edge is determined to be perfect and is assigned a perfect score (1.0).   If the actual value of the constraint is not the ideal value, the constraint score is defined by the Edge Evaluation Function.

It is worth noting that the scoring engine has been optimized for one particular combination of constraints/discriminators.   When the only active constraint is *Pair Size* and the only active discriminator is *Edge Polarity* an optimized version fo the engine is invoked.

The scoring engine has the ability to place a greater emphasis on constraints that are considered to be of greater importance by assigning Weights to each of the constraint scores, known as a weighted score. The final step in scoring an edge is to calculate the Total Score for that edge.

By modifying the weights and the Edge Evaluation Functions for each of the active constraints, the scoring engine can be made to select a specific edge/edge pair from a list of candidates. By correctly setting up the weighting parameters, a powerful tool can be developed to perform various edge/edge pair finding applications, including:

i) extracting an edge pair of specific width (e.g. looking for a single pair of edges such as finding a fiducial that could appear anywhere in the Viewport; or looking for multiple pairs of edges across the Viewport such as finding all the leads on an integrated circuit);

ii) extracting an edge pair centered about a particular position (e.g. searching for tooling holes of a specific size on a plate that has other holes of similar size);

iii) extracting an edge pair that starts or ends in a particular position (e.g. looking for, and measuring the width of an object of variable size that always has one side located in a specific position such as measuring the length of bolts);

iv) extracting a count of edges in a Viewport (e.g. counting the number of threads on a bolt by finding all the positive polarity edges in the Viewport);

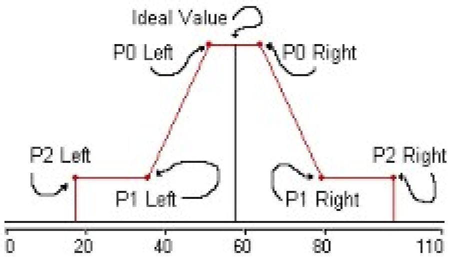
v) extracting the presence of a single edge (e.g. detecting the presence of a component in an assembly by placing an EdgeLocator where the edge of the component is expected to be);

vi) locating a specific edge, (e.g. looking for the fourth edge in a list of edges to determine its location; or locating the side of an object to find its position for further analysis).

Edge Tools Theory: Edge Evaluation Functions

All detected edges that pass the Discriminators are scored with Constraints to find the best edge or edge pair depending on the mode of operation.   Edge constraint scores that are said to be perfect are scored highest, assigned a constraint score of 1.0, otherwise they are given a constraint score less than 1.0, possibly as low as 0 depending on the shape of the Edge Evaluation Function.

There is an EEF associated with each active constraint.   Each EEF consists of an Ideal Value, and six EEF Points.   These define how ***XCaliper*** should score a constraint.



The Edge Evaluation Function defines how rapidly the constraint score assigned to a non-perfect edge falls to zero as the constraint becomes less and less perfect.   If the actual value of a constraint is equal to the ideal value, the Edge Evaluation Function assigns a perfect constraint score of 1.00 to that property.   If the actual value of the property is to the left of P2 Left, or to the right of P2 Right, the constraint score is zero; and if the actual value of the property is located somewhere between the P2 Left and the P2 Right, the constraint score assigned is the 'Y' value of the Edge Evaluation Function curve at the position on the curve of the input value.

The shape of the EEF can be altered using the Edge Evaluation Function Editor in the Edges Dialog Box.   To move any of the points simply click and drag the point to its new location.   The points that define the regions of the EEF are restricted in their movement around the curve: P0 can only move in the horizontal direction between the ideal value and P1; and P1 must be closer to the ideal value than P2, although it is possible for P1 and P2 to overlap.

Edge Tools Theory: Weighting and Scoring

XCaliper enables the user to place a greater emphasis on Constraints that are considered to be of greater importance by assigning weights to each of the Constraint Scores, known as the weighted score.   Weights are only applied to active constraints, where the sum of all weights is defined to be always 1.00, or 100%.   All weights in the default configuration are set to be equal (e.g. if two constraints are selected each is evaluated at 50%, if three constraints are selected each is evaluated at 33%, etc.) Assigning weights to a constraint is done in the Setup Dialog Box for EdgeLocators, and in the Weights Dialog Box for Caliper tools.

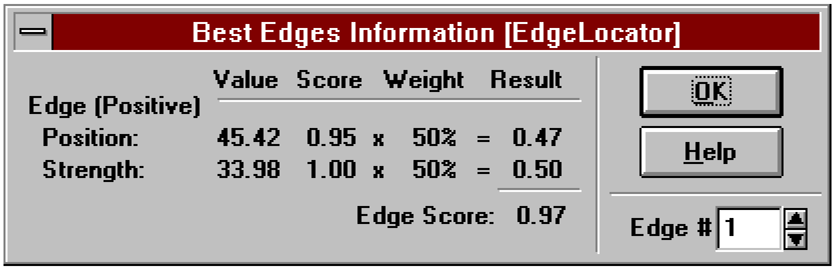
Edge Tools Theory: Total Score of an Edge

The total edge score is defined as the sum of all Weighted Scores of active Constraints for that edge. Each weighted score is determined by multiplying the Constraint Score obtained from the Edge Evaluation Function, by the weight set for that constraint.   The edge or edge pair that has the best total edge score is the best edge or edge pair found.   This total edge score is available to the Visual Basic application in the Score property, and a list of edges returned in a linked list based on their score is available under program control.   For visual feedback, the scoring breakdown for all edges or edge pairs can be seen by invoking the Edge/Edge Pair Information Report Box.   This is done by clicking on one of the edge/edge pair scores displayed in the Edges Dialog Box.

Edge Tools Setup: Edge Information Report Box

The Edge Information Report Box is invoked by clicking the numbers that indicate the Score of an edge near the top of the Edges Dialog Box.   When the cursor is moved over a score, it will change to a hand indicating that the mouse can be clicked to invoke the Edge Information Report box.   This dialog box displays information about each edge found and a scoring breakdown that indicates how the score for the edge was obtained.

Click on the dialog box for help on a specific item.



Best Edges Information [EdgeLocator]

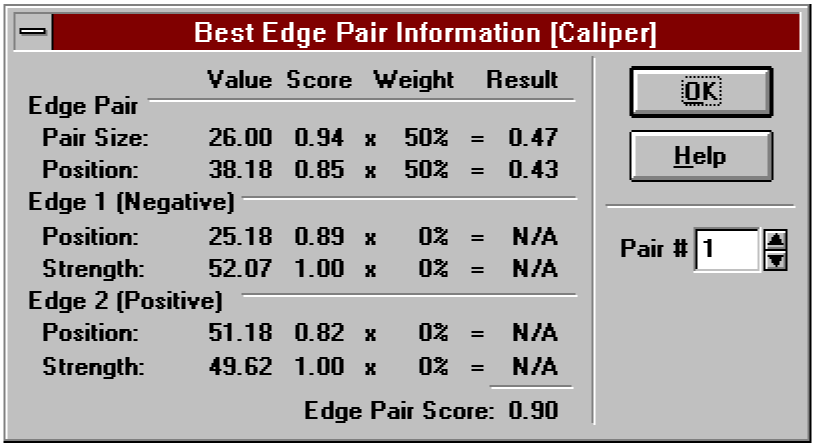
Edge (Positive)	Value	Score		Weight		Result
Position:	45.42	0.95	x	50%	=	0.47
Strength:	33.98	1.00	x	50%	=	0.50
					Edge Score:	0.97

Edge # 1

Edge Tools Setup: Edge Pair Information Report Box

The Edge Information Report Box is invoked by clicking the numbers that indicate the Score of an edge pair near the top of the Edges Dialog Box. When the cursor is moved over a score it will change to a hand indicating that the mouse can be clicked to invoke the Edge Pair Information Report box. This dialog box displays information about each edge pair found and a scoring breakdown that indicates how the score for the edge pair was obtained.

Click on the dialog box for help on a specific item.



Best Edge Pair Information [Caliper]

	Value	Score		Weight		Result
Edge Pair						
Pair Size:	26.00	0.94	x	50%	=	0.47
Position:	38.18	0.85	x	50%	=	0.43
Edge 1 (Negative)						
Position:	25.18	0.89	x	0%	=	N/A
Strength:	52.07	1.00	x	0%	=	N/A
Edge 2 (Positive)						
Position:	51.18	0.82	x	0%	=	N/A
Strength:	49.62	1.00	x	0%	=	N/A

Edge Pair Score: 0.90

Pair # 1

Edge Tools Setup: Polarity Group

This group chooses what Polarity an edge should be to be accepted for further evaluation.

Checking Don't Care will causes both positive (dark-to-light) and negative (light-to-dark) polarity edges to be accepted.

Checking Positive will causes only positive polarity edges to be accepted.

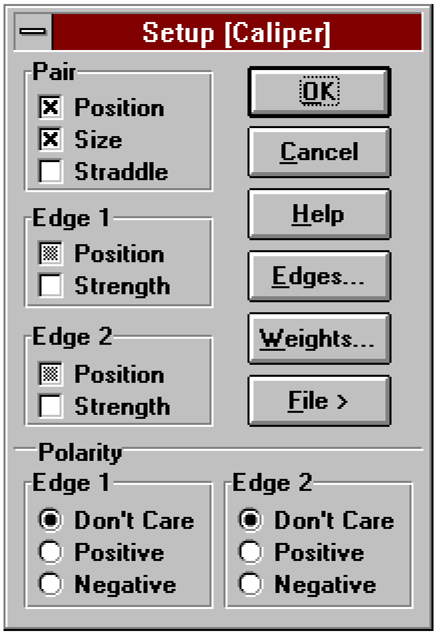
Checking Negative causes only negative polarity edges to be accepted.

Edge Tools Setup: Caliper Setup Dialog Box

The Setup Dialog Box is invoked in one of two ways: either at design time when the Setup property from the Properties window of a Caliper tool is double clicked, or at run time when the mouse is double clicked anywhere inside an edge tools Viewport. This dialog box allows the various Constraints to be enabled or disabled and launches the Edges Dialog Box and the Weights Dialog Box.

When a Caliper is initially created, Pair Position and Pair Size are selected by default, and both have a Weight of 50%. Selecting any more constraints will result in an equal distribution of weights between the selected constraints so the weights sum to 100%.

Click on the dialog box for help on a specific item.



Setup [Caliper]

Pair
- ☒ Position
- ☒ Size
- □ Straddle

Edge 1
- ▨ Position
- □ Strength

Edge 2
- ▨ Position
- □ Strength

Polarity

Edge 1
- ● Don't Care
- ○ Positive
- ○ Negative

Edge 2
- ● Don't Care
- ○ Positive
- ○ Negative

Edge Tools Setup: Pair Position Check Box

Enabling this box selects the Pair Position as a Constraint when evaluating edge pairs.

When this Position check box is enabled, the Position check boxes in the Edge 1 and Edge 2 groups become disabled since scoring on the Pair Position and the individual edge positions are mutually exclusive.

Edge Tools Setup: Pair Size Check Box

Enabling this box selects the Pair Size as a Constraint when evaluating edge pairs.   Similarly when this box is disabled, the Pair Size is not used in the evaluation of edge pairs.

Edge Tools Setup: Pair Straddle Check Box

Enabling this box causes the Caliper to search for edge pairs whose Edge 1 and Edge 2 are on opposite sides of the Pair Position.   Similarly when this box is disabled Pair Straddle is not used in the evaluation of edge pairs.

Pair Straddle is a Boolean Discriminator, hence there is no Weight associated with it.

Edge Tools Setup: Edge 1 Position Check Box

Enabling this box selects the Edge 1 Position as a Constraint when evaluating edge pairs.

When the Edge 1 Position check box is enabled, the Position check box in the Pair group becomes disabled since scoring on Edge 1 Position and Pair Position are mutually exclusive.

Edge Tools Setup: Edge 1 Strength Check Box

Enabling this box selects the Edge 1 Strength as a Constraint when evaluating edge pairs.

Edge Tools Setup: Edge 2 Position Check Box

Enabling this box selects the Edge 2 Position as a Constraint when evaluating edge pairs.   Similarly when this box is disabled, Edge 2 Position is not used in the evaluation of edge pairs.

When this Position check box is enabled, the Position check box in the Pair group becomes disabled since scoring on Edge 2 Position and Pair Position are mutually exclusive.

Edge Tools Setup: Edge 2 Strength Check Box

Enabling this box selects the Edge 2 Strength as a Constraint when evaluating edge pairs. Similarly when this box is disabled, Edge 2 Strength is not used in the evaluation of edge pairs.

Edge Tools Setup: Position Check Box

Enabling this box selects the Edge Position as a Constraint when evaluating edges.   Similarly when this box is disabled, Edge Position is not used in the evaluation of edges.

Edge Tools Setup: Strength Check Box

Enabling this box selects the Edge Strength as a Constraint when evaluating edges.   Similarly when this box is disabled, Edge Strength is not used in the evaluation of edges.

Polarity: Don't Care Button

The Don't Care button will cause the edge tool to allow an edge of any Polarity to be accepted.

Polarity: Positive Button

The Positive button will cause the edge tool to allow only an edge of positive (dark-to-light) Polarity to be accepted.

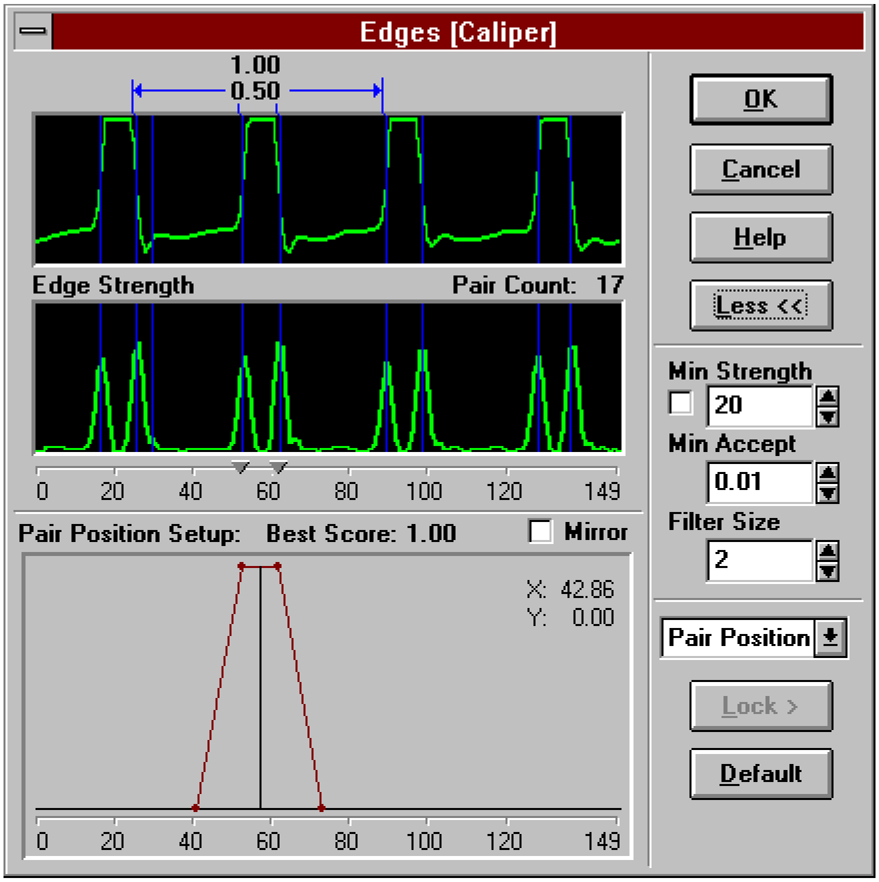
Polarity: Negative Button

The Negative button will cause the edge tool to allow only an edge of negative (light-to-dark) Polarity to be accepted.

Edge Tools Setup: Edges Dialog Box

The Edges Dialog Box is invoked when the **Edges...** button is clicked in the Setup dialog box. This dialog box allows the Edge Evaluation Functions for the various Constraints to be set up and altered. The dialog box will also allow several other parameters to be altered such as the Minimum Strength, the Minimum Accept Threshold, and the Filter Size.

Click on the dialog box for help on a specific item:



Edges [Caliper]

Edge Strength   Pair Count: 17

Pair Position Setup: Best Score: 1.00   □ Mirror

X: 42.86
Y: 0.00

Min Strength
□ 20

Min Accept
0.01

Filter Size
2

Pair Position

Edge Tools Setup: Profile View

The Profile View is a group of items in the Edges Dialog Box that include the following items:

Projection Window
Edge Strength Window
Best Edge Pairs Indicators
Expected Position Markers

Edge Tools Setup: Filter Size Edit Box

The Filter Size Edit Box allows the Filter Size to be edited.   To change the filter size, type the new filter size into the box, or use the spin buttons to change the value by unit increments.   The entry is parsed so as to accept only valid numbers, between 0-255.

Edge Tools Setup: Edge Evaluation Function Editor

The Edge Evaluation Function Editor occupies the bottom of the Edges Dialog Box, and consists of a title bar and the editor.

The title bar contains   the name of the function that is in the editor area, the Constraint Score of the best edge pair found, and the Mirror Checkbox.   The title that represents the function in the editor area can be changed by using the Edge Evaluation Function Selector.

The editor itself is a large window containing a representation of an Edge Evaluation Function.   The editor allows all six EEF Points and the Expected Position to be moved by clicking and dragging the item to be moved to the desired location.   Whenever the mouse pointer is on top of a movable object, the mouse pointer will change to arrows representing the directions the object is allowed to move.   Should the object being moved be attempted to move outside its valid range, the object will stop moving at the last valid place in its range.

Should a EEF Point be moved off the screen by moving the Expected Position, the Point can be brought back on to the screen at its original position relative to the Expected Position by moving the Expected Position back far enough so that the EEF Point appears on the screen once again.

While the mouse pointer moves across the editing area, the current position of the mouse pointer is displayed in the upper left or right corner of the editing area.   The X value is the position of the cursor with respect to the left of the Viewport, and the Y value would be the score obtained for the current height of the cursor in the editing area.

A ruler is located at the bottom of the editing area which shows the width of the editing area with respect to the Viewport width.

Edge Tools Setup: Mirror Checkbox

The Mirror Checkbox makes editing symmetrical Edge Evaluation Functions easier. When the Mirror Checkbox is enabled, all edits made in the Edge Evaluation Function Editor will be mirrored about the Expected Position.

In order to specify which side of the Edge Evaluation Function mirrors the other, when Mirror is first enabled the function in the Edge Evaluation Function Editor remains as it was. However when the first EEF Point is clicked, all EEF Points on the opposite side of the Expected Position are set to mirror the EEF Points on the side just clicked.

Edge Tools Setup: Edge Evaluation Function Selector

The Edge Evaluation Function Selector is a standard Windows drop down combo box that selects one of the Edge Evaluation Functions for the active Constraints into the Edge Evaluation Function Editor.

Only the currently active constraints will be displayed in the Edge Evaluation Function Selector. Constraints can be activated or de-activated from within the Setup dialog box or the Weights Dialog Box.

Edge Tools Setup: Lock> Button

The Lock> Button will invoke the Function Lock group which contains two check boxes: Position and Strength. Both check boxes perform the same function except Position operates on the E1 Position and E2 Position EEFs, and Strength operates on the E1 Strength and E2 Strength Edge Evaluation Functions (EEFs).

Enabling one of these Function Lock buttons will force the E1 and E2 functions of Position or Strength to have identical EEFs about their Expected Positions. This means that all edits on one of the E1 or E2 EEFs will cause the same changes to be made to its counterpart for the opposite edge.

When either of these two buttons are enabled, a dialog box will open up to prompt which function will initial mirror the other. Clicking the E1 = E2 button will cause E1 to initially mirror the E2 function, and vice-versa for the E2 = E1 button.

Enabling Function Lock will not make the Expected Positions equal, but it will make the EEF about the Expected Positions equal.

Edge Tools Setup: Projection Window

The Projection Window is the upper window in the Edges Dialog Box that displays a representative gray scale picture of the Viewport overlaid by a graph of the Projection of the Viewport. All edges or edge pairs that pass the imposed constraints will be represented by vertical lines. Above the lines, the two best Edge Scores will be displayed.

Edge Tools Setup: Edge Strength Window

The Edge Strength Window is the lower window in the Edges Dialog Box that displays a gray scale representation of the Projection of the Viewport. The picture is made up of vertical stripes where the brightness of each stripe represents the average light intensity of each column in the Viewport. Superimposed over the representation of the projection and scaled for visibility is a graph of the Edge Strength. In addition, the location of the edges found in the Viewport are represented by vertical lines.

When the Check Box in the Minimum Strength Edit Box is selected, a horizontal line across the Edge Strength Window will appear. Clicking and Dragging this line will have the effect of increasing or decreasing the value in the Minimum Strength Edit Box.

Edge Tools Setup: Expected Position Markers

The Expected Position Marker resides on the scale that appears below the Edge Strength Window and can be identified as a triangular object that points to a value on the scale.   For a Caliper tool, there are usually two Expected Position Markers, the leftmost for Edge 1 and the rightmost for Edge 2.   For an EdgeLocator there is only one Expected Position Marker as only single edges are to be found.

An Expected Position Marker identifies the ideal location of an edge.   This information, in combination with the Edge Evaluation Functions, can be used to score edges/edge pairs as they appear in the Viewport.

Clicking and dragging the Edge 1 Marker will update the Expected Position for Position E1.   Similarly, clicking and dragging the Edge 2 Marker will update the Expected Position for Position E2.   If Pair Size or any one of the Position EEFs are displayed in the EEF Editor, any changes made to the Expected Positions using these Markers will automatically cause these functions to update based on the new Edge 1 and Edge 2 Expected Positions.

When an Expected Position Marker is being dragged, a vertical line representing the Expected Position appears in the Projection Window and the Edge Strength Window and moves along with the Expected Position Marker.

Edge Tools Setup: Minimum Strength Edit Box

The Minimum Strength Edit Box allows the Minimum Strength to be edited.   To change the minimum strength, type the new minimum strength into the box, or use the spin buttons to change the value by unit increments.

The valid range for Minimum Strength is from 0 to 255.

To the left of the Minimum Strength Edit Box is a check box.   Enabling this check box will cause a horizontal line to appear in the Edge Strength Window.   The horizontal line represents the value of the Minimum Strength and how it compares to the Edge Strength.   Clicking and Dragging this horizontal line will increase or decrease the value of the Minimum Strength.

If the check box is clicked and does not become enabled, then the value of the Minimum Strength currently set in the edit box is too large to appear in the Edge Strength Window.   If this happens, reduce the value of the Minimum Strength and try again.

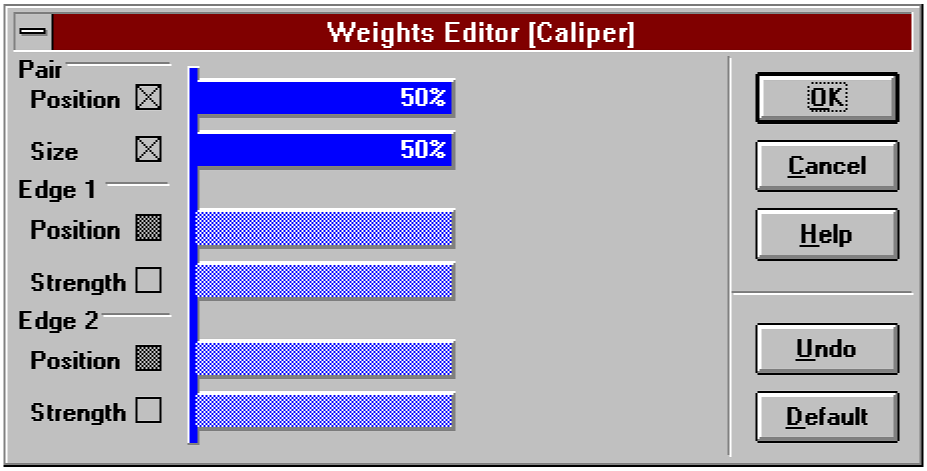
Edge Tools Setup: Minimum Accept Threshold Edit Box

The Minimum Accept Threshold Edit Box allows the Minimum Accept Threshold to be edited. To change the minimum accept threshold, enter a new threshold between 0-255 into the box, or use the spin buttons to change the value by unit increments.

Edge Tools Setup: Weights Dialog Box

The Weights Dialog Box is invoked when the **Weights...** button is clicked in the Setup dialog box.   This dialog box allows the relative Weights of the various Constraints to be altered so as to place greater emphasis on some contraints at the expense of others.   When a Caliper is first created the weights for all constraints are equal so that every constraint that is active has a weight equal to all of the other active constraints.   This is also the Default condition.

Click on the dialog box for help on a specific item.



Pair
Position ☒ 50%
Size ☒ 50%
Edge 1
Position ▨
Strength □
Edge 2
Position ▨
Strength □

Edge Tools Setup: Weight Distribution Bar

The Weight Distribution Bar shows the relative Weight of each of the active Constraints used in an edge tool.   Each active Weight Distribution Bar has a value between 0-100% and can be altered to suit the application.   The total of all weights will always sum to 100%.

To change the weight of a constraint, place the mouse pointer over the end of the bar.   When the cursor changes to a set of horizontal arrows, press the left mouse button and drag the bar to the left or right until the percentage display in the bar is at the value that is desired.   Release the left mouse button for the change to take effect.

A constraints weighting can be locked at a set percentage so that changes made to the weights of other constraints do not affect it.   To lock a weighting, click on the center of the bar.   The bar then will appear to have a chain wrapped around it, representing that its weight is locked.   To unlock the weight, click the bar again and the chain will disappear.

Undo Button

This button will undo all the changes that have been made in the dialog box since the dialog box was opened.

Edge Tools Setup: Default Button

This button will change the state of the associated control to a suitable pre-defined default value that works well in most situations.

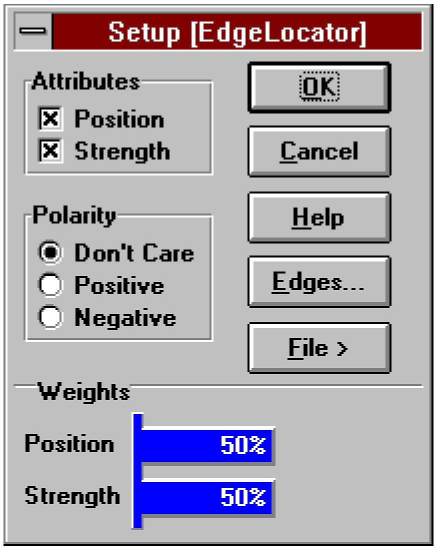
If this button is clicked within the Edges Dialog Box the associated feature that is returned to its default value is the Edge Evaluation Function Editor.   The default value in this case is a curve that would work in most situations.

If this button is clicked within the Weights Dialog Box, the associated feature that is returned to its default value is the Weights Editor.   The default value in this case is to set all constraint Weights to equal values so that a constraint will have an equal weight with other active constraints when it becomes enabled.

Edge Tools Setup: EdgeLocator Setup Dialog Box

The Setup Dialog Box is invoked in one of two ways: either at design time when the Setup property from the Properties window of an edge tool is double clicked, or at run time when the mouse is double clicked anywhere inside an edge tools Viewport.   This dialog box allows the various Constraints to be enabled or disabled and launches the Edges Dialog BoxWhen an **EdgeLocator** is initially created, Position and Strength are selected by default, and both have a Weight of 50%.   In addition, the Polarity of an edge is set to Dont Care.

Click on the dialog box for help on a specific item.



Setup [EdgeLocator]

Attributes
☒ Position
☒ Strength

Polarity
● Don't Care
○ Positive
○ Negative

Weights
Position 50%
Strength 50%

Edge Tools Setup: Best Edge/Edge Pair Markers

Best Edge/Edge Pair markers are used to identify the best two edges or edge pairs found using the current set up.

For Caliper tools, the Best Edge Pair Markers are visible above the Projection Window as a set of distance markers pointing to Edge 1 and Edge 2 of an Edge Pair.   There are up to two Best Edge Pair Markers, the highest one in the dialog box shows the Best Edge Pair, and the one on the second row represents the Second Best Edge Pair.   The Score of the Edge Pair is shown in the center of the Edge Pair Markers.   Clicking the Score will open the Edge Pair Information Report Box which will illustrate how the score for that edge pair was obtained.

For EdgeLocator tools, the Best Edge Markers are visible above the Projection Window as single markers pointing to the best two edges.   There are up to two Best Edge Markers, the highest one being the Best Edge.   The Score of the edge is displayed directly above the marker.   Clicking the Score will open the Edge Pair Information Report Box which will illustrate how the score for that edge was obtained.

As other components in the Edges Dialog Box are changed, the Edge Markers may move and change to reflect the new best edges/edge pairs that were found as a result of the changes made.

File> Button

Clicking the **File>** Button provides access to the **File** Menu from where the complete configuration of the tool can be saved or loaded.   The **File** Menu that is invoked contains four options:

New This restores the state of the entire tool to the default values.   This option can be selected when a new configuration file is to be created.

Open... This allows an existing configuration file to be loaded.

Save This saves the current configuration using the active configuration file name (which is set whenever an *Open...* or *Save As...* option is used)

Save As... This saves the current configuration using a new filename.

OK Button

The **OK** Button will exit the current dialog box and will keep any changes that were made.

Help Button

The **Help** Button invokes the XCaliper help on a specific topic.

Cancel Button

The **Cancel** Button will exit the current dialog box and will undo any changes that have been made in the dialog box while it was open.

Weights... Button

The **Weights...** Button will open the Weights Dialog Box.

Edges... Button

The **Edges...** Button will open the Edges Dialog Box.

Edge/Edge Pair Selection Box

Use this drop down selection box to change which edge/edge pair is displayed in the Information Box. Selecting a new edge will automatically update the contents of the dialog box.

More >> / Less << Button

The **More >>** / **Less <<** button will expand and shrink the Edges Dialog Box.

ImageDevice Tool

See Also Properties Commands

Description

The **ImageDevice** control manages the Frame Grabber. The control manages all aspects of the image acquisition sub-system such as gain and offset controls, input channel selection, sync settings and the position of the acquired image in the Frame Buffer. It also manages the display of the grabbed image on the underlay or secondary monitor and provides services that allow the analysis and processing tools access to the acquired image.

It also provides support for display of the image on the Windows monitor, either in "pass-through" mode or as a standard Windows bitmap. It does this through its implicit or bundled **ImageHook**. This implicit hook behaves identically to an actual **ImageHook** except that an **ImageDevice**s hook can only attach to the devices image.

File Name

XVBXIMG.VBX

Remarks

During design time the control is represented by an icon depicting an image inside a monitor. The control is invisible during run time except for its possible effect on the background of the Container.

See Also

ImageHook

MemoryBuffer

Properties

All of the properties are listed in the following table. Properties that are not standard or that require special consideration when used with this control are marked with an asterisk (*). For information on standard Visual Basic properties, please see the Visual Basic Programmer's Guide or the Visual Basic on-line Help.

About*
ActiveControl*
BufferSizeX*
BufferSizeY*
Command*
ContGrabType*
DeviceFile*
ElapsedTime*
Genlock*
GrabMode*
Height
Index
InputChannel*
InputGain*
InputLut*
InputOffset*
InputOriginX*
InputOriginY*
InputPan*
InputScroll*
InputSizeX*
InputSizeY*
InputZoomX*
InputZoomY*
Left
Name
OutputLut*
OutputOriginX*
OutputOriginY*
OutputPan*
OutputScroll*
OutputSizeX*
OutputSizeY*
OutputZoomX*
OutputZoomY*
ScaleType*
ScaleX*
ScaleY*
Setup*
SyncMode*
Tag*
Top
Transparent*
UpdateMode*
Width

Commands

All of the commands are listed in the following table. Commands can be executed either during design time or run time. To execute a command during design time, enter the command into Command property of an **ImageDevice**.

Clear

Close

Convolve

Copy

Dilate

Erode

LoadFile

Median

Open

SaveFile

TuneInput

ImageHook Tool

See Also Properties Commands

Description

The **ImageHook** control manages the display of an **ImageDevice** or **MemoryBuffer** on the Windows screen. It is responsible for performing whatever operations are necessary to make a copy of the image appear on the system monitor. It also supplies support mechanisms and mappings which allow image tools to access the **ImageDevice**. The **ImageDevice** and **MemoryBuffer** have built-in image hooks for managing the display, but for advanced display methods, such as viewing different parts of the same frame buffer, a separate **ImageHook** control (or controls) is required.

File Name

XVBXIMG.VBX

Remarks

An **ImageHook** works by taking over the background display of the Container in which it is placed. The background properties of the container are then ignored. Instead of drawing the color or Picture specified by these properties, the image buffer is displayed instead. All of the other properties of the container work normally, therefore other controls such as buttons may be placed inside the container. (This will make it look as though the button is on top of the frame buffer which is what is normally wanted). Overlay lines and text using graphic controls or draw methods can also be used.

The **ImageHook** control is represented in the Visual Basic toolbox by an icon depicting an image on a fish hook. At design time, this icon is displayed on the container. At run time, it is invisible except for its effect on the container's background.

There are two basic types of **ImageHooks**: transparent and non-transparent. There are two types of transparent modes: Locked Window mode and Floating Window mode.

A transparent **ImageHook** is a window which looks through to the underlying Frame Buffer. This type of hook assumes that the frame buffer and Windows are displayed on a single monitor and that the frame grabber supports switching from one to the other on a pixel-by-pixel basis. In the Floating Window case the *OutputPan, OutputScroll, OutputOriginX* and *OutputOriginY* properties of the **ImageDevice** control the alignment between the VGA screen and the image buffer. Thus, changing the location of the container changes the part of the frame buffer being displayed. On the other hand, in the Locked Window case, these same properties control the alignment between the container and the image buffer. In this mode, changing the location of the container does not result in any change in the display image.

Note that, when employing an actual **ImageHook** control in a transparent mode, any modifications to its output properties are passed on to its attached device as it is the device which controls the underlay.

Non-transparent hooks work like the *Picture* property of the enclosing container. That is, if you move the container, the image viewed moves with it. Non-transparent hooks are the only type of **ImageHook** which will work with devices that do not support hardware overlay. Your frame buffer card may or may not support overlay; **MemoryBuffers** will not.

Non-transparent hooking, however, gives more control over the relationship between the Frame Store and the image. It is possible to specify that the image being displayed be automatically updated. The **ImageHook** supplies zoom factors and the position in the frame buffer to display.

Most importantly, the **ImageHook** specifies a spatial relationship, or mapping, between the frame buffer and image tools placed in the container. A Caliper placed inside a hooked container will correctly access the frame buffer using this relationship. In contrast, if the Caliper is placed on an unhooked container, it uses a default mapping strictly to allow the tool to function without crashing the system. The use of tools that are placed on unhooked containers is strongly discouraged as the result is undefined.

See Also

ImageDevice

MemoryBuffer

Properties

All of the properties are listed in the following table. Properties that are not standard or that require special consideration when used with this control are marked with an asterisk (*). For information on standard Visual Basic properties, please see the Visual Basic Programmer's Guide or the Visual Basic on-line Help.

About*
ActiveControl
Command*
DeviceName*
ElapsedTime*
Height
Index
Left
Name
OutputPan*
OutputScroll*
OutputZoomX*
OutputZoomY*
Setup*
Tag*
Top
Transparent*
UpdateMode*
Width

Commands

All of the commands are listed in the following table. Commands can be executed either during design time or run time. To execute a command during design time, enter the command into Command property of an **ImageHook**.

Clear
Close
Convolve
Copy
Dilate
Erode
LoadFile
Median
Open
SaveFile
TuneInput

MemoryBuffer Tool

See Also Properties Commands

Description

A **MemoryBuffer** is a Frame Buffer allocated out of system memory. **MemoryBuffers** can be used to store images in system memory and process them in a manner similar to those associated with **ImageDevices**. They may be displayed on the Windows screen using non-transparent setting of the *Transparent*.

File Name

XVBXIMG.VBX

Remarks

The **MemoryBuffer** control is represented in the Visual Basic Toolbox by an icon depicting an image sitting on an integrated circuit. At design time, this icon is displayed on the Container. At run time, it is invisible except for its effect on the containers background.

There may be as many **MemoryBuffers** as desired, limited only by the amount of memory available in the system. The size of the buffer is specified by the *BufferSizeX* and *BufferSizeY* properties. Changing the size of a buffer will result in the loss of its contents.

In order to add a **MemoryBuffer** to an application, simply place the icon in a Visual Basic container. If you wish to have an image in the buffer on initialization, set the *Command* property to load the image. It will automatically be loaded when the buffer comes up.

See Also

ImageDevice

ImageHook

Properties

All of the properties are listed in the following table. Properties that are not standard or that require special consideration when used with this control are marked with an asterisk (*). For information on standard Visual Basic properties, please see the Visual Basic Programmer's Guide or the Visual Basic Help screens.

About*
ActiveControl*
AutoSize*
BufferSizeX*
BufferSizeY*
Command*
ElapsedTime*
Height
Index
Left
Name
OutputLut*
OutputPan*
OutputScroll*
OutputZoomX*
OutputZoomY*
ScaleType*
ScaleX*
ScaleY*
Setup*
Tag*
Top
Transparent*
UpdateMode*
Width

Commands

All of the commands are listed in the following table. Commands can be executed either during design time or run time. To execute a command during design time, enter the command into Command property of the **MemoryBuffer**.

Clear

Close

Convolve

Copy

Dilate

Erode

LoadFile

Median

Open

SaveFile

Image Tools Theory: Overview

XCaliper includes three controls, implemented as Custom Controls, which interact with the Frame Buffer and the display: the ImageDevice which manages interaction with a Frame Grabber, the MemoryBuffer which stores an image into system memory, and the ImageHook which optionally manages the interface between one of the first two controls and the monitor.   The **ImageDevice** and **MemoryBuffer** have built in display capabilities which make the use of an **ImageHook** unnecessary when displaying only one image from one image source (i.e.   an **ImageDevice** or **MemoryBuffer**). The **ImageHook** provides the developer with the capability to separate image display from image storage.   There are several reasons why this may be necessary.   Most importantly, it makes it possible to display different images in the same location on the screen, or the same image at different locations on the screen.   At run-time the tools themselves are invisible although their effects, namely the display of an image on the screen, may be visible.

The **XCaliper Image Tools Theory** on-line documentation provides a discussion on the following topics:

i) Accessing Images
ii) Creating a Frame Store
iii) Grabbing an Image
iv) Displaying an Image
v) MemoryBuffers
vi) Device Drivers Restrictions

Image Tools Theory: Accessing Images

XCaliper is a set of image processing and analysis tools which are designed to run in single monitor mode.   That is, unlike some other packages, the user interface and the images are kept together on a single screen.   It would, in fact, be difficult to imagine a dual monitor approach which would work in the Visual Basic environment.   Visual Basic allows the user to add a series of building blocks together on one or more **Forms** to create an application.

Since ***XCaliper*** only uses one monitor, it must incorporate a mechanism for mixing graphics and images together on a single screen.   In fact two mechanisms exist: the image may be copied on the VGA screen or a pass-through mechanism may be put in place that allows the VGA buffer and the image frame buffer to be mixed on a single monitor on a pixel-by-pixel basis.   This second technique requires special hardware to perform the mixing.   In either case, the ***XCaliper*** image controls must be placed within Visual Basic container controls which have a *Picture* property.   Applications involving dual monitors require special considerations.

Because image storage and image display are distinct operations, ***XCaliper*** has two types of image controls: the Frame Stores (the ImageDevice and MemoryBuffer) and the ImageHook.   The **ImageDevice** provides access to a frame grabber while the **MemoryBuffer** exists to allow storage of images in high speed system memory.   In fact, a **MemoryBuffer** could be viewed as a special kind of **ImageDevice**, which stores images in system memory, and is not capable of digitization, nor does it support pass-through display.   Both of these controls will also display their image within the container in which they reside.   The **ImageHook** is strictly an image display device and becomes necessary when the developer wishes to control image storage and display independently   for example, when attempting to display the same image in multiple locations on the screen, or perhaps using the same container to display images from different frame stores.   All three of these controls appear as icons on the VGA screen and are only visible at design time.

In summary, if you wish to see an image on the display, a frame store control is all that is necessary to hold the image and display it in the chosen container.   The frame store could be either an **ImageDevice** or a **MemoryBuffer** depending on the requirements of the application.   If, on the other hand the goal is to display different portions of an image, perhaps at different zoom settings, each in its own container, then a frame store and multiple **ImageHooks** must be used.   The frame store is placed in one container and an **ImageHook** is placed in the remaining containers in order to tie, or hook, each of the remaining containers display to the frame store.   By including multiple frame stores in the application, containers can be made to display different images with various settings simply by setting and resetting certain **ImageHook** properties.

Image Tools Theory: Creating a Frame Store

Frame grabber interfaces are extensions of ***XCaliper*** kept in DLLs where each supported device has its own DLL. To use a frame grabber the appropriate DLL must be loaded into your project. By convention, these DLLs have names of the form: XDRV????.DLL where the question marks are some combination of letters which describe the frame grabber. For example, the NULL device is called XDRVNULL.DLL. To gain access to a device driver, simply place an ImageDevice control on some **Form** in your project and change its *DeviceFile* property to that of the file name of the DLL. ***XCaliper*** assumes that if the name is four letters long or less, then the device name is being specified, not the filename. Thus, if you write NULL into the *DeviceFile* property, ***XCaliper*** will take this to mean xdrvnull.dll.

To add a MemoryBuffer to your project, simply place the control on some container it should immediately go to black. Unlike an ImageDevice, before actually using the buffer, you have to allocate some image memory. This can be done in two ways: either the *AutoSize* property can be set to True and perform some operation which will copy an image into it (the buffer will be re-sized to fit), or set the *AutoSize* property to False and modify the *BufferSizeX* and *BufferSizeY* properties as desired. If the AutoSize property is set to False, the size of the buffer will be fixed and images that are too large to fit into the buffer will be clipped.

Image Tools Theory: Displaying an Image

Adding a Frame Store to an application will typically provide all the display capability that is required by the application.   The ImageHook is used when the developer requires more flexibility and control over the display of images.   The **ImageHook** provides a mapping between a frame store and a Visual Basic container   that is, it *hooks* the container to a frame stores image.   It does this by taking over the redraw feature and associated properties of the container control and displaying an image in it.   In fact, the display capabilities of a frame store control are provided by a bundled **ImageHook** that maps, or hooks, the controls image to its container.   The use of an **ImageHook** control, on the other hand, permits the developer to select which existing frame store to display simply by manipulating the **ImageHook**s property values.

It should be noted at this point that a container can only be hooked to one frame store at any one time (i.e.   it can only display one image at a time), either through a frame stores own hooking capabilities or through an actual **ImageHook** control.   Which control is actually hooked to the container is determined by the controls *Transparent* property.   This property has four enumerated values   three that select a different kind of hooked state, while the fourth unhooks the control from the container.   ***XCaliper*** permits only one image control per container to be in a hooked state, the remainder are set to the unhooked state.   Therefore, the only way to change which control is hooked to a given container is to first unhook the currently hooked control.

Hooking a container to an **ImageHook** is a three-step operation.   First the **ImageHook** must be pasted into the control in question.   At this point, if there is not already a hooked control in the container, it will take over the display of the container contents and the background should change in some way.   Next the frame store which is to be mapped into the container needs to be selected, or *attached*.   This is done by setting the *DeviceName* property of the hook to be the same as the *Tag* property of the frame store.   Since this is the specification technique, the *Tag* property should always be unique across all frame stores.   Finally, the **ImageHook**s *Transparent* property should be set appropriately.   Note that if the container already has a hooked control, any attempt to change the new **ImageHook** to a hooked state will generate a Visual Basic error.

XCaliper tries to facilitate this process as much as possible, which is accomplished in several ways. When a frame store control is created, the *Tag* property is given a default value (unlike other controls for which the default value is the empty string). This value will be copied from the *Name* property so that it will initially be MemBuf1 or ImgDev3.   There are two things to remember at this point: first the *Name* property is not necessarily unique.   Controls in control arrays or in different forms may have the same name.   Hence there is no guarantee that the initial *Tag* property value will be unique.   The second thing to remember is that it is the value of the *Tag* property that is relevant, not that of the *Name* property.

When the **ImageHook** is created, it will attempt to find a frame store to which it should attach.   It will try to find a frame store in the container, or failing that, in the **Form**.   If it succeeds it will set its *DeviceName* property appropriately.   And similarly, when a frame store is created, any existing unattached **ImageHooks** will attempt to try to attach to the new frame store.

The final step which may be required when setting up the hook is choosing between the various hooked settings for the *Transparent* property.   Once again, ***XCaliper*** tries to help.   Each frame store has its own default Transparent mode (either Non-Transparent, or Floating Window Transparent) that is used when the hook is first attached, although this can be changed if the frame store will support more than one type of display mode.

Image Tools Theory: Pass-Through vs. VGA-only Display Modes

In one of the two Pass-Through, or Transparent, modes, the Frame Store is displayed as if it is sitting just behind the VGA screen. A Transparent **ImageHook** or frame store opens up a window in the VGA display that allows the user to see through to the image buffer. The first type of pass-through mode, called Floating Window mode, causes the container to behave like a floating window looking through to the stationary image behind. When the window on the VGA display is moved around, a different portion of the image buffer becomes visible. The second type, called Locked Window mode, causes the container to always display the same portion of the image buffer regardless of its location on the screen that is, the underlay moves with the container.

In VGA-only, or Non-transparent mode, by contrast, the image is copied into the VGA buffer for display. Therefore, as in the Locked Window mode, when the window is moved around, the portion of the frame buffer viewed remains the same. VGA-only mode is slower because of the time required to transfer the image to VGA memory (as such, a live image display is not possible with ISA bus frame grabbers. However it is the only mode which supports MemoryBuffers.

Image Tools Theory: Pass-Through Mode Display

In pass-through mode, the process is somewhat more complicated because the ImageDevice and the hook (either bundled with the frame store or used explicitly) interact in order to decide what will be displayed.   In Pass-Through mode, the **ImageDevice**s output properties control the Underlay (as opposed to controlling the Overlay as they do in VGA-only mode).   The result is that they have the same visual effect on the image that they do in VGA-only mode.   This is also true when an explicit ImageHook is used to display the image   that is, the hooks output properties values are passed onto the device and so affect the underlay as well.   This is because the device is responsible for all operations taking place in the underlay, while the **ImageHook** is responsible for all operations in the overlay.   Therefore what is seen on the screen will depend on the output properties of the device (set either directly or through an attached hook) as well as the exact transparency mode being employed. It should also be noted that since it is the underlay that is being modified, changing these properties on the **ImageDevice** will have an effect on all containers that are hooked to that device   not simply the container within which the **ImageDevice** resides.

In Floating Window mode, the frame bufferorigin is mapped to the upper left corner of the VGA monitor (with the output origin, pan and scroll properties set to 0) regardless of the position of the container. The container acts as a floating window displaying different areas of the frame buffer depending on its location on the VGA screen.   In the Locked Window case, the frame buffers origin is mapped to the origin of the container (again, with the output origin, pan and scroll properties set to 0), regardless of its location on the VGA screen.

The developer should be aware that certain interactions arise when a Locked Window mode container and another transparent mode container (either locked or floating) share the VGA screen.   Due to hardware limitations, changing the position of the Locked Window mode container results in an identical translation in the image contained within the second container.   If the second container is also a Locked Window container, this is typically an undesirable, but unavoidable effect.   As such, it is recommended that an application have only a single Locked Window mode container.   However, secondary Floating Window containers are fine   they will shift in step with the Locked Windows.

Image Tools Theory: VGA-only Mode Display

In VGA-only mode, display is controlled exclusively by the hook (be it the bundled hook in the frame store, or an actual ImageHook control). What is displayed on the VGA monitor (and how it is displayed) is controlled by the following output properties of the **ImageHook** or frame store: Briefly these are: the OutputPan and OutputScroll positions which specify what part of the image is to be displayed. The pixel in the Frame Buffer at (*OutputPan*, *OutputScroll*) will be displayed at the upper-left corner of the Container. Note that the amount of the frame store displayed is implicit: it depends on the *Width* and *Height* of the container. The *OutputZoomX* and *OutputZoomY* properties specify the zoom factor used when transferring from the frame store to the display. These are floating point numbers; any positive zoom factors are allowed. Factors less than 1.0 cause minification of the image while numbers greater than 1.0 cause magnification. A nearest-neighbor interpolation is used to deal with non-integral zooms.

Image Tools Theory: Dual Monitor Considerations

When the system requires the use of a secondary monitor for image display purposes, the developer must be aware of certain special considerations.   When displaying an image in one of the transparent modes, any alterations to the output properties of the device will alter the secondary image in an identical fashion as the image displayed on the VGA screen.   On the other hand, when these properties are altered on a device in Non-Transparent mode they have no effect on the image displayed on the secondary monitor.   The reason being, that the properties are modifying the Overlay as opposed to the Underlay.   If it is necessary to modify the underlay when displaying a non-transparent image, you must unhook the ImageDevice (by setting its *Transparent* property appropriately) and use an explicit ImageHook to handle the image display on the VGA screen.   Now, with the **ImageDevice** unhooked, any alterations to the devices output properties will only affect the underlay and hence will only have an effect on the secondary monitors image.   If you wish to modify the image on the VGA screen you must employ the **ImageHooks** output properties accordingly.

Image Tools Theory: Grabbing an Image

The ImageDevice has a series of properties that are used to control the information that is digitized from an image, as well as where the resulting data is written.   Some of these properties control the Input Signal Conditioning, while others manipulate channels and sync sources, as well as control the Image Display.

Image Tools Theory: Signal Conditioning

There are three properties which are used to modify the signal between the source and the Frame Buffer: the *InputGain*, *InputOffset,* and *InputLUT*.

The *InputGain* and *InputOffset* specify what the dynamic range of the input signal will be and what voltage will digitize to black.   Frame Grabbers define these terms in many different ways.   ***XCaliper*** tries to reduce the variations by imposing the following rules on the values: it represents the gain and offset values ranging from 0 to 255.   The lower the value of either one, the darker the resulting image. Exactly what voltage is associated with any of these numbers is undefined, as is whether the gain is added to the offset or it is a separate value (i.e.   whether it is defined as a slope up from the black level or as a white level).

The *InputLUT* is an Array of 256 Integers containing the lookup table used when an image is digitized. Because it is an array, it is not accessible at design time.   To modify the LUT simply write to the array.

Image Tools Theory: Selecting the Image Size

In ***XCaliper***, an input image is assumed to have a nominal size. This is the size of images acquired when the zoom factors are 1 in both X and Y and the entire incoming signal is digitized. For example, RS-170 signals are normally considered to have a nominal size of 480 lines in the Y-direction. In the X-direction, things are a bit more complicated because the signal is continuous. Typically a line will be broken up into either 512 pixels (corresponding to a digitization rate of 9.83Mhz) or 640 pixels (12.27Mhz). In the current implementation, this nominal size is not available directly but is an attribute of the frame grabber and/or camera used.

The ImageDevice has *InputSizeX*, *InputSizeY*, *InputOriginX*, and *InputOriginY* properties. The size properties do NOT give the nominal size of the signal, instead they specify how much of the incoming signal will be digitized. Therefore, although the number might be less than the nominal size, it can never be greater. The origin properties state at what point in the video signal to start digitizing relative to the upper-left corner of the incoming signal.

It is important to understand that these properties determine how the incoming signal is treated. They do not specify how many pixels will be placed in the frame buffer nor where they will be placed. The point in the frame buffer at which digitization starts is determined by the *InputPan* and *InputScroll* properties. The number of pixels digitized is calculated from the size properties together with the corresponding zoom property: *InputZoomX* or *InputZoomY*. It is equal to the *InputSize* divided by the *InputZoom* (i.e. the number of nominal pixels digitized divided by the hold rate).

Image Tools Theory: Image Display

It is best to view the transparent modes as being a mixture of two video signals together on a single monitor.   The ImageDevice is responsible for deciding what is displayed on the image video signal. There are several properties which go together to display what is seen on the monitor As this is so, there are several properties which go together to describe what is seen on the monitor   whether underneath the VGA screen, or on a separate display.

Most of these properties directly correspond to the equivalent input property.   The *OutputOriginX***,** *OutputOriginY***,** *OutputSizeX***,** and *OutputSizeY* describe the placement of the image on the output video signal.   These numbers are fixed on many frame grabber boards because they are either incapable of changing the values or it would be meaningless to do so . The *OutputPan***,** *OutputScroll***,** *OutputZoomX*, and *OutputZoomY* properties describe what part of the frame buffer will be placed on the screen starting at the position in the video signal given by the output origin.

There is also an OutputLut.   This is an array of 256 Long Integers (a Visual Basic data type) which contain RGB values in exactly the same format as other Visual Basic color properties such as *BackColor*.   Like any array, it is not available at design time and hence special color effects are only available when the program is running.   In design mode the *OutputLut* is set to a gray-scale ramp.   It is suggested that you turn the UpdateMode off when writing more than one entry to the *OutputLut***.**   If *UpdateMode* is on, the modification will be much slower and involve disturbing visual effects.   The *OutputLut* of the device is also used by the ImageHook when it displays an image in VGA-only mode. Therefore, consistency between the two modes is ensured.

Image Tools Theory: MemoryBuffers

A MemoryBuffer is simply a restricted type of ImageDevice that resides in system memory instead of a plug-in board. As such it can only be displayed in VGA-only mode. Its *Transparent* property (or that of an ImageHook attached to it) cannot be set to Pass-Through Mode. The tool has an output LUT which will be used by the hook when it is displayed. This is done to be consistent with the way an **ImageDevice** behaves.

Another difference between a **MemoryBuffer** and an **ImageDevice** is that it is possible to dynamically change the amount of image memory allocated in a **MemoryBuffer**. This can be set in one of two ways: if the *AutoSize* property is set to True, the buffer will take on the size and attributes of whatever image is copied into it; if it is set to False, the buffer will have a fixed size as set in the *BufferSizeX* and *BufferSizeY* properties. When an image is copied into the device, it will be clipped to fit.

Image Tools Theory: Device Drivers Restrictions

An ImageDevice is an abstraction of a Frame Grabber.   These devices have a variety of tradeoffs in what they can and cannot do.   Some have offset and gain capabilities, others have LUTs.   Some can support special cameras, others are restricted to RS-170 (TV camera) input.   ***XCaliper*** takes the approach that special capabilities should be supported.   This is especially true in machine vision where the choice of grabber and/or camera may be directly related to these capabilities.

Therefore an abstract frame grabber is not a minimal device but rather incorporates the best capabilities of all.   If a particular grabber does not support a given feature, the driver will attempt to do the best it can and change the property setting to reflect the actual state of the card, not the state requested by the programmer.   As a consequence, unless you know how the driver you are using will react, you should always check the value of a property after you set it to see how the driver reacted.

For example, if you set the InputZoomX to 5.2, one grabber might choose to set it to 4.0 because it only supports zooms in powers of two while the next one might set it to 5.0 and another, which doesnt zoom at all might set it to 1.0.   Another possibility might be the x and y-zooms are linked so that this change to the x-zoom would cause a similar change to the y-zoom.

As a result, you should always know the capabilities of your hardware and how it will react to a request. One feature which can help notify you how the hardware will respond, is the write back feature of Visual Basic.   If a property is set in design mode, Visual Basic will react by writing the property to the control and reading back the values of ALL properties in the control.   If you are not sure how the hardware will react, try to set the property in design mode and examine the changes on the property list as Visual Basic automatically updates all property values any time one is altered.

BlobAnalyzer Theory: Overview

The ***XCaliper*** BlobAnalyzer tool is implemented as a custom control which operates inside a region of interest known as a Viewport. All of the image inside the Viewport is subject to the blob analysis.

The **BlobAnalyzer** tool is used to extract more than thirty different characteristics, or features, of the blobs in a Viewport. It is possible to restrict the list of features extracted in order to reduce execution time and memory requirements. The current implementation is limited to about 15000 blobs in a single image. As even the most complicated scenes never have more than a thousand or so, this should be far more than necessary for any real-world application.

The tool starts by splitting its Viewport into foreground and background pixels based on light intensity or on a user-specified mask. Next, each object in the Viewport is given identifying tags and their features are stored into an internal list. These features may be used to filter out uninteresting objects and to display the resultant blobs. Alternately, the features can be reported back to the programmer or another software module for further processing.

The **BlobAnalyzer** tool is often used in qualitative measurements. Determining presence or absence of a crucial feature in a complex scene or sorting similar parts into different categories are common applications. Normally, the built-in filtering provided with the tool is used to find the object which needs to be measured among all of the objects in the image. In addition, the **BlobAnalyzer** can also be used to determine if a part is good or bad based on the value of one or more of its features. Once the desired object is found, other features or processing tools can be used to perform final measurements.

The **BlobAnalyzer** Theory of Operation on-line documentation provides a discussion on the following topics:

i) Principles of Blob Analysis
ii) Application of the Tool
iii) Examining Extracted Features
iv) Blob Statistics

BlobAnalyzer Theory: Principles of Blob Analysis

Blob analysis is the process of extracting blobs from images in order to obtain statistics and other information about them.   This information can then be used to determine presence or absence, location and many other characteristics of the real-world objects which these blobs represent.

Blob analysis consists of four essential steps:

i) Segmentation
ii) Labelization
iii) Feature Extraction
iv) Filtration

In addition, this tool incorporates mechanisms to display results, to support user interaction with the tool and to read the results of the analysis.

While a complete analysis requires all four of the above operations, it is possible to restrict the actions of the **BlobAnalyzer** tool, thus reducing overhead in those cases where part of the analysis has already been performed.   For example, it is often necessary to Filter the same set of data more than once. Re-extracting this data from the image would be a waste of time.

BlobAnalyzer Theory: Segmentation

Segmentation is the processing step of separating an image into foreground and background, determining which pixels belong to blobs and which belong to the background of the image.

The **BlobAnalyzer** tool segments an image based on pixel intensity. That is, a pixel is attributed to the foreground or to the background based on its gray level intensity. In most real-world images, there is one range of intensities which contains the background pixels and another range which contains the foreground pixels. In addition, there will be an intermediate set of gray level values which represent those pixels which are on the edge of objects. As such, they should not be considered as either foreground or background but rather as transitional values on the edge of objects. Accurate processing of any blob to extract features must take into account these border pixels to get a true representation of the object.

The **BlobAnalyzer** tool supports a WeightTable used to specify what percentage of each pixel is attributed to foreground for all of the 256 possible gray level values. Gray levels below the background threshold are assigned a weighting of 0, those above the foreground level are assigned a weighting of 1, and the gray levels in between are assigned a weighting between 0-1 based on their gray level. For example, assume that the transition region starts at intensity level 82 and continues to 160. A pixel with a value of 85 should be considered mainly part of the background while one with a value of 150 is mainly foreground. Allowing a pixel to be considered as partial foreground gives more precision in the calculation of several blob features, particularly position and area. The more common technique of setting an arbitrary threshold point in the middle of the transition is less accurate.

Three properties, the *LowerThreshold*, *UpperThreshold* and *BlobType* exist to write typical segmentation algorithms into the weight table without requiring the programmer to calculate all 256 entries individually. Note that they serve no purpose on their own but simply make it easier to write to the WeightTable. However, if other unsupported techniques are required, the capability does exist in the **BlobAnalyzer** to write a custom WeightTable. This can be done by writing to the table directly. The *BlobType* property selects among five schemes and the *LowerThreshold* and *UpperThreshold* properties supply associated limits.

The weighting rules are modified by the *HoleFill* property. If *HoleFill* is set to True, holes inside blobs are filled in when segmentation occurs. Hole-filling is the process whereby background areas (and other blobs) which are inside a blob are considered as part of that blob for the purpose of later analysis . A hole is defined as an area of one or more pixels with weights of less than 1.0 inside a blob which is entirely surrounded by pixels whose weightings are 1.0. A pixel is considered to be inside a blob if it is entirely surrounded by a connected area of pixels which are 100% foreground. Pixels in the hole, even those with weights of 0.0, will have their weights changed to 1.0 after segmentation.

The **BlobAnalyzer** tool supports a second method of describing what part of the image is to be considered foreground and what part is background. This method involves segmentation using the Mask.

BlobAnalyzer Theory: Segmentation Using the Mask

The **BlobAnalyzer** tool supports an alternate method of describing what part of the image is be considered foreground and what part is background. The *Mask* property allows specification of foreground and background on a pixel-by-pixel basis. This can be done by copying a binary bitmap into the *Mask* as follows:

```
BlobAnalyzer1.Mask = picture1.Picture
```

then using an *Apply* mode which does not segment the image, such as `AM_MASK_LABEL` or `AM_MASK_NO_LABEL`, since the Mask provides the segmentation. When a segmentation mask is supplied, analysis is performed on the area which is the intersection between the *Mask* and the *WeightTable*. The *Mask* will determine which blob, if any, a pixel belongs to while the *WeightTable* will determine its relative weighting within the blob. If the intent is to use only the Mask and ignore the WeightTable all together, then the *BlobType* should be set to `BT_MASKONLY`. In this mode pixels which are set in the mask are considered foreground pixels those which are not set are part of the background.

It is also possible to extract the *Mask* from the **BlobAnalyzer** tool as shown below. However, the tool mask will only contain background pixels unless previously set using the *Mask* property or one of the segmenting *Apply* modes.

```
picture1.Picture = BlobAnalyzer1.Mask
            or
SavePicture  BlobAnalyzer1.Mask, <FILENAME>
```

When a segmentation mask is extracted from the **BlobAnalyzer** it is converted from a weighted form to a binary form. Pixels with a non-zero weight are considered part of the foreground. Pixels with a zero weight are considered to be part of the background. While it is necessary, a segmentation must be applied to obtain a valid mask. If not, the mask will contain only background pixels. Currently there is no method to save either the weighting or blob identifiers.

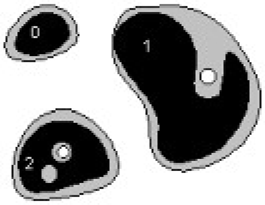
BlobAnalyzer Theory: Labelization

This is the process of assigning unique identifiers to each blob in the image. The segmented image is scanned from top left to bottom right and each connected group of blobs is assigned a unique number starting with 0 for the first blob and continuing on to the last blob in the image. Blob numbers are assigned in scan-order. The following is the eight connectivity, neighboring pixels used in labelization:

1	2	3
4		5
6	7	8

This means that pixels are considered to be part of the same blob if they touch in any of the eight major directions including diagonals. The weighting assigned to each pixel is ignored for the labelization process. As long as the weight of two adjoining pixels is above zero, they will be considered part of the same blob. Therefore, once the labelization process is complete, labeled blobs do not touch in any direction and are completely isolated by pixels which are 100% background.

For example, the following diagram shows three blobs numbered in scan-major order just as the tool would do. The hole in blob 1 will not be filled in as there is no connected area of fully foreground while the partial foreground pixels on the boundary will be left unchanged.



BlobAnalyzer Theory: Feature Extraction

Feature extraction is the process whereby information is extracted about each of the blobs contained in the image.   The **BlobAnalyzer** tool, in its current release, supports over 30 different features including such parameters as area, location, perimeter, orientation, shape, density and texture.   The tool extracts only those features which are actually requested, thus saving processing time and memory.   The information obtained may be retrieved for analysis by the program or passed to other VBXs or programs such as the grid and graph tools supplied by Microsoft.

The list of features extracted can be specified in two ways: through use of the Filters and Features dialog box or by specifying the features individually at run time through Visual Basic code.   The dialog box is simple to use and is often the best way to play with an image and decide exactly which features are needed to extract and which filters to apply.   However, it is only able to specify a single set of analysis and filter criteria for the entire application.   This will not be sufficient if either the list of features or, as is more often the case, the filters and filter limits will have to change dynamically.   Using Visual Basic code at run time allows the designer to change the filter criteria while the program is executing.

All features extracted are scaled and translated, if appropriate, in both the x and y direction according to the current scaling mode.   This mode is controlled by three properties which determine the scaling factor and two properties which control the origin of the co-ordinate system.   The *ScaleX* and *ScaleY* properties give the scale factors in X and Y respectively.   *ScaleType* is an enumerated integer which determines what generates these scaling factors.

The **BlobAnalyzer** tool is currently capable of extracting the following features from blobs:

Area
Moment1X
Moment1Y
Moment2X
Moment2Y
MomentXY
BoundBoxLeft
BoundBoxTop
BoundBoxRight
BoundBoxBottom
Perimeter
FormFactor
TouchesBorder
Feret0
Feret90
Orientation
Eccentricity
EquivDiameter
EulerNumber
NumHoles
GrayMass
GrayMean
GrayStdDev
GrayMin
GrayMax
GrayRange
GrayContrast
GrayTexture

BlobAnalyzer Theory: Application of the Tool

The basic programming technique for the BlobAnalyzer tool is to set up the analysis desired in Design mode using Visual Basic properties which control each of the four phases of the blob analysis.   Once all the properties are set, the Apply property must be set to its appropriate value to cause the analysis to take place.   When the value of this property is written, the **BlobAnalyzer** executes, updating its internal data lists and, optionally, the display.

The main set up operations include specifying which features to extract and which filters to apply. Other considerations include calibration of the analysis to real-world co-ordinates and determining the order in which blobs are to be reported back to the programmer.

BlobAnalyzer Theory: Filtration

Once the features have been extracted for all the blobs in a region of interest, uninteresting blobs can be eliminated from further analysis through the process of Filtering. For example, it might be desired to exclude all blobs which have an area below three pixels and all those which touch the border of the **BlobAnalyzer**. This would be done in order to set a noise threshold and to eliminate partial objects from the analysis. The next step would be to find those blobs with some interesting characteristic, for example a highly deformed shape.

In a typical application, these blobs would be tagged and the abnormal shape signaled to the user in some fashion. To that end, the tool supports mechanisms for displaying all blobs or just those blobs which pass or fail a filter criteria. It is also possible to set up multiple classes of blobs. In this example, it may be desired to color the deformed blobs red and the other ones green. Perhaps the small ones should not be colored at all to reduce clutter will those which touch the border are set to black.

Two properties exist to select the list of features to extract and to enable or disable filtration: *Features* and *Filters*. These properties are arrays indexed by feature number. *Features* is an array of enumerated integers which specify whether a given feature is to be extracted and, if so, whether it should be filtered as well. Assuming filtration is enabled for a given property then the *Filters* array specifies exactly which filter to apply.

For example, assume that we wish to extract the area and orientation of each blob and to report back a count of those blobs with an orientation between 0º and 90º. We need therefore to set the area and orientation entries in the *Features* array and the orientation entry in the *Filters* array as follows:

```
BlobTool1.Features(BA_AREA) = FE_EXTRACT
BlobTool1.Features(BA_ORIENTATION) = FE_EXTRACT_AND_FILTER
BlobTool1.Filters(BA_ORIENTATION) = [0,90]
BlobTool1.Apply = AM_SEGMENT_LABEL
MsgBox Found + Format$(BlobTool1.Count) + blobs
```

BlobAnalyzer Theory: Displaying Blobs

The display properties are used to show the results of the blob analysis overlaid on the image. Blobs are sorted according to whether they passed or failed the active filter criteria. The *AcceptDisplay* and *RejectDisplay* properties control how the tool reacts to blobs which pass and to blobs which fail.

By setting the *RejectDisplay* property to NoChange and the *AcceptDisplay* property to Color, it is possible to segment the blobs into multiple categories by applying successive Filters, one for each class. Choose a color for a class. Apply the tool with the appropriate filter set. The blobs which pass the filter will appear in the correct color. Now choose a second filter and a second color for the next class. Apply the tool again. The blobs which passed the first filter will remain in the first color while those which passed the second will appear in the second color. Repeat this operation once for each class. If it is desired to have everything appear to happen at once, perform these operations with the tools *Visible* property set to False. When the sequence is finished, set the *Visible* property to True.

The last property controlling blob display is the *DisplayMode*. This chooses the appearance of blobs on the screen. This property applies to all blobs on the screen, not just the last group to pass or fail a filter.

BlobAnalyzer Theory: Examining Extracted Features

In many applications it is sufficient to simply set up some Filters and determine how many blobs pass them. However, sometimes it may be necessary to actually analyze the information extracted about features. In the **BlobAnalyzer**, two methods are supported for accessing the data: the results can be returned feature by feature, or as a single string containing all the data extracted from the image.

BlobAnalyzer Theory: Examining Individual Features

An individual feature datum can be retrieved through use of three properties: *CurValue*, *CurFeature* and *CurBlob*.   For example, the following code would obtain the area of the fifth blob:

```
BlobTool1.CurFeature = BA_AREA
BlobTool1.CurBlob = 5
f = BlobTool1.CurValue
```

In practice the end user would almost always want to access multiple pieces of data.   A built-in Auto-increment mode automatically increments the values of the *CurFeature* and *CurBlob* properties on each access to the *CurValue* property making code to process the data simpler and faster.   The Autoincrement mechanism, supplied through the *CurAutoInc* property, can also be used to cycle through all the features of a blob as opposed to the same feature in all the blobs.   For example, the following code fragments would obtain the sum of the areas of all the blobs in an image:

```
BlobTool1.CurAutoInc = AI_BY_BLOB
BlobTool1.CurFeature = BA_AREA
area = 0
For I = 0 To BlobTool1.Count - 1
    area = area + BlobTool1.CurValue
Next I
```

Alternatively, this could be written less efficiently, as:

```
BlobTool1.CurAutoInc = AI_NONE
BlobTool1.CurFeature = BA_AREA
area = 0
For I = 0 To BlobTool1.Count - 1
    BlobTool1.CurBlob = I
    area = area + BlobTool1.CurValue
Next I
```

The best method, however, would be to obtain the Sum statistic.

Autoincrement only goes through the set of features actually enabled and through the set of blobs which passed the last filtering operation.   Note, however, that all the data actually extracted at the last blob analysis is saved and therefore can be retrieved.   Consider the following code fragment:

```
Rem extract area and orientation, filter on area
BlobTool1.Features(BA_AREA) = FE_EXTRACT_AND_FILTER
BlobTool1.Features(BA_ORIENTATION) = FE_EXTRACT
BlobTool1.Features(BA_NUM_HOLES) = FE_EXTRACT
BlobTool1.Filters(BA_AREA) = (100,1000)
BlobTool1.Apply = AM_SEGMENT_LABEL
Rem at this point auto-inc would go through three features
Rem and all the blobs which passed the filter

BlobTool1.Features(BA_NUM_HOLES) = FE_DONT_EXTRACT
Rem at this point auto-inc would only access two features

BlobTool1.Features(BA_AREA) = FE_EXTRACT
BlobTool1.Apply = AM_FILTER_ORIGINAL
Rem this would cause the filter to be discarded
Rem all of the blobs would be returned
```

Note that eliminating the `BA_NUM_HOLES` feature took effect immediately while eliminating the filter had to wait until the next apply. This is because eliminating the filter requires a re-evaluation of the data while eliminating a feature merely changes the list of which things are to be returned. At this point it would be possible to add back the `BA_NUM_HOLES` feature since it was extracted from the image and the information still exists. However an attempt to read information about a feature which was not extracted, for example the perimeter, would cause a Visual Basic error.

BlobAnalyzer Theory: Extracting All the Data at Once

It is possible to obtain all the data at once using the *QuickData* property.   It returns all of the data in a single Basic string.   This string is a 2-dimensional table with values separated by <TAB> characters in the horizontal direction and <CR><LF> pairs in the vertical direction.   This format is the one used by Visual Basics **Grid** and **Graph** tools and hence can be used to copy data into those controls for display or analysis.   The order of the table depends on the setting of the *CurAutoInc* property.   If *CurAutoInc* is set to `AI_BY_FEATURE`, features will be shown along the horizontal axis; if it is set to `AI_BY_BLOB`, blobs will be shown horizontally.   The string may also be labeled, if desired, by setting the *QuickLabels* property to True.

The following is an example of *QuickData* results with *QuickLabels* on and CurAutoInc set to `AI_BY_BLOB`:

	0	1	2	3
Area	27.23	56.97	45.33	83.68
NumberOfHoles	1	0	0	0
GrayMean	187.39	163.77	192.44	62.2

BlobAnalyzer Theory: Blob Statistics

The BlobAnalyzer tool can perform statistical analysis of the distribution of blobs according to any feature. The Stats property is an array of values from which a variety of statistical measures can be obtained. Statistics are only available from code and are obtained by indexing the array with an enumerated integer type as:

```
sd = ctlName.Stats(statsNumber%)
```

The *Stats* property is affected by the CurFeature property which specifies which feature is analyzed. Statistics are calculated on request. E.g. accessing the array causes the statistic in question to be calculated. The following statistics can be extracted:

Mean
StdDev
Mode
Median
Min
Max
Range
Sum
SumOfSquares
Variance

Area

The area in pixels of a blob.

Moment1X

The x-component of the blob first moment (otherwise known as centroid).

Moment1Y

The y-component of the blob first moment (otherwise known as centroid).

Moment2X

The standard deviation of all points in the blob from the centroid in the x-direction.

Moment2Y

The standard deviation of all points in the blob from the centroid in the y-direction.

MomentXY

The tendency of the X and Y moments to deviate in the same manner..

BoundBoxLeft

The left side of the smallest enclosing rectangle that can hold the blob.

BoundBoxTop

The top side of the smallest enclosing rectangle that can hold the blob.

BoundBoxRight

The right side of the smallest enclosing rectangle that can hold the blob.

BoundBoxBottom

The bottom side of the smallest enclosing rectangle that can hold the blob.

Perimeter

The actual perimeter of the blob, in pixels.

FormFactor

The ratio of the blobs perimeter to the area. Ranges from 0.0 to 1.0 such that 1.0 indicates that the blob is a perfect circle.

TouchesBorder

Set to 1.0 if the blob touches the border of the Viewport and to 0.0 if it does not.

Feret0

The diameter of the blob in the 0 direction (e.g. BoundBoxRight - BoundBoxLeft).

Feret90

The diameter of the blob in the 90 direction (e.g. BoundBoxTop - BoundBoxBottom).

Orientation

The axis of minimum inertia in engineering degrees (e.g. 0 is due East and numbers increase in the counter-clockwise direction).

Eccentricity

Moment-based shape factor where 0.0 indicates a perfect circle.

EquivDiameter

Diameter of a circle with the same area as the selected blob.

EulerNumber

Number of particles in the blob minus the number of holes.

NumHoles

Number of holes in the blob.

GrayMass

The sum total of the pixel values of the blob.

GrayMean

The average pixel value in the blob.

GrayStdDev

The standard deviation of gray levels in the blob from the average value.

GrayMin

The lowest gray level in a blob.

GrayMax

The highest gray level in a blob.

GrayRange
The difference between the min and max gray levels in a blob.

GrayContrast

The average difference of a pixel in gray level from its eight nearest neighbors. As currently implemented this feature takes into account neighbors which are outside the blob. Hence the value tends to become unusable for very small blobs (those with a diameter below about 7 pixels).

GrayTexture

Slope of the line through the **GrayContrasts** as the distance of the neighbors from the pixel gets larger. Currently the slope is calculated from the **GrayContrasts** at distances of 1, 2, 3 pixels.

BlobAnalyzer

Properties Commands Events Methods

Description

The **BlobAnalyzer** tool analyzes characteristics of objects within a Viewport. It can generate both binary and gray level information about objects, or blobs. The tool can segment images into foreground and background, assign unique labels to all the objects in the resulting binary image, extract characteristics or features of each blob, and finally performing a filtration process to eliminate unwanted blobs. The filtration process can be set up in software or by using the Setup Dialog Box.

File Name

XVBXBLOB.VBX

Remarks

The control is represented in the Visual Basic toolbox by an icon depicting a blob superimposed by a question mark and a ruler lying horizontally below the blob. When placed in a container, it is represented by a rectangular Viewport in which the blob analysis will take place.

Properties

All of the properties are listed in the following table. Properties that are not standard or that require special consideration when used with this control are marked with an asterisk (*). For information on standard Visual Basic properties, please see the Visual Basic Programmer's Guide or the Visual Basic on-line Help.

About*
AcceptColor*
AcceptDisplay*
Apply*
ApplyOnChange*
BlobType*
Command*
Count*
CurAutoInc*
CurBlob*
CurFeature*
CurValue*
DisplayMode*
DrawColor*
ElapsedTime*
Features*
Filters*
Height
HoleFill*
Index
Left
LowerThreshold*
Mask*
Name
QuickData*
QuickLabels*
RefType*
RejectColor*
RejectDisplay*
ScaleType*
ScaleX*
ScaleY*
Setup*
SortFeature*
SortOrder*
Stats*
Tag*
Top
UpperThreshold*
UserInteface*
Visible
WeightTable*
Width

Commands

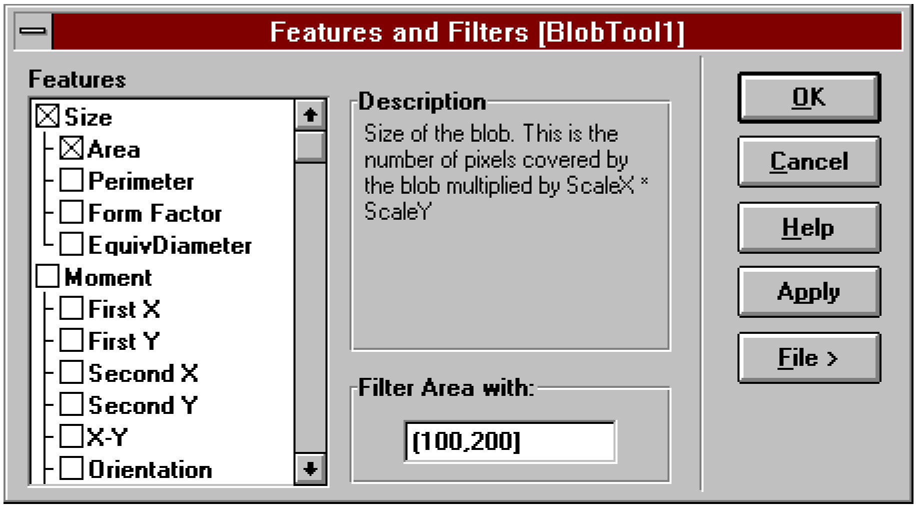
All of the commands are listed in the following table. Commands can be executed either during design time or run time. To execute a command during design time, enter the command into Command property of a **BlobAnalyzer**.

- Clear
- Close
- Convolve
- Copy
- Dilate
- Erode
- LoadFile
- Median
- Open
- SaveFile
- TuneInput

BlobAnalyzer Setup: Features & Filters

The Features and Filters Dialog Box provides the designer or end user a method to review and change the features extracted from a set of blobs in a region of interest known as a Viewport, and the filter criteria used on those features to eliminate unwanted blobs from the list.   This dialog box is invoked at design time by double-clicking the Setup property in the Properties window, and is invoked at run time either by program action or by double-clicking inside the tool's Viewport if the UserInterface property is to allow this action.

Click on the dialog box for help on a specific item.



Features and Filters [BlobTool1]

Features

☒ Size
- ☒ Area
- □ Perimeter
- □ Form Factor
- □ EquivDiameter

□ Moment
- □ First X
- □ First Y
- □ Second X
- □ Second Y
- □ X-Y
- □ Orientation

Description

Size of the blob. This is the number of pixels covered by the blob multiplied by ScaleX * ScaleY

Filter Area with:

[100,200]

BlobAnalyzer Setup: Feature Check List

The Feature Check List is a list box containing the features that can be extracted from the blobs in a region of interest.   Beside each feature is a check box that allows the feature to become enabled.   The features are organized into groups of similar features, where each feature is connected by hierarchical lines that lead to the parent feature.   Selecting any feature within a group causes its parent feature to be selected as it would have to be included in the calculation required for each blob..   When a parent feature is selected or de-selected, all of its children become selected or de-selected.

To select or de-select a feature, the cursor must be over the check box for the feature.   If a feature name is selected, it will be highlighted and the relevant filter string, if any, will be displayed on the Filter Edit Box.

BlobAnalyzer Setup: Filter Edit Box

The Filter Edit Box allows the designer to eliminate those blobs that do not meet specific filter criteria. This is done by defining a set of rules through which the features that have been calculated for the blobs in a Viewport must pass. Filters can be set by entering a string into the Filter Edit Box. This same string can also be specified in Visual Basic code. To view or change a filter, first select the desired feature from the Feature Check List by clicking on the feature name. To modify the filter, enter the criteria using the same syntax as that used in Visual Basic code. The string entered is then validated and will be rejected if the syntax is incorrect.

BlobAnalyzer Setup: Filter Syntax

A filter is defined using a string which must adhere to the following syntax:

[NOT] { (| [} [NUMBER] , [NUMBER] {] |) }

[NOT] - an optional operator that inverts the logical sense of the filter

{ (| [} - a required symbol that indicates greater than ([ ) or greater than or equal to (( )

[NUMBER] - a floating point number that indicates a lower or upper bound

{] |) } - a required symbol that indicates less than ( ]) or less than or equal to ( ))

Examples:

5 < Area < 10	[5,10]
Area <= 45	[,45)
Area <= 5 OR Area >= 10	NOT [5,10]
	[10,5]

Apply Button

The **Apply Button** will apply the tool, updating the display showing those blobs which pass the new set of filter criteria.

Description Box

The **Description Box** contains a brief description of the currently selected feature.

LightMeter Theory: Overview

The ***XCaliper*** LightMeter tool is implemented as a Custom Control which operates inside a Viewport.

The **LightMeter** tool extracts information about light levels in its Viewport.   It can obtain the Histogram of intensity values in the Viewport, or extract various Statistical information about light levels of the Viewport.   This tool is most often used to detect failures in the camera or lighting equipment of an automatic inspection process.   As a result, it is often the first tool to be executed in an inspection cycle. The tool can also be used for simple inspection tasks, such as detecting the presence or absence of a part or performing simple texture analysis, simply by looking at the shape and distribution of the histogram of the appropriate region of interest.

The **LightMeter Theory of Operation** on-line documentation provides a discussion on the following topics:

i) Application of the Tool
ii) Statistical Quantities
iii) Histogram
iv) Examining the Data

LightMeter Theory: Application of the Tool

Using the LightMeter control is as straightforward as using the mouse to draw a rectangular box on the screen image. The rectangular region specifies the region of interest or Viewport that defines which pixels are to be included in the generation of the Histogram and other Statistical data.

The programmer can then generate, simply by reading or writing the Apply property, a histogram of the intensity values contained within the Viewport.   If the UserInterface property has been set appropriately, an implicit *Apply* will also occur whenever the Viewport is repositioned or re-sized.

LightMeter Theory: Examining the Data

The LightMeter tool offers two methods for examining the data generated when the control is Applied. One method is to access each data point in the histogram array through the Histogram property.   An alternative is through the use of the *QuickData* property, which is similar to the *QuickData* property of the **Graph** control.   Through this property, the programmer can quickly and easily access all the histogram data at once for further processing and analysis.

LightMeter Theory: Statistical Quantities

The LightMeter tool can provide Statistical measures through the read-only array property, *Stats*. Statistics are computed from the pixel intensity data obtained the last time the control was applied through the *Apply* property.   These quantities are only available from code and are obtained by indexing the array with an enumerated integer type as:

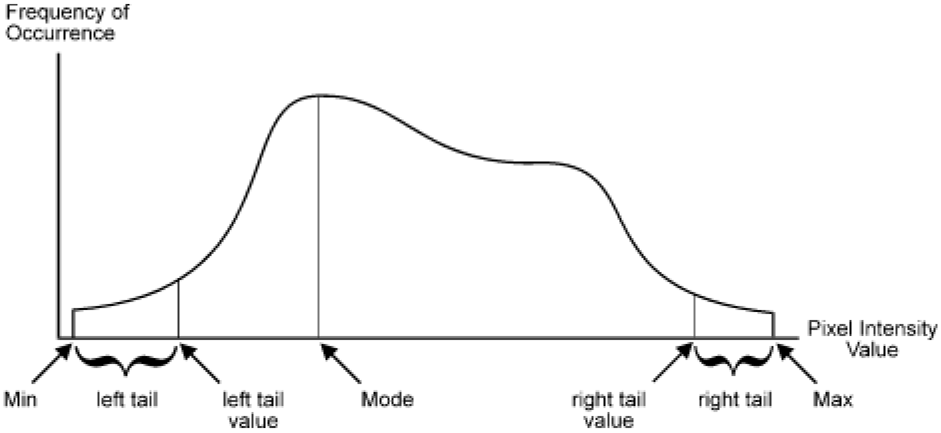
```
sd = ctlName.Stats[StdDev]
```

Statistics are calculated on request.   E.g.   accessing the array causes the statistic in question to be calculated.   The following statistics can be extracted:

Mean
StdDev
Mode
Median
Min
Max
Range
Sum
SumOfSquares
Variance
LeftTail
RightTail

LightMeter Theory: Histogram

The Histogram generated by the LightMeter tool is a plot of the pixel intensity values versus the frequency of occurrence of those intensity values. The pixel intensity values are plotted along the x-axis while the frequency of occurrence of those intensity values is plotted along the y-axis.



Two areas of the histogram worthy of further mention are the Left Tail and Right Tail regions. These represent the extremes of the data, and are often considered unreliable or biased due to noise, signal saturation, etc., and as such, pixels falling into these regions are often ignored in subsequent analysis. Exactly how large the tail sizes should be is left up to the programmer. Through the *LeftTailSize* and *RightTailSize* properties, the programmer can set the percentage area of the total histogram that should be considered part of the left or right tail respectively. This percentage can also be interpreted as a percentage of the total number of pixels under examination.

After generating a histogram for a particular region of the image, the programmer can obtain, through the *Stats* property, the left and right tail values - the pixel intensity values that bound the left and right tails.

Mean
The average value.

StdDev

The standard deviation of a particular value from the average value.

Mode

The most common value.

Median

The point at which half the blobs have a value below and half the blobs have a value above.

Min

The lowest value of any blob.

Max

The highest value of any blob.

Range

The difference between the Min and the Max.

Sum

The arithmetic sum of the values of all blobs.

SumOfSquares

The arithmetic sum of the squares of the values of all blobs.

Variance

The variance of a particular value from the average value (effectively the square of the StdDev).

LeftTail

The smallest intensity value such that one or more (and possibly all) of its pixels are not part of the Left tail as determined by the *LeftTailSize* property.

RightTail

The largest intensity value such that one or more (and possibly all) of its pixels are not part of the right tail as determined by the *RightTailSize* property.

LightMeter

Properties Commands Events Methods

Description

The **LightMeter** generates statistics of the intensity values inside a Viewport. The tool provides the capability to obtain and manipulate a Histogram of the intensity values. It also allows the program designer and/or user to obtain a variety of Statistical quantities derived from these values.

File Name

XVBXLMTR.VBX

Remarks

The control is represented in the Visual Basic Toolbox by a picture of a light bulb with a scale across it. When placed on a form, it is represented by a rectangular box with eight pick points on its perimeter. All processing is performed on the intensity values inside the box.

Properties

All of the properties are listed in the following table. Properties that are not standard or that require special consideration when used with this control are marked with an asterisk (*). For information on standard Visual Basic properties, please see the Visual Basic Programmer's Guide or the Visual Basic on-line Help.

About*
Apply*
ApplyOnChange*
Area*
Command*
DrawColor*
ElapsedTime*
Height
Index
Left
Histogram*
LeftTailSize*
LowerLimit*
Name
QuickData*
QuickDataMode*
RightTailSize*
Setup*
Stats*
Tag*
Top
UpperLimit*
UserInteface*
Visible
Width

Commands

All of the commands are listed in the following table. Commands can be executed either during design time or run time. To execute a command during design time, enter the command into Command property of a **LightMeter**.

- Clear
- Close
- Convolve
- Copy
- Dilate
- Erode
- LoadFile
- Median
- Open
- SaveFile
- TuneInput