# IMAQ™

# IMAQ Vision for LabWindows/CVI Reference Manual

NATIONAL INSTRUMENTS™

January 2000 Edition
Part Number 322567A-01

**Worldwide Technical Support and Product Information**

www.ni.com

**National Instruments Corporate Headquarters**

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 794 0100

**Worldwide Offices**

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Calgary) 403 274 9391, Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521,
China 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11, France 01 48 14 24 24,
Germany 089 741 31 30, Greece 30 1 42 96 427, Hong Kong 2645 3186, India 91805275406,
Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Mexico (D.F.) 5 280 7625,
Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466, New Zealand 09 914 0488, Norway 32 27 73 00,
Poland 0 22 528 94 06, Portugal 351 1 726 9011, Singapore 2265886, Spain 91 640 0085,
Sweden 08 587 895 00, Switzerland 056 200 51 51, Taiwan 02 2528 7227, United Kingdom 01635 523545

For further support information, see the *Technical Support Resources* appendix. To comment on the documentation, send e-mail to techpubs@ni.com

# Important Information

## Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

## Copyright



## Trademarks



## WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

# Conventions

The following conventions are used in this manual:

This icon denotes a note, which alerts you to important information.

**bold** Bold text denotes items that you must select or click on in the software, such as menu items and dialog box options. Bold text also denotes parameter names.

*italic* Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.

`monospace` Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.

# Contents

## Chapter 1
## Basic Concepts



## Chapter 2
## Image Management

## Chapter 5
## Acquisition



## Chapter 6
## Display

## Chapter 7
## Regions of Interest

## Chapter 8
## File I/O



## Chapter 9
## Image Analysis



## Chapter 10
## Grayscale Processing

## Chapter 11
## Binary Processing



## Chapter 12
## Color Processing

## Chapter 13
## Pattern Matching



## Chapter 14
## Caliper



## Chapter 15
## Operators

# Figures



# Tables

# 1

# Basic Concepts

This chapter explains the basic ideas underlying image processing with IMAQ Vision for LabWindows/CVI. For more information about images and image processing, see the *IMAQ Vision User Manual.*

## Image Properties

An image is a two dimensional array of values representing light intensity. An image has the following properties: spatial resolution (size), definition (pixel depth), and number of planes. An image's pixel depth and number of planes combine to determine the type of image.

The IMAQ Vision library can manipulate three categories of images: grayscale, color, and complex images. The most common image type for the scientific and industrial fields is an 8-bit grayscale image. IMAQ Vision can also acquire and process 16-bit grayscale images, floating point images, complex images, and color images in Red, Green, and Blue (RGB) or Hue, Saturation, and Luminance (HSL) format.

In IMAQ Vision, you define an image type by using the `imaqCreateImage( )` function to create the image object. IMAQ Vision uses the following enumerated values to represent the possible image types.

**Table 1-1.** IMAQ Vision for LabWindows/CVI Image Types

| Value | Description |
|---|---|
| `IMAQ_IMAGE_U8` | 8 bits per pixel—unsigned, standard monochrome |
| `IMAQ_IMAGE_I16` | 16 bits per pixel—signed, monochrome |
| `IMAQ_IMAGE_SGL` | 32 bits per pixel—floating point, monochrome |
| `IMAQ_IMAGE_COMPLEX` | $2 \times 32$ bits per pixel—floating point, native format after a Fast Fourier Transform (FFT) |
| `IMAQ_IMAGE_RGB` | 32 bits per pixel—standard color |
| `IMAQ_IMAGE_HSL` | 32 bits per pixel—color |

In addition to spatial resolution, definition, and number of planes, IMAQ Vision images have border and pixel calibration properties.

The image border attribute physically reserves additional space around the image. You must use borders when you perform a morphological transformation, a convolution, or a particle analysis. These calculations include operations that assign a new value to a pixel in relation to the value of its neighbors. By using image borders, IMAQ Vision can perform these calculations quickly.

The calibration attribute defines the physical horizontal and vertical dimensions of the pixels. With the ability to calibrate two axes independently, you can correct skew resulting from a camera sensor. Only calculations based on morphological transformations use this calibration information. Calibration has no effect on processing or operations between images.

# Overview

**Note** You must install LabWindows/CVI 5.0.1 or higher before installing IMAQ Vision for LabWindows/CVI.

The IMAQ Vision installer adds three important files to your current LabWindows/CVI directory:

- `NIVision.h` contains all constants, enumerated types, structures, and prototypes related to IMAQ Vision.
- `NIVision.lfp` contains all function panels. These panels are similar to the other LabWindows/CVI function panels.
- `NIVision.lib` is an import library for the IMAQ Vision functions.

The IMAQ Vision installer also adds the following files to your system directory. These files contain function code for IMAQ Vision.

- `NIVision.dll`
- `NIVisSvc.dll`

The IMAQ Vision installer also adds the following files to your Program Files\National Instruments\IMAQ Vision directory. Use these files with Microsoft Visual C++ and Borland C++/C++ Builder.

- `NIVision.h` contains all constants, enumerated types, structures, and prototypes related to IMAQ Vision.
- `NIVision.lib` is an import library for the IMAQ Vision functions.

## Application Development Environments

This release of IMAQ Vision for LabWindows/CVI supports the following Application Development Environments (ADEs) for Windows 2000/NT/9*x*.

- LabWindows/CVI version 5.0.1 and higher
- Borland C/C++ version 5.0 and higher
- Borland C++ Builder 3.0 and higher
- Microsoft Visual C/C++ version 6.0 and higher

**Note** Although IMAQ Vision has been tested and found to work with these ADEs, other ADEs may also work.

## IMAQ Vision Function Tree

The IMAQ Vision function tree (`NIVision.lfp`) contains separate classes corresponding to groups or types of functions. Table 1-2 lists the IMAQ Vision function types and gives a description of each type.

**Table 1-2.** IMAQ Vision for LabWindows/CVI Function Types

| Function Type | Description |
|---|---|
| Image Management | Functions that create space in memory for images and perform basic image manipulation. |
| Memory Management | A function that returns memory you no longer need back to the operating system. |
| Error Management | Functions that set the current error, return the name of the function in which the last error occurred, return the error code of the last error, and clear any pending errors. |
| Acquisition | Functions that acquire images through an IMAQ hardware device. |
| Display | Functions that cover all aspects of image visualization and image window management. |
| Regions of Interest | Functions that create and manipulate regions of interest. |
| File I/O | Functions that read and write images to and from disk files. |
| Image Analysis | Functions that compute the centroid of an image, profile of a line of pixels, and the mean line profile. This type also includes functions that calculate the pixel distribution and statistical parameters of an image. |

**Table 1-2.** IMAQ Vision for LabWindows/CVI Function Types (Continued)

| Function Type | Description |
|---|---|
| Grayscale Processing | Functions for grayscale image processing and analysis. |
| Binary Processing | Functions for binary image processing and analysis. |
| Color Processing | Functions for color image processing and analysis. |
| Pattern Matching | Functions that learn patterns and search for patterns in images. |
| Caliper | Functions designed for gauging, measurement, and inspection applications. |
| Operators | Functions that perform arithmetic, logic, and comparison functions with two images or with an image and a constant value. |
| Analytic Geometry | Functions that perform basic geometric calculations on an image. |
| Frequency Domain Analysis | Functions for the extraction and manipulation of complex planes. Functions of this type perform FFTs, inverse FFTs, truncation, attenuation, addition, subtraction, multiplication, and division of complex images. |
| Barcode | A function that reads a barcode. |
| LCD | Functions that find and read seven-segment LCD characters. |
| Meter | Functions that return the arc information of a meter and read the meter. |
| Utilities | Functions that return structures, and a function that returns a pointer to predefined convolution matrices. |

# Using Images with IMAQ Vision for LabWindows/CVI

To create an image in IMAQ Vision for LabWindows/CVI, call `imaqCreateImage()`. This function returns an image reference you use when calling other IMAQ Vision functions. The only limitation to the size and number of images you can acquire and process is the amount of memory installed in your computer.

## Source and Destination Images

All IMAQ Vision functions that modify the contents of an image have source image and destination image input parameters. The source image is the original image you want to process. The destination image is the result of the image processing. If you do not want the

contents of the original image to change, use separate source and destination images. If you want to replace the original image with the processed image, pass the image as both the source and destination. IMAQ Vision resizes the destination image to hold the result if the destination is not the appropriate size.

The following examples illustrate source and destination images with the `imaqTranspose()` function:

- `imaqTranspose(myImage, myImage);`

  This function creates a transposed image using the same image for the source and destination. The contents of `myImage` change.

- `imaqTranspose(myTransposedImage, myImage);`

  This function creates a transposed image and stores it in a destination different from the source. The `myImage` image remains unchanged, and `myTransposedImage` contains the result.

Some functions perform arithmetic or logical operations between two images. These functions require two source images of the same size. You can use either source image as your destination image, or you can supply a different image as your destination image.

The following examples show the possible combinations using the `imaqAdd()` function:

- `imaqAdd(myResultImage, myImageA, myImageB);`

  This function adds two source images (`myImageA` and `myImageB`) and stores the result in a third image (`myResultImage`). Both source images remain intact after processing.

- `imaqAdd(myImageA, myImageA, myImageB);`

  This function adds two source images and stores the result in the first source image.

- `imaqAdd(myImageA, myImageA, myImageB);`

  This function adds two source images and stores the result in the second source image.

## Image Masks

An image mask isolates parts of an image for further processing. If a function has a **mask** parameter, the function process or analysis depends on both the source image and the mask image. The image mask must be an 8-bit binary image.

Pixels in the source image are processed if corresponding pixels in the mask image have values other than zero. Figure 1-1 shows how a mask affects the output of `imaqInverse()`. Figure 1-1a shows the source image. Figure 1-1b shows the mask image. Figure 1-1c shows the inverse of the source image using the mask image. Figure 1-1d shows the inverse of the source image without the mask image.

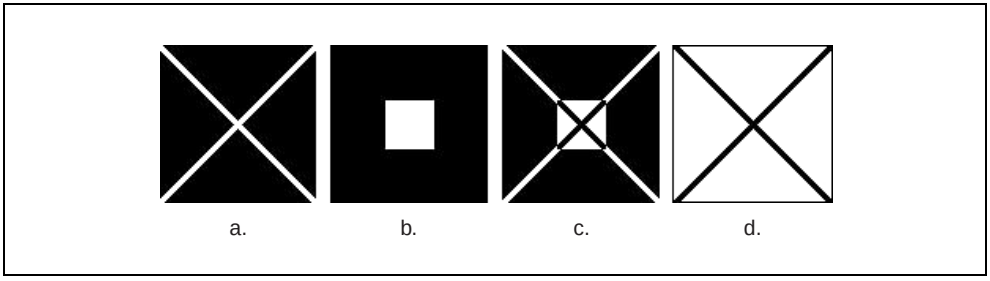



**Figure 1-1.** Image Masks

The following examples illustrate how to call a function with a `mask` parameter:

- `imaqInverse(myDest, myImage, myMask);`

  This call performs an inverse computation using a mask image, as in Figure 1-1c.
- `imaqInverse(myDest, myImage, NULL);`

  This call performs an inverse computation on the entire source image, as in Figure 1-1d.

# Processing Options

When performing morphological functions, particle analysis, and filtering, you can specify certain processing options—connectivity and structuring elements. Connectivity controls the way IMAQ Vision determines which pixels belong to a particle. Structuring elements determine which pixels belong to a neighborhood.

## Connectivity

In some functions, you can set the pixel connectivity, which specifies how IMAQ Vision determines whether two adjacent pixels are in the same particle. With connectivity-4, two pixels are part of the same particle if they are horizontally or vertically adjacent. With connectivity-8, two pixels are part of the same particle if they are horizontally, vertically, or diagonally adjacent. Figure 1-2 illustrates the two types of connectivity.

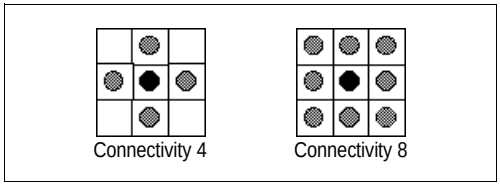



**Figure 1-2.** Connectivity Types

Figure 1-3 illustrates how connectivity-4 and connectivity-8 affect the way IMAQ Vision determines the number of particles in an image. In Figure 1-3a, IMAQ Vision considers the image as having two particles with connectivity-4. In Figure 1-3b, IMAQ Vision considers the same image as having one particle with connectivity-8. For more information about pixel connectivity in morphological operations and particle analysis, see the *IMAQ Vision User Manual*.

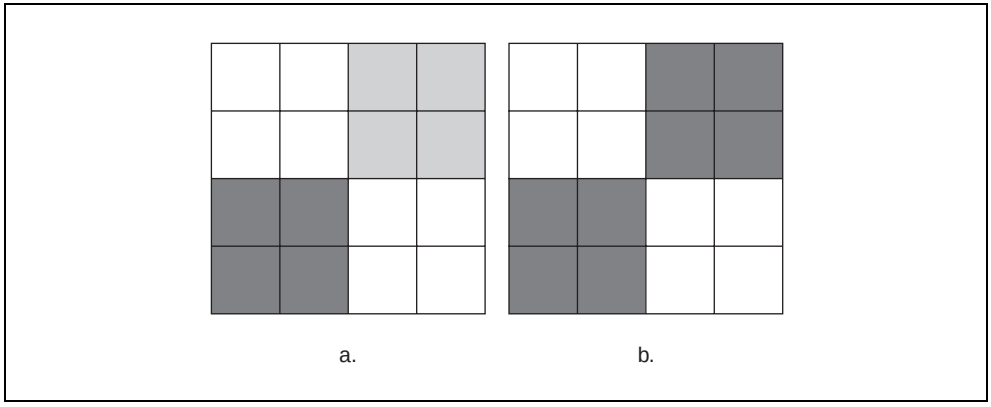



**Figure 1-3.** Example of Connectivity Processing

## Structuring Element

A structuring element determines which pixels constitute a neighborhood. Structuring elements have three parts: structuring element size, pixel frame shape, and kernel values.

A structuring element descriptor is a specific IMAQ Vision structure defined as:

```
typedef struct StructuringElement_struct {
   int matrixCols;
   int matrixRows;
   int hexa;
   int* kernel;
} StructuringElement;
```

The first two fields set the size of the structuring element itself. The third field sets the pixel frame shape. The fourth field is a pointer to the structuring element values.

## Structuring Element Size

Using structuring elements requires that the image has a border. The application of a 3 × 3 structuring element requires a minimum border size of 1. In the same way, structuring elements of 5 × 5 and 7 × 7 require a minimum border size of 2 and 3, respectively. Bigger structuring elements require corresponding increases in the image border size.

The coordinates of the central pixel (the pixel being processed) are determined as a function of the structuring element. In Figure 1-4 the coordinates of the processed pixels are (1,1), (2,2), and (3,3), respectively. The origin (0,0) is always the upper left-hand corner pixel.

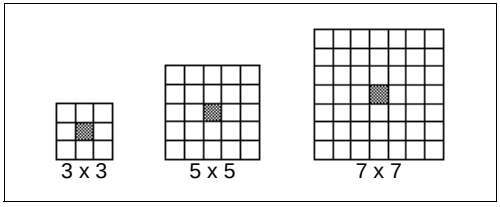



**Figure 1-4.** Structuring Element

## Pixel Frames

A digital image is a two dimensional array of pixels arranged in a rectangular grid. In image processing, this grid can have two different pixel frames: square or hexagonal. As a result, the structuring element can have either a square or hexagonal frame. Your decision to use a square or hexagonal frame affects how IMAQ Vision analyzes the image when you process it with functions that use this frame concept. The chosen pixel frame directly affects the output from morphological measurements.

By default IMAQ Vision uses the square frame (`hexa` contains `0`). Use a hexagonal frame to obtain highly precise morphological measurements. As shown below, with a hexagonal plane, the even lines shift half a pixel to the right. Therefore, the hexagonal frame places the pixels in a configuration similar to a true circle.

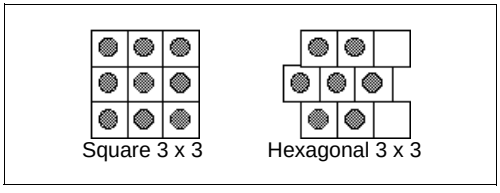



**Figure 1-5.** Square vs. Hexagonal Frames

The size of the structuring element directly determines the speed of the morphological transformation. Different results occur when you change the contents of the structuring element. You should possess a solid comprehension of morphology or spend some time learning how to use these elements before changing the standard structuring element (filled with 1s).

## Kernel

The kernel specifies which pixels are in a neighborhood. The values contained in a structuring element are either 0 or 1. These values dictate which pixels to take into account during processing. If the value in a kernel is 1, the corresponding source image pixel is part of the neighborhood. If the value in the kernel is 0, the corresponding source pixel image is not part of the neighborhood.

# 2

# Image Management

This chapter describes the Image Management functions in IMAQ Vision for LabWindows/CVI.

## Image Management Function Panels

Table 2-1 lists the Image Management functions in a tree structure. The functions in the Image Management class are grouped according to the types of operations they perform. The first-level heading in the tree is the name of the class. The second-level headings are names of function subclasses. The third-level headings are names of individual function panels. Each image management function panel represents one function call.

**Table 2-1.** Image Management Function Tree

| Class/Panel Name | Function Name |
|---|---|
| Image Management | |
| Create Image | `imaqCreateImage` |
| Pixel Manipulation | |
| Array To Complex Plane | `imaqArrayToComplexPlane` |
| Array To Image | `imaqArrayToImage` |
| Complex Plane To Array | `imaqComplexPlaneToArray` |
| Extract Color Planes | `imaqExtractColorPlanes` |
| Extract Complex Plane | `imaqExtractComplexPlane` |
| Fill Image | `imaqFillImage` |
| Get Line | `imaqGetLine` |
| Get Pixel | `imaqGetPixel` |
| Image To Array | `imaqImageToArray` |
| Replace Color Planes | `imaqReplaceColorPlanes` |
| Replace Complex Plane | `imaqReplaceComplexPlane` |
| Set Line | `imaqSetLine` |
| Set Pixel | `imaqSetPixel` |
| Image Manipulation | |
| Cast | `imaqCast` |
| Copy Rectangle | `imaqCopyRect` |
| Duplicate | `imaqDuplicate` |
| Flip | `imaqFlip` |
| Mask | `imaqMask` |
| Resample | `imaqResample` |
| Rotate | `imaqRotate` |
| Scale | `imaqScale` |

**Table 2-1.** Image Management Function Tree (Continued)

| Class/Panel Name | Function Name |
|---|---|
| Shift | `imaqShift` |
| Transpose | `imaqTranspose` |
| Image Information | |
| Get Bytes Per Pixel | `imaqGetBytesPerPixel` |
| Get Calibration Information | `imaqGetCalibrationInfo` |
| Get Image Information | `imaqGetImageInfo` |
| Get Image Size | `imaqGetImageSize` |
| Get Image Type | `imaqGetImageType` |
| Get Mask Offset | `imaqGetMaskOffset` |
| Get Pixel Address | `imaqGetPixelAddress` |
| Is Image Empty | `imaqIsImageEmpty` |
| Set Calibration Information | `imaqSetCalibrationInfo` |
| Set Image Size | `imaqSetImageSize` |
| Set Mask Offset | `imaqSetMaskOffset` |
| Drawing | |
| Draw Line On Image | `imaqDrawLineOnImage` |
| Draw Shape On Image | `imaqDrawShapeOnImage` |
| Draw Text On Image | `imaqDrawTextOnImage` |
| Clipboard | |
| Clipboard To Image | `imaqClipboardToImage` |
| Image To Clipboard | `imaqImageToClipboard` |
| Interlacing | |
| Interlace Combine | `imaqInterlaceCombine` |
| Interlace Separate | `imaqInterlaceSeparate` |
| Border | |
| Fill Border | `imaqFillBorder` |
| Get Border Size | `imaqGetBorderSize` |
| Set Border Size | `imaqSetBorderSize` |

# Subclass Descriptions

Image Management subclass descriptions are as follows:

- Pixel Manipulation functions allow you to manipulate images at the pixel level. You can use functions in the Pixel Manipulation subclass to extract image planes, replace image planes, set and return pixel values, and convert images to arrays and arrays to images.
- Image Manipulation functions allow you to manipulate images in their entirety. Functions in this subclass copy, scale, shift, and transpose images.
- Image Information functions allow you to gather information about pixels and images.
- Drawing functions allow you to draw lines, shapes, and text on images.
- Clipboard functions allow you to copy images to and from the clipboard.

- Interlacing functions allow you to combine two fields into one image frame or separate a frame into two fields.
- Border functions allow you to fill an image border with a set of values, get the size of an image border, and set the size of image border.

# imaqArrayToComplexPlane

## Usage

```
int = imaqArrayToComplexPlane(Image* dest, const Image* source, const float*
                              newPixels, ComplexPlane plane)
```

## Purpose

Replaces a plane of a complex image with the given array of pixel values. The array must be the same size as the source image.

## Image Type Supported

`IMAQ_IMAGE_COMPLEX`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **newPixels** | const float* | The two-dimensional array of pixel values. This array must be the same size as the source image. |
| **plane** | ComplexPlane | The plane to replace. Specify `IMAQ_REAL` to replace the real plane or `IMAQ_IMAGINARY` to replace the imaginary plane. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqArrayToImage

## Usage

```
int = imaqArrayToImage(Image* image, const void* array, int numCols, int
                       numRows)
```

## Purpose

Sets the pixels of an image to the values in a given array. This function resizes the image to the size of the source array.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | Image* | The image whose pixels the function sets to match the input array. |
| **array** | const void* | The two-dimensional array of pixel values. The type of the array you provide depends on the image type.<br>**Image Type** — **Array Type**<br>IMAQ_IMAGE_U8 — unsigned char<br>IMAQ_IMAGE_I16 — short<br>IMAQ_IMAGE_SGL — float<br>IMAQ_IMAGE_COMPLEX — Complex structures<br>IMAQ_IMAGE_RGB — RGBValue structures<br>IMAQ_IMAGE_HSL — HSLValue structures |
| **numCols** | int | The number of columns in the data array. |
| **numRows** | int | The number of rows in the data array. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqCast

## Usage

```
int = imaqCast(Image* dest, Image* source, ImageType type, float* lookup, int
                          shift)
```

## Purpose

Changes the type of an image. This function can perform the change directly on the source image, or it can leave the source image unchanged and instead copy the source image to a destination image and then convert the destination image. If **dest** is equal to **source**, the function changes the type of **source**. Otherwise, the function resizes **dest** to the size of **source** and then copies the pixels.

If the **source** type and the **type** parameter are the same, the function copies pixels unmodified. You can also use `imaqCopyRect()` to copy pixels without modifying them. If the **source** type and the **type** parameter are not the same, the function casts the pixel values to the new type, as Table 2-2 shows.

**Table 2-2.** Copying Source Pixels to a Destination Image

| source Type | type Parameter | Result |
|---|---|---|
| `IMAQ_IMAGE_U8` | `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL` | If you provide a lookup table, the destination pixel will have the lookup value of the source pixel. The lookup table must contain 256 elements. If you do not provide a lookup table, the function copies the source value to the destination unmodified. |
| `IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL` | `IMAQ_IMAGE_RGB32` | Each color component of the destination is set to the source value. If the source value is greater than 255, the function sets each color component to 255. If the source value is less than 0, the function sets each color component to 0. |

**Table 2-2.** Copying Source Pixels to a Destination Image (Continued)

| source Type | type Parameter | Result |
|---|---|---|
| IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL | IMAQ_IMAGE_HSL32 | The function sets the luminance component of the destination to the source value. If the source value is greater than 255, the function sets the luminance to 255. If the source value is less than 0, the function sets the luminance to 0. The function sets hue and saturation to 0. |
| IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL | IMAQ_IMAGE_COMPLEX | The function sets the real component of the destination to the source value. The function sets the imaginary component of the destination to 0. |
| IMAQ_IMAGE_I16 | IMAQ_IMAGE_U8 | The function right-shifts the source value by the given shift value (divides each source pixel value by $2^{shift}$) and stores the value in the destination. If the shifted value is greater than 255, the function sets the destination value to 255. |
| IMAQ_IMAGE_I16 | IMAQ_IMAGE_SGL | If you provide a lookup table, the destination pixel will have the lookup value of the source pixel. The lookup table must contain 65,536 elements. If you do not provide a lookup table, the function copies the source value to the destination unmodified. |
| IMAQ_IMAGE_SGL | IMAQ_IMAGE_U8, IMAQ_IMAGE_I16 | The function sets the destination value to the source value. If the source value is out of the range of the destination, the function coerces the source to the range. |
| IMAQ_IMAGE_RGB | IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL | The function sets the destination value to the average of the three color components of the source. |
| IMAQ_IMAGE_RGB | IMAQ_IMAGE_HSL | The function converts each pixel from the RGB color space to the HSL color space. |

**Table 2-2.** Copying Source Pixels to a Destination Image (Continued)

| source Type | type Parameter | Result |
|---|---|---|
| IMAQ_IMAGE_RGB | IMAQ_IMAGE_COMPLEX | The function sets the real portion of the destination value to the average of the three color components of the source, and it sets the imaginary portion of the destination to 0. |
| IMAQ_IMAGE_HSL | IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL | The function sets the destination value to the luminance component of the source value. |
| IMAQ_IMAGE_HSL | IMAQ_IMAGE_RGB | The function converts each pixel from the HSL color space to the RGB color space. |
| IMAQ_IMAGE_HSL | IMAQ_IMAGE_COMPLEX | The function sets the real portion of the destination value to the value of the luminance component of the source, and it sets the imaginary portion of the destination to 0. |
| IMAQ_IMAGE_COMPLEX | IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL | The function sets the destination value to the magnitude of the source value. |
| IMAQ_IMAGE_COMPLEX | IMAQ_IMAGE_RGB | The function sets each color component of the destination value to the magnitude of the source value. |
| IMAQ_IMAGE_COMPLEX | IMAQ_IMAGE_HSL | The function sets the luminance component of the destination value to the magnitude of the source value, and it sets the hue and saturation components to 0. |

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. Set this parameter to **source** to change the source image directly. |
| **source** | Image* | The source image. |
| **type** | ImageType | The new type for the image. |
| **lookup** | float* | An optional lookup table. Set this parameter to NULL if you do not want to use a lookup table. See Table 2-2 for a description of how the function employs the lookup table. |
| **shift** | int | The shift value for converting 16-bit images to 8-bit images. The function ignores this value for all other conversions. See Table 2-2 for a description of how the function employs the shift value. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqClipboardToImage

## Usage

```
int = imaqClipboardToImage(Image* dest, RGBValue* palette)
```

## Purpose

Copies an image from the clipboard.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_RGB

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The image into which the function copies the clipboard image. |
| **palette** | RGBValue* | An array of 256 entries that receives the palette associated with the 8-bit clipboard image. If there is no palette associated with the image, the function will fill in a gray palette. Set this parameter to NULL if you do not need the palette. |

## Return Values

**int**—On success, this function returns 1 if there was an image on the clipboard or –1 if there was no image on the clipboard. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqComplexPlaneToArray

## Usage

```
float* = imaqComplexPlaneToArray(const Image* image, ComplexPlane plane,
                                 Rect rect, int* columns, int* rows)
```

## Purpose

Extracts a plane from a complex image into a two-dimensional array.

## Image Type Supported

`IMAQ_IMAGE_COMPLEX`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image whose plane of pixel values the function places into an array. |
| **plane** | ComplexPlane | The plane to extract. Set this parameter to `IMAQ_REAL`, `IMAQ_IMAGINARY`, `IMAQ_MAGNITUDE`, or `IMAQ_PHASE`. |
| **rect** | Rect | Specifies a rectangular region of the image to return. Set this parameter to `IMAQ_NO_RECT` to return the specified plane of the entire image. |
| **columns** | int* | On return, the number of columns in the returned array. Set this parameter to NULL if you do not need this information. |
| **rows** | int* | On return, the number of rows in the returned array. Set this parameter to NULL if you do not need this information. |

## Return Value

**float***—On success, this function returns a two-dimensional array of values. These values are the pixel values of the extracted plane. On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with the array, dispose of it by calling `imaqDispose()`.

# imaqCopyRect

## Usage

```
int = imaqCopyRect(Image* dest, const Image* source, Rect rect, Point dest)
```

## Purpose

Copies an area of one image into another image. You can copy the source area to a new destination image or to another area in the source image. The source and destination images must be of the same type. If the source area is larger than the destination area, the source area will be clipped. The size of the destination image and pixels outside the destination area remain unchanged. To make a duplicate of an image, including border and calibration information, use `imaqDuplicate()`.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_COMPLEX`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **rect** | Rect | The area of the source to copy into the destination. Set this parameter to `IMAQ_NO_RECT` to copy the whole source image to the destination. |
| **dest** | Point | The coordinates of the top-left pixel in the destination image where the function copies the source area. This location can be anywhere on the image or on the border of the image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqCreateImage

## Usage

```
Image* = imaqCreateImage(ImageType type, int borderSize)
```

## Purpose

Creates an image. The created image will be $0 \times 0$ pixels in size. To change the image size, use `imaqSetImageSize()`.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_COMPLEX`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **type** | ImageType | The type of image to create. |
| **borderSize** | int | The size of the image border. For more information about borders, see the *IMAQ Vision User Manual*. |

## Return Value

**Image***—On success, this function returns the created image. On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with the created image, dispose of it by calling `imaqDispose()`.

# imaqDrawLineOnImage

## Usage

```
int = imaqDrawLineOnImage(Image* dest, const Image* source, DrawMode mode,
                          Point start, Point end, float newPixelValue)
```

## Purpose

Draws a line on an image.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **mode** | DrawMode | The method that the function uses to draw a line. Set this parameter to IMAQ_DRAW_VALUE to draw a line with the specified value or IMAQ_DRAW_INVERT to draw a line by inverting underlying pixel values. |
| **start** | Point | The coordinate location of the starting point of the line. |
| **end** | Point | The coordinate location of the ending point of the line. |
| **newPixelValue** | float | If you set **mode** to IMAQ_DRAW_VALUE, **newPixelValue** sets the pixel value in which the function draws the line. If you set **mode** to IMAQ_DRAW_INVERT, the function ignores this parameter. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqDrawShapeOnImage

## Usage

```
int = imaqDrawShapeOnImage(Image* dest, const Image* source, Rect rect,
                           DrawMode mode, ShapeMode shape, float
                           newPixelValue)
```

## Purpose

Draws a shape on an image.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **rect** | Rect | The bounding rectangle of the shape. |
| **mode** | DrawMode | The method that the function uses to draw the shape. The following options are valid:<br>IMAQ_DRAW_VALUE—Draws the boundary of the shape with the specified pixel value.<br>IMAQ_DRAW_INVERT—Inverts the pixel values of the boundary of the shape.<br>IMAQ_PAINT_VALUE—Fills the shape with the given pixel value.<br>IMAQ_PAINT_INVERT—Inverts the pixel values of the shape. |
| **shape** | ShapeMode | The shape to draw. The following options are valid:<br>IMAQ_SHAPE_RECT—Draws a rectangle.<br>IMAQ_SHAPE_OVAL—Draws an oval. |
| **newPixelValue** | float | If you set **mode** to IMAQ_DRAW_VALUE or IMAQ_PAINT_VALUE, **newPixelValue** sets the pixel value that the function uses to draw a shape |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqDrawTextOnImage

## Usage

```
int = imaqDrawTextOnImage(Image* dest, const Image* source, Point coord,
                          const char* text, const DrawTextOptions* options,
                          int* fontNameUsed)
```

## Purpose

Draws text on an image.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_RGB`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **coord** | Point | The coordinates of the text reference point. |
| **text** | const char* | The text that the function draws. This parameter is required and cannot be NULL. |
| **options** | const DrawTextOptions* | The method that the function uses to draw text. |
| **fontNameUsed** | int* | On return, this parameter equals TRUE if the function located and used `fontName` in **options**. This parameter equals FALSE if the function could not locate the `fontName` in **options**. When FALSE, the function uses the font Arial. Set this parameter to NULL if you do not need this information. |

## Parameter Discussion

**options**—A `DrawTextOptions` structure contains the following elements:

- `fontName`—The name of the font to use. This parameter is limited to 32 characters.
- `fontSize`—The size of the font.
- `bold`—Set this parameter to TRUE to bold the text.
- `italic`—Set this parameter to TRUE to italicize the text.
- `underline`—Set this parameter to TRUE to underline the text.

- strikeout—Set this parameter to TRUE to strikeout the text.
- textAlignment—Sets the alignment of the text. The following options are valid:
  - IMAQ_LEFT—Left aligns the text at the reference point.
  - IMAQ_CENTER—Centers the text around the reference point.
  - IMAQ_RIGHT—Right aligns the text at the reference point.
- fontColor—Sets the color of the font. The following options are valid:
  - IMAQ_WHITE—Draws text in white.
  - IMAQ_BLACK—Draws text in black.
  - IMAQ_INVERT—Inverts the text pixels.
  - IMAQ_BLACK_ON_WHITE—Draws text in black with a white background.
  - IMAQ_WHITE_ON_BLACK—Draws text in white with a black background.

Set the **options** parameter to NULL to use the default options, as follows:

- fontName—Arial
- fontSize—12
- bold—FALSE
- italic—FALSE
- underline—FALSE
- strikeout—FALSE
- textAlignment—IMAQ_LEFT
- fontColor—IMAQ_WHITE

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqDuplicate

## Usage

```
int = imaqDuplicate(Image* dest, const Image* source)
```

## Purpose

Copies the source image to the destination image, including the border size and calibration information. To copy an area of one image to an area of another image, use `imaqCopyRect()`.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_COMPLEX`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The duplicated image. |
| **source** | const Image* | The image to copy. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqExtractColorPlanes

## Usage

```
int = imaqExtractColorPlanes(const Image* image, ColorMode mode, Image*
                             plane1, Image* plane2, Image* plane3)
```

## Purpose

Extracts the individual color planes from a color image. The plane you extract is independent from the type of the image. For example, you can extract the hue plane from an RGB image or the green plane from an HSL image.

## Image Types Supported

`IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The source image from which the function extracts the planes. |
| **mode** | ColorMode | The color space from which the function extracts the planes. Valid options are `IMAQ_RGB`, `IMAQ_HSL`, `IMAQ_HSV`, and `IMAQ_HSI`. For more information about color spaces, see the *IMAQ Vision User Manual*. |
| **plane1** | Image* | On return, the first extracted plane. The data contained in **plane1** depends on the **mode**, as follows:<br>**Mode** — **Plane**<br>`IMAQ_RGB` — Red<br>`IMAQ_HSL` — Hue<br>`IMAQ_HSV` — Hue<br>`IMAQ_HSI` — Hue<br>Set this parameter to NULL if you do not need this information. |

| Name | Type | Description |
|---|---|---|
| **plane2** | Image* | On return, the second extracted plane. The data contained in **plane2** depends on the **mode**, as follows:<br>**Mode** — **Plane**<br>IMAQ_RGB — Green<br>IMAQ_HSL — Saturation<br>IMAQ_HSV — Saturation<br>IMAQ_HSI — Saturation<br>Set this parameter to NULL if you do not need this information. |
| **plane3** | Image* | On return, the third plane. The data contained in **plane3** depends on the **mode**, as follows:<br>**Mode** — **Plane**<br>IMAQ_RGB — Blue<br>IMAQ_HSL — Luminance<br>IMAQ_HSV — Value<br>IMAQ_HSI — Intensity<br>Set this parameter to NULL if you do not need this information. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqExtractComplexPlane

## Usage

```
int = imaqExtractComplexPlane(Image* dest, const Image* source, ComplexPlane
                              plane)
```

## Purpose

Extracts a plane from a complex image and places the plane into another image.

## Image Type Supported

`IMAQ_IMAGE_COMPLEX`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image, which must be an `IMAQ_IMAGE_SGL` image. |
| **source** | const Image* | The image whose plane the function extracts. |
| **plane** | ComplexPlane | The plane to extract. Set this parameter to `IMAQ_REAL` to extract the real plane or `IMAQ_IMAGINARY` to extract the imaginary plane. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqFillBorder

## Usage

```
int = imaqFillBorder(Image* image, BorderMethod method)
```

## Purpose

Modifies the border of an image.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | Image* | The image whose border the function modifies. |
| **method** | BorderMethod | The method by which the function modifies the border. The following options are valid:<br>IMAQ_BORDER_MIRROR—Symmetrically copies pixel values from the image into the border.<br>IMAQ_BORDER_COPY—Copies the value of the pixel closest to the edge of the image into the border.<br>IMAQ_BORDER_CLEAR—Sets all pixels in the border to 0. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqFillImage

## Usage

```
int = imaqFillImage(Image* image, PixelValue value, const Image* mask)
```

## Purpose

Sets each pixel in an image to a specified value.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | Image* | The image whose pixel values the function overwrites with the given value. |
| **value** | PixelValue | The value with which the function fills the image pixels. The field of the PixelValue union you set depends on the image type, as follows:<br>**Image Type** — **PixelValue Field**<br>IMAQ_IMAGE_U8 — grayscale<br>IMAQ_IMAGE_I16 — grayscale<br>IMAQ_IMAGE_SGL — grayscale<br>IMAQ_IMAGE_COMPLEX — complex<br>IMAQ_IMAGE_HSL — hsl<br>IMAQ_IMAGE_RGB — rgb |
| **mask** | const Image* | A mask image. This image must be an IMAQ_IMAGE_U8 image. The function sets only those source pixels whose corresponding mask pixels are non-zero to the specified value. Set this parameter to NULL if you want the function to set every pixel in the source image to the specified value. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqFlip

## Usage

```
int = imaqFlip(Image* dest, const Image* source, FlipAxis axis)
```

## Purpose

Flips an image over an axis.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The image that the function flips over an axis. |
| **axis** | FlipAxis | The axis to flip the image over. The following options are valid:<br>IMAQ_HORIZONTAL_AXIS—Flips the image over the central horizontal axis.<br>IMAQ_VERTICAL_AXIS—Flips the image over the central vertical axis.<br>IMAQ_CENTER_AXIS—Flips the image over both the central vertical and horizontal axes.<br>IMAQ_DIAG_L_TO_R_AXIS—Flips the image over an axis from the upper left corner to lower right corner.<br>IMAQ_DIAG_R_TO_L_AXIS—Flips the image over an axis from the upper right corner to lower left corner. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqGetBorderSize

## Usage

```
int = imaqGetBorderSize(const Image* image, int* borderSize)
```

## Purpose

Returns the border size of the given image.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image whose border size the function queries. |
| **borderSize** | int* | On return, the border size of the image. This parameter is required and cannot be NULL. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetBytesPerPixel

## Usage

```
int = imaqGetBytesPerPixel(const Image* image, int* byteCount)
```

## Purpose

Returns the number of bytes that a single pixel occupies in the given image.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_COMPLEX`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image whose bytes per pixel the function queries. |
| **byteCount** | int* | On return, the number of bytes per pixel. This parameter is required and cannot be NULL. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetCalibrationInfo

## Usage

```
int = imaqGetCalibrationInfo(const Image* image, CalibrationUnit* unit,
                             float* xDistance, float* yDistance)
```

## Purpose

Returns the calibration information of an image.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_COMPLEX`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image whose calibration information the function queries. |
| **unit** | CalibrationUnit* | On return, the unit of measure that you specified in `imaqSetCalibrationInfo()`. Set this parameter to NULL if you do not need the unit information. |
| **xDistance** | float* | On return, the distance between two adjacent pixels in the *x* direction. This value is in the unit specified in **unit**. Set this parameter to NULL if you do not need the *x* distance information. |
| **yDistance** | float* | On return, the distance between two adjacent pixels in the *y* direction. This value is in the unit specified in **unit**. Set this parameter to NULL if you do not need the *y* distance information. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetImageInfo

## Usage

```
int = imaqGetImageInfo(const Image* image, ImageInfo* info)
```

## Purpose

Returns the size, border, type, calibration, and memory layout of an image.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image whose information the function returns. |
| **info** | ImageInfo* | On return, the information about the image. |

## Parameter Discussion

**info**—The ImageInfo structure consists of the following elements:

- imageUnit—If you set calibration information with imaqSetCalibrationInfo(), imageUnit is the calibration unit.
- stepX—If you set calibration information with imaqSetCalibrationInfo(), stepX is the distance, in the calibration unit, between two pixels in the *x* direction.
- stepY—If you set calibration information with imaqSetCalibrationInfo(), stepY is the distance, in the calibration unit, between two pixels in the *y* direction.
- imageType—The type of the image.
- xRes—The number of columns in the image.
- yRes—The number of rows in the image.
- xOffset—If you set mask offset information with imaqSetMaskOffset(), xOffset is the offset of the mask origin in the *x* direction.
- yOffset—If you set mask offset information with imaqSetMaskOffset(), yOffset is the offset of the mask origin in the *y* direction.
- border—The number of border pixels around the image.
- pixelsPerLine—The number of pixels stored for each line of the image. This value may be larger than xRes.
- imageStart—A pointer to pixel (0,0).

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetImageSize

## Usage

```
int = imaqGetImageSize(const Image* image, int* width, int* height)
```

## Purpose

Returns the size of a given image.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_COMPLEX`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image whose size the function queries. |
| **width** | int* | On return, the width of the image. Set this parameter to NULL if you do not need this information. |
| **height** | int* | On return, the height of the image. Set this parameter to NULL if you do not need this information. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetImageType

## Usage

```
int = imaqGetImageType(const Image* image, ImageType* type)
```

## Purpose

Returns the type of the given image.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image whose type the function queries. |
| **type** | ImageType* | On return, the type of the image. This parameter is required and cannot be NULL. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqGetLine

## Usage

```
void* = imaqGetLine(const Image* image, Point start, Point end, int*
                    numPoints)
```

## Purpose

Returns the pixel values along a given line in an image. If the starting or ending point of the line is outside the image, the line clips at the last visible pixel.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image containing a line whose pixels the function returns. |
| **start** | Point | The coordinate location of the starting point of the line. |
| **end** | Point | The coordinate location of the ending point of the line. |
| **numPoints** | int* | The number of elements in the returned array. Set this parameter to NULL if you do not need this information. |

## Return Value

**void***—On success, this function returns the values of the pixels along the given line in the image. The type of array the function returns depends on the image type, as follows:

| Image Type | Array Type |
|---|---|
| IMAQ_IMAGE_U8 | unsigned char |
| IMAQ_IMAGE_I16 | short |
| IMAQ_IMAGE_SGL | float |
| IMAQ_IMAGE_RGB | RGBValue structures |
| IMAQ_IMAGE_HSL | HSLValue structures |

On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`.When you are finished with the array, dispose of it by calling `imaqDispose()`.

# imaqGetMaskOffset

## Usage

```
int = imaqGetMaskOffset(const Image* image, Point* offset)
```

## Purpose

Retrieves the point in the source image at which the function places the (0,0) pixel of the mask image, as set by `imaqSetMaskOffset()`.

## Image Type Supported

`IMAQ_IMAGE_U8`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The mask image whose offset the function retrieves. |
| **offset** | Point* | On return, the coordinates where the function applies the mask. This parameter is required and cannot be NULL. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetPixel

## Usage

```
int = imaqGetPixel(const Image* image, Point pixel, PixelValue* value)
```

## Purpose

Returns the value of a pixel within an image.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image whose pixel value the function queries. |
| **pixel** | Point | The coordinates of the pixel that the function queries. |
| **value** | PixelValue* | On return, the value of the image pixel. The field of the PixelValue union that the function sets depends on the image type, as follows:<br>**Image Types** — **PixelValue Field**<br>IMAQ_IMAGE_U8 — grayscale<br>IMAQ_IMAGE_I16 — grayscale<br>IMAQ_IMAGE_SGL — grayscale<br>IMAQ_IMAGE_COMPLEX — complex<br>IMAQ_IMAGE_HSL — hsl<br>IMAQ_IMAGE_RGB — rgb |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqGetPixelAddress

## Usage

```
void* = imaqGetPixelAddress(const Image* image, Point pixel)
```

## Purpose

Returns the address of a given pixel in an image. If the requested pixel location is outside of the image, the function fails and returns NULL.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image containing the requested pixel. |
| **pixel** | Point | The coordinates of the pixel whose pointer the function retrieves. |

## Return Value

**void***—On success, this function returns a pointer to the requested pixel in the image. The type of the pointer the function returns depends on the type of the image, as follows:

| Image Type | Pointer Type |
|---|---|
| IMAQ_IMAGE_U8 | unsigned char |
| IMAQ_IMAGE_I16 | short |
| IMAQ_IMAGE_SGL | float |
| IMAQ_IMAGE_COMPLEX | Complex structures |
| IMAQ_IMAGE_RGB | RGBValue structures |
| IMAQ_IMAGE_HSL | HSLValue structures |

On failure, this function returns NULL. To get extended error information, call imaqGetLastError().

# imaqImageToArray

## Usage

```
void* = imaqImageToArray(const Image* image, Rect rect, int* columns, int*
                         rows)
```

## Purpose

Creates a two-dimensional array from an image.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image from which the function makes the array. |
| **rect** | Rect | Specifies a rectangular region of the image to return. Set this parameter to IMAQ_NO_RECT if you want the function to return the whole image. |
| **columns** | int* | The number of columns in the returned array. Set this parameter to NULL if you do not need this information. |
| **rows** | int* | The number of rows in the returned array. Set this parameter to NULL if you do not need this information. |

## Return Value

**void***—On success, this function returns a two-dimensional array. The type of the returned array depends on the image type, as follows:

| Image Type | Pointer Type |
|---|---|
| IMAQ_IMAGE_U8 | unsigned char |
| IMAQ_IMAGE_I16 | short |
| IMAQ_IMAGE_SGL | float |
| IMAQ_IMAGE_COMPLEX | Complex structures |
| IMAQ_IMAGE_RGB | RGBValue structures |
| IMAQ_IMAGE_HSL | HSLValue structures |

On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with the array, dispose of it by calling `imaqDispose()`.

# imaqImageToClipboard

## Usage

```
int = imaqImageToClipboard(const Image* image, const RGBValue* palette)
```

## Purpose

Copies an image onto the clipboard.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_RGB`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image to copy onto the clipboard |
| **palette** | const RGBValue* | An optional palette to associate with 8-bit images. If this parameter is not NULL, it must point to an array of 256 colors, which represent the color palette that the function associates with the image. If this parameter is NULL, the function associates a grayscale palette with the image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqInterlaceCombine

## Usage

```
int = imaqInterlaceCombine(Image* frame, const Image* odd, const Image* even)
```

## Purpose

Combines two field images to create a single frame image.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **frame** | Image* | On return, the combined image. |
| **odd** | const Image* | The odd field. |
| **even** | const Image* | The even field. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqInterlaceSeparate

## Usage

```
int = imaqInterlaceSeparate(const Image* frame, Image* odd, Image* even)
```

## Purpose

Separates a frame image into two field images.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **frame** | const Image* | The frame image that the function separates into odd and even fields. |
| **odd** | Image* | The image into which the function places the odd field of the frame area. Set this parameter to NULL if you do not need the odd field. |
| **even** | Image* | The image into which the function places the even field of the frame area. Set this parameter to NULL if you do not need the even field. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqIsImageEmpty

## Usage

```
int = imaqIsImageEmpty(const Image* image, int* empty)
```

## Purpose

Tests to see if the supplied image is empty. An empty image is an image that only contains pixels with a value equal to zero. Use this function in conjunction with `imaqCompare()` and `imaqCompareConstant()` to see if the compare operation cleared all of the pixels in an image.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image that the function checks for emptiness. |
| **empty** | int* | On return, this parameter equals TRUE if the image is empty and FALSE if the image contains pixels with values other than 0. This parameter is required and cannot be NULL. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqMask

## Usage

```
int = imaqMask(Image* dest, const Image* source, const Image* mask)
```

## Purpose

Copies the source image to the destination image in the following manner: If a pixel in the mask has a value of 0, the function sets the corresponding source pixel to 0. Otherwise the function copies the corresponding source pixel to the destination image.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_COMPLEX`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **mask** | const Image* | The mask image. This image must be an `IMAQ_IMAGE_U8` image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqReplaceColorPlanes

## Usage

```
int = imaqReplaceColorPlanes(Image* dest, const Image* source, ColorMode
                             mode, const Image* plane1, const Image* plane2,
                             const Image* plane3)
```

## Purpose

Replaces one or more of the color planes of a color image. The plane you replace is independent of the image type. For example, you can replace the hue plane of an RGB image or the green plane of an HSL image.

## Image Types Supported

`IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **mode** | ColorMode | The color space in which the function replaces planes. Valid options are `IMAQ_RGB`, `IMAQ_HSL`, `IMAQ_HSV`, and `IMAQ_HSI`. For more information on color spaces, see the *IMAQ Vision User Manual*. |
| **plane1** | const Image* | The first plane of replacement data. Set this parameter to NULL if you do not want to change the first plane of the source image. The data contained here depends on **mode**, as follows:<br>**Mode** — **Plane**<br>`IMAQ_RGB` — Red<br>`IMAQ_HSL` — Hue<br>`IMAQ_HSV` — Hue<br>`IMAQ_HSI` — Hue |

| Name | Type | Description |
|---|---|---|
| **plane2** | const Image* | The second plane of replacement data. Set this parameter to NULL if you do not want to change the second plane of the source image. The data contained here depends on **mode**, as follows:<br>**Mode** — **Plane**<br>IMAQ_RGB — Green<br>IMAQ_HSL — Saturation<br>IMAQ_HSV — Saturation<br>IMAQ_HSI — Saturation |
| **plane3** | const Image* | The third plane of replacement data. Set this parameter to NULL if you do not want to change the third plane of the source image. The data contained here depends on mode, as follows:<br>**Mode** — **Plane**<br>IMAQ_RGB — Blue<br>IMAQ_HSL — Luminance<br>IMAQ_HSV — Value<br>IMAQ_HSI — Intensity |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqReplaceComplexPlane

## Usage

```
int = imaqReplaceComplexPlane(Image* dest, const Image* source, const Image*
                              newValues, ComplexPlane plane)
```

## Purpose

Replaces a plane of a complex image with the pixel values from a given image.

## Image Type Supported

`IMAQ_IMAGE_COMPLEX`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The image whose data the function modifies. |
| **newValues** | const Image* | The image containing the replacement values. This image may be `IMAQ_IMAGE_U8`, `IMAQ_IMAGE_U16`, or `IMAQ_IMAGE_SGL`. |
| **plane** | ComplexPlane | The complex image plane to replace. Set this value to `IMAQ_REAL` or `IMAQ_IMAGINARY`. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqResample

## Usage

```
int = imaqResample(Image* dest, const Image* source, int newWidth, int
                           newHeight, InterpolationMethod method, Rect rect)
```

## Purpose

Resizes an image to a given resolution. The source image and destination image must be the same image type. After execution, the size destination is **newWidth** × **newHeight**. For fast zero-order scaling, use `imaqScale()`.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The image into which the function places the resampled data. This may be the same as source. |
| **source** | const Image* | The image to resample. |
| **newWidth** | int | The width of the resampled area. |
| **newHeight** | int | The height of the resampled area. |
| **method** | InterpolationMethod | The method of interpolation. The following options are valid:<br>`IMAQ_ZERO_ORDER`<br>`IMAQ_BILINEAR`<br>`IMAQ_QUADRATIC`<br>`IMAQ_CUBIC_SPLINE` |
| **rect** | Rect | Specifies an area of the source image to resample. Set this parameter to `IMAQ_NO_RECT` to resample the entire image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqRotate

## Usage

```
int = imaqRotate(Image* dest, const Image* source, float angle, PixelValue
                         fill, InterpolationMethod method)
```

## Purpose

Rotates an image counterclockwise.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_RGB

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The image to rotate. |
| **angle** | float | The angle, in degrees, to rotate the image. |
| **fill** | PixelValue | The value with which the function fills the image pixels not covered by the rotated image. The field of the PixelValue union you set depends on the image type, as follows:<br>**Image Types** — **PixelValue Field**<br>IMAQ_IMAGE_U8 — grayscale<br>IMAQ_IMAGE_I16 — grayscale<br>IMAQ_IMAGE_SGL — grayscale<br>IMAQ_IMAGE_RGB — rgb |
| **method** | InterpolationMethod | The method of interpolation. The valid interpolation methods for rotation are IMAQ_ZERO_ORDER and IMAQ_BILINEAR. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqScale

## Usage

```
int = imaqScale(Image* dest, const Image* source, int xScale, int yScale,
                        ScalingMode scaleMode, Rect rect)
```

## Purpose

Scales an image or area of an image. The source image and destination image must be the same image type. This function makes an image larger by duplicating pixels, and it makes an image smaller by subsampling pixels. For more sophisticated scaling techniques, use `imaqResample()`.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image to scale. |
| **xScale** | int | The scaling factor in the *x* direction. If you set **scaleMode** to `IMAQ_SCALE_LARGER`, **xScale** is a multiplicative factor, meaning the function duplicates each source pixel **xScale** times. If you set **scaleMode** to `IMAQ_SCALE_SMALLER`, **xScale** is a divisory factor, meaning the function takes one pixel for every **xScale** pixels. |
| **yScale** | int | The scaling factor in the *y* direction. If you set **scaleMode** to `IMAQ_SCALE_LARGER`, **yScale** is a multiplicative factor, meaning the function duplicates each source pixel **yScale** times. If you set **scaleMode** to `IMAQ_SCALE_SMALLER`, **yScale** is a divisory factor, meaning the function takes one pixel for every **yScale** pixels. |
| **scaleMode** | ScalingMode | The scaling mode. Set this parameter to `IMAQ_SCALE_LARGER` to duplicate pixels or `IMAQ_SCALE_SMALLER` to subsample pixels. |
| **rect** | Rect | Specifies the rectangular region of the source image to scale. Set this parameter to `IMAQ_NO_RECT` to scale the whole image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetBorderSize

## Usage

```
int = imaqSetBorderSize(Image* image, int size)
```

## Purpose

Sets the border size of an image. This operation preserves image pixels.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | Image* | The image whose border size the function sets. |
| **size** | int | The new border size. Valid border sizes range from 0–50. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetCalibrationInfo

## Usage

```
int = imaqSetCalibrationInfo(Image* image, CalibrationUnit unit, float
                             xDistance, float yDistance)
```

## Purpose

Sets the physical calibration attributes of an image.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | Image* | The image whose calibration information the function sets. |
| **unit** | CalibrationUnit | The unit of measure that the function uses to calibrate the image. |
| **xDistance** | float | The distance in the *x* direction between two adjacent pixels in units specified by **unit**. |
| **yDistance** | float | The distance in the *y* direction between to adjacent pixels in units specified by **unit**. |

## Parameter Discussion

**unit**—The following options are valid: IMAQ_UNDEFINED, IMAQ_ANGSTROM, IMAQ_MICROMETER, IMAQ_MILLIMETER, IMAQ_CENTIMETER, IMAQ_METER, IMAQ_KILOMETER, IMAQ_MICROINCH, IMAQ_INCH, IMAQ_FOOT, IMAQ_NAUTICMILE, IMAQ_GROUNDMILE.

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqSetImageSize

## Usage

```
int = imaqSetImageSize(Image* image, int width, int height)
```

## Purpose

Sets the size of an image. The original pixels are not transferred to the new image. To resize the image and retain the original information, use `imaqScale()` or `imaqResample()`.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_COMPLEX`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | Image* | The image to resize. |
| **width** | int | The new width of the image. |
| **height** | int | The new height of the image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetLine

## Usage

```
int = imaqSetLine(Image* image, const void* array, int arraySize, Point
                  start, Point end)
```

## Purpose

Sets the pixel values along a line in an image.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | Image* | The image whose line pixel values the function modifies. |
| **array** | const void* | The one-dimensional array of pixel values that the function uses to replace the values in the line. The type of array you provide depends on the image type, as follows:<br>**Image Type** — **Array Type**<br>`IMAQ_IMAGE_U8` — unsigned char<br>`IMAQ_IMAGE_I16` — short<br>`IMAQ_IMAGE_SGL` — float<br>`IMAQ_IMAGE_RGB` — RGBValue structures<br>`IMAQ_IMAGE_HSL` — HSLValue structures |
| **arraySize** | int | The number of pixels in the array. |
| **start** | Point | The coordinate location of the starting point of the line. |
| **end** | Point | The coordinate location of the ending point of the line. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetMaskOffset

## Usage

```
int = imaqSetMaskOffset(Image* image, Point offset)
```

## Purpose

When the given image is used as a mask, sets the location in the source image at which the function places the (0,0) pixel of the mask image.

## Image Type Supported

`IMAQ_IMAGE_U8`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | Image* | The mask image whose offset the function sets. |
| **offset** | Point | The coordinates where the function applies the mask. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetPixel

## Usage

```
int = imaqSetPixel(Image* image, Point coord, PixelValue value)
```

## Purpose

Sets the value of a pixel within an image.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_COMPLEX`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | Image* | The image whose pixel value the function sets. |
| **coord** | Point | The coordinates of the pixel the function sets. |
| **value** | PixelValue | The value to which the function sets the image pixel. The field of the PixelValue union you set depends on the image type, as follows:<br>**Image Types** — **PixelValue Field**<br>`IMAQ_IMAGE_U8` — `grayscale`<br>`IMAQ_IMAGE_I16` — `grayscale`<br>`IMAQ_IMAGE_SGL` — `grayscale`<br>`IMAQ_IMAGE_COMPLEX` — `complex`<br>`IMAQ_IMAGE_HSL` — `hsl`<br>`IMAQ_IMAGE_RGB` — `rgb` |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqShift

## Usage

```
int = imaqShift(Image* dest, const Image* source, int shiftX, int shiftY,
                          PixelValue fill)
```

## Purpose

Shifts an image.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The image to shift. |
| **shiftX** | int | Specifies how many pixels to the right to shift the image. |
| **shiftY** | int | Specifies how many pixels down to shift the image. |
| **fill** | PixelValue | The value with which the function fills the uncovered image pixels. The field of the PixelValue union you set depends on the image type, as follows:<br>**Image Types** — **PixelValue Field**<br>IMAQ_IMAGE_U8 — grayscale<br>IMAQ_IMAGE_I16 — grayscale<br>IMAQ_IMAGE_SGL — grayscale<br>IMAQ_IMAGE_HSL — hsl<br>IMAQ_IMAGE_RGB — rgb |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqTranspose

## Usage

```
int = imaqTranspose(Image* dest, const Image* source)
```

## Purpose

Transposes an image.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_COMPLEX`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The image to transpose. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# 3

# Memory Management

This chapter describes the Memory Management function in IMAQ Vision for LabWindows/CVI. The Memory Management function, `imaqDispose()`, deletes images, ROIs, arrays, and reports and frees the space they occupied in memory.

## Memory Management Function Panel

Table 3-1 lists the Memory Management function in a tree structure. The Memory Management function panel represents one function.

**Table 3-1.** Memory Management

| Class/Panel Name | Function Name |
|---|---|
| Memory Management | |
| Dispose | `imaqDispose` |

# imaqDispose

## Usage

```
int = imaqDispose(void* object)
```

## Purpose

Cleans up resources associated with images, ROIs, arrays, and reports that you no longer need. After you dispose something, you can no longer use it.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_COMPLEX`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **object** | void* | The image, ROI, array, or report whose memory you want to free. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# 4

# Error Management

This chapter describes the Error Management functions in IMAQ Vision for LabWindows/CVI. Error Management functions clear pending errors, return the last error, return the function in which the last error occurred, and set the error.

## Image Management Function Panels

Table 4-1 lists the Error Management functions in a tree structure. The first-level heading in the tree is the name of the class. The second-level headings are names of the individual function panels. Each error management function panel generates one function call.

**Table 4-1.** Error Management

| **Class/Panel Name** | **Function Name** |
|---|---|
| Error Management | |
| Clear Error | `imaqClearError` |
| Get Error Text | `imaqGetErrorText` |
| Get Last Error | `imaqGetLastError` |
| Get Last Error Function | `imaqGetLastErrorFunc` |
| Set Error | `imaqSetError` |

# imaqClearError

## Usage

```
int = imaqClearError()
```

## Purpose

Sets the IMAQ Vision error status for the current thread to ERR_SUCCESS.

## Return Value

**int**—This function returns a non-zero value.

# imaqGetErrorText

## Usage

```
char* = imaqGetErrorText(int errorCode)
```

## Purpose

Returns the error text corresponding to an error code. The error text is a description of what the error code signifies.

## Parameters

| Name | Type | Description |
|---|---|---|
| **errorCode** | int | The error code whose error text the function returns. You can obtain this error code by calling `imaqGetLastError()`. |

## Return Value

**char***—This function returns the error text corresponding to the error code input. This function returns `Unknown Error` if no error text corresponds to the error code you specified. When you are finished with this string, dispose of it by calling `imaqDispose()`.

# imaqGetLastError

## Usage

```
int = imaqGetLastError()
```

## Purpose

Returns the error code of the last IMAQ Vision function executed in the calling thread. For a complete list of possible errors, see Appendix A, *Error Codes*.

## Return Value

**int**—This function returns the last error code. This function returns `ERR_SUCCESS` if there is no pending error.

# imaqGetLastErrorFunc

## Usage

```
const char* = imaqGetLastErrorFunc()
```

## Purpose

Returns the name of the function in which the last error occurred.

## Return Value

**const char***—This function returns the name of the last function that failed. The function returns an empty string if there is no pending error. When you are finished with this information, dispose of the string by calling `imaqDispose()`.

# imaqSetError

## Usage

```
int = imaqSetError(int code, const char* function)
```

## Purpose

Sets the current error.

## Parameters

| Name | Type | Description |
|---|---|---|
| **code** | int | The code of the error to set. |
| **function** | const char* | The name of the function in which the error occurred. Set this parameter to NULL to record no function. |

## Return Value

**int**—The error code set.

# 5

# Acquisition

This chapter describes the Acquisition functions in IMAQ Vision for LabWindows/CVI. Functions in this chapter require NI-IMAQ 2.2 or higher. Acquisition functions let you perform common acquisition tasks, such as ring acquisitions, sequence acquisitions, and grabs, directly into an IMAQ Vision image. To perform more advanced acquisitions, such as triggered acquisitions, see the *NI-IMAQ Function Reference Manual*. You can use the Acquisition functions with the Signal I/O functions described in the *NI-IMAQ Function Reference Manual*.

## Acquisition Function Panels

Table 5-1 lists the Acquisition functions in a tree structure. The first-level heading in the tree is the name of the class. The second-level headings are names of the individual function panels. Each acquisition function panel represents one function.

**Table 5-1.** Acquisition Function Tree

| Class/Panel Name | Function Name |
|---|---|
| Acquisition | |
| Copy From Ring | `imaqCopyFromRing` |
| Easy Acquire | `imaqEasyAcquire` |
| Extract From Ring | `imaqExtractFromRing` |
| Grab | `imaqGrab` |
| Release Image | `imaqReleaseImage` |
| Setup Grab | `imaqSetupGrab` |
| Setup Ring | `imaqSetupRing` |
| Setup Sequence | `imaqSetupSequence` |
| Snap | `imaqSnap` |
| Start Acquisition | `imaqStartAcquisition` |
| Stop Acquisition | `imaqStopAcquisition` |

# imaqCopyFromRing

## Usage

```
Image* = imaqCopyFromRing(SESSION_ID sessionID, Image* image, int
                          imageToCopy, int* imageNumber, Rect rect)
```

## Purpose

Copies an area of a buffer to a user-specified image. This function is useful for ring acquisitions if you do not want to extract the buffer.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **sessionID** | SESSION_ID | A valid session ID. |
| **image** | Image* | Receives the copied image. If you set this parameter to NULL, the function creates an image to receive the copy. |
| **imageToCopy** | int | The image to copy from the ring acquisition. The cumulative buffer index specifies the image to copy. If another image has overwritten **imageToCopy**, the function returns the next available image. |
| **imageNumber** | int* | The cumulative buffer index of the copied image. Set this parameter to NULL if you do not need this information. |
| **rect** | Rect | The rectangular area of the image in the ring that the function copies. If you set this parameter to `IMAQ_NO_RECT`, or if the area is larger than the image in the ring, the function copies the entire image. |

## Return Value

**Image***—On success, this function returns the copied image. On failure, the function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with the image, dispose of it by calling `imaqDispose()`.

# imaqEasyAcquire

## Usage

```
Image* = imaqEasyAcquire(const char* interfaceName)
```

## Purpose

Configures the IMAQ device specified by **interfaceName**, acquires one image, and returns the acquired image.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **interfaceName** | const char* | A null-terminated string that specifies the name of the interface to open. Examples of interfaces are `img0` and `img1`. For more information about interfaces, see the *NI-IMAQ User Manual* or the Measurement & Automation Explorer online help. |

## Return Value

**Image***—On success, this function returns the acquired image. On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with the image, dispose of it by calling `imaqDispose()`.

# imaqExtractFromRing

## Usage

```
Image* = imaqExtractFromRing(SESSION_ID sessionID, int imageToExtract, int*
                             imageNumber)
```

## Purpose

Extracts an image from a live acquisition. This function lets you lock an image out of a continuous loop for processing during a ring acquisition. To unlock the image, call `imaqReleaseImage()`. The acquisition pauses when the continuous loop reaches the image you locked out.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **sessionID** | SESSION_ID | A valid session ID. |
| **imageToExtract** | int | The cumulative buffer index of the image to extract. If another image has overwritten **imageToExtract**, the function returns the next available image. |
| **imageNumber** | int* | The cumulative buffer index of the extracted image. Set this parameter to NULL if you do not need this information. |

## Return Value

**Image***—On success, this function returns a pointer to the extracted image. On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. Because the return value is a pointer to an image that you provided to `imaqSetupRing()`, you should not dispose of the image until the acquisition is finished.

# imaqGrab

## Usage

```
Image* = imaqGrab(SESSION_ID sessionID, Image* image, int immediate)
```

## Purpose

Returns a copy of the current image in the grab buffer. A grab performs an acquisition that loops continually on one buffer. Call this function only after calling `imaqGrabSetup()`.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **sessionID** | SESSION_ID | A valid session ID. |
| **image** | Image* | A pointer to the acquired image. If **image** is NULL, `imaqGrab()` creates the image into which the function copies the grab buffer. |
| **immediate** | int | Determines the acquisition timing method. Set this parameter to FALSE if you want the grab operation to synchronize on the vertical blank. Set the parameter to TRUE if you want an immediate transfer. See the *NI-IMAQ Function Reference Manual* or the *NI-IMAQ User Manual* for more information about acquisition timing methods. |

## Return Value

**Image***—On success, this function returns the acquired image. If you set **image** to NULL, the function returns a new image. Otherwise, the function returns a pointer to **image**.
On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`.

# imaqReleaseImage

## Usage

```
int = imaqReleaseImage(SESSION_ID sessionID)
```

## Purpose

Releases an image that `imaqExtractFromRing()` previously extracted. After the function releases the image, the live acquisition continues if the acquisition had paused.

## Parameters

| Name | Type | Description |
|---|---|---|
| **sessionID** | SESSION_ID | A valid session ID. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetupGrab

## Usage

```
int = imaqSetupGrab(SESSION_ID sessionID, Rect rect)
```

## Purpose

Configures and starts a grab acquisition. A grab performs an acquisition that loops continually on one buffer. Use `imaqGrab()` to copy an image out of the buffer. Use `imaqStopAcquisition()` to end the acquisition.

## Parameters

| Name | Type | Description |
|---|---|---|
| **sessionID** | SESSION_ID | A valid session ID. |
| **rect** | Rect | The area to acquire. If you set this parameter to `IMAQ_NO_RECT`, the function uses the entire acquisition window. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetupRing

## Usage

```
int = imaqSetupRing(SESSION_ID sessionID, Image** images, int numImages,
                            int skipCount, Rect rect)
```

## Purpose

Configures a ring acquisition. A ring acquisition acquires images continuously and loops them into a buffer list. To start the acquisition, call `imaqStartAcquisition()`. To stop the acquisition, call `imaqStopAcquisition()`. To get an image from the ring, call `imaqExtractFromRing()` or `imaqCopyFromRing()`. Do not modify or dispose of the images in the ring until you end the acquisition with `imaqStopAcquistion()`.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **sessionID** | SESSION_ID | A valid session ID. |
| **images** | Image** | An array of images. Each element in the array must be a pointer to a valid image. |
| **numImages** | int | The number of images in the images array. |
| **skipCount** | int | The number of frames to skip between each acquired image. A **skipCount** of 0 acquires images continuously without skipping frames between acquired images. |
| **rect** | Rect | The area to acquire. Set this parameter to `IMAQ_NO_RECT` to acquire the entire acquisition window. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetupSequence

## Usage

```
int = imaqSetupSequence(sessionID SESSION_ID, Image** images, int numImages,
                        int skipCount, Rect rect)
```

## Purpose

Configures a sequence acquisition. A sequence acquisition acquires a full sequence of images into the image array. To start the acquisition, call `imaqStartAcquisition()`. The acquisition finishes upon reaching the end of the sequence or when you call `imaqStopAcquisition()`. Do not modify or dispose of the images in the sequence until the acquisition has finished.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **sessionID** | SESSION_ID | A valid session ID. |
| **images** | Image** | An array of images. Each element in the array must be a pointer to a valid image. |
| **numImages** | int | The number of images in the **images** array. |
| **skipCount** | int | The number of frames to skip between each acquired image. A **skipCount** of 0 acquires images continuously without skipping frames between acquired images. |
| **rect** | Rect | The area to acquire. Set this parameter to `IMAQ_NO_RECT` to acquire the entire acquisition window. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSnap

## Usage

```
Image* = imaqSnap(SESSION_ID sessionID, Image* image, Rect rect)
```

## Purpose

Acquires a single image.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **sessionID** | SESSION_ID | A valid session ID. |
| **image** | Image* | The image into which to acquire. If **image** is NULL, `imaqSnap()` creates a new image. |
| **rect** | Rect | The area to acquire. Set this parameter to `IMAQ_NO_RECT` to acquire the entire acquisition window. |

## Return Value

**Image***—On success, this function returns the acquired image. If you set **image** to NULL, the function returns a new image. Otherwise, the function returns a pointer to **image**. On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`.

# imaqStartAcquisition

## Usage

```
int = imaqStartAcquisition(SESSION_ID sessionID)
```

## Purpose

Starts an acquisition identified by **sessionID**. Use this function with sequence and ring functions.

## Parameters

| Name | Type | Description |
|---|---|---|
| **sessionID** | SESSION_ID | A valid session ID. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqStopAcquisition

## Usage

```
int = imaqStopAcquisition(SESSION_ID sessionID)
```

## Purpose

Stops a session acquisition identified by **sessionID**. Use this function with grab, ring, and sequence functions.

## Parameters

| Name | Type | Description |
|---|---|---|
| **sessionID** | SESSION_ID | A valid session ID. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# 6

# Display

This chapter describes the Display functions in IMAQ Vision for LabWindows/CVI.

## Display Function Panels

Table 6-1 lists the Display functions in a tree structure. The functions in the Display class are grouped according to the types of operations they perform. The first-level heading in the tree is the name of the class. The second-level headings are names of function subclasses. The third-level headings are names of individual function panels. Each Display function panel represents one function.

**Table 6-1.** Display Function Tree

| Class/Panel Name | Function Name |
|---|---|
| Display | |
| Close Window | `imaqCloseWindow` |
| Display Image | `imaqDisplayImage` |
| Get Last Key | `imaqGetLastKey` |
| Get System Window Handle | `imaqGetSystemWindowHandle` |
| Show Window | `imaqShowWindow` |
| Window Management | |
| Are Scrollbars Visible | `imaqAreScrollbarsVisible` |
| Bring Window To Top | `imaqBringWindowToTop` |
| Get Mouse Position | `imaqGetMousePos` |
| Get Window Grid | `imaqGetWindowGrid` |
| Get Window Handle | `imaqGetWindowHandle` |
| Get Window Position | `imaqGetWindowPos` |
| Get Window Size | `imaqGetWindowSize` |
| Get Window Title | `imaqGetWindowTitle` |
| Get Window Zoom | `imaqGetWindowZoom` |
| Is Window Visible | `imaqIsWindowVisible` |
| Move Window | `imaqMoveWindow` |
| Set Window Grid | `imaqSetWindowGrid` |
| Set Window Palette | `imaqSetWindowPalette` |
| Set Window Size | `imaqSetWindowSize` |
| Set Window Thread Policy | `imaqSetWindowThreadPolicy` |
| Set Window Title | `imaqSetWindowTitle` |
| Setup Window | `imaqSetupWindow` |
| Show Scrollbars | `imaqShowScrollbars` |
| Zoom Window | `imaqZoomWindow` |

**Table 6-1.** Display Function Tree (Continued)

| Class/Panel Name | Function Name |
|---|---|
| Tool Window | |
| Close Tool Window | `imaqCloseToolWindow` |
| Get Current Tool | `imaqGetCurrentTool` |
| Get Last Event | `imaqGetLastEvent` |
| Get Tool Window Position | `imaqGetToolWindowPos` |
| Is Tool Window Visible | `imaqIsToolWindowVisible` |
| Move Tool Window | `imaqMoveToolWindow` |
| Set Current Tool | `imaqSetCurrentTool` |
| Set Event Callback | `imaqSetEventCallback` |
| Set Tool Color | `imaqSetToolColor` |
| Setup Tool Window | `imaqSetupToolWindow` |
| Show Tool Window | `imaqShowToolWindow` |
| Overlay | |
| Create Overlay From Metafile | `imaqCreateOverlayFromMetafile` |
| Create Overlay From ROI | `imaqCreateOverlayFromROI` |
| Set Window Overlay | `imaqSetWindowOverlay` |

# Subclass Descriptions

Display functions allow you to display images in image windows. Display subclass descriptions are as follows:

- Window Management functions allow you to configure, move, and resize image windows. You can control up to 16 image windows at a time.
- Tool Windows functions allow you to manage the tool palette, which you use to select areas of an image in an image window.
- Overlay functions allow you to create overlays and associate them with image windows.

# Tool Window

The examples of the tool palette in Figure 6-1 have four icons per line. The tool palette on the left automatically transforms to the palette on the right when you manipulate a region tool in an image window.

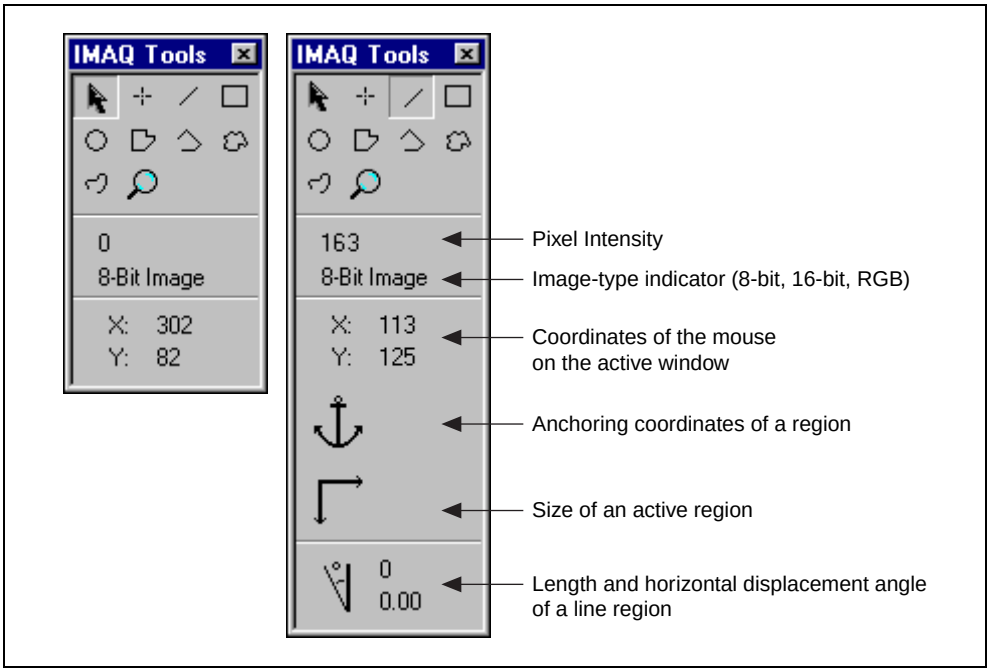



**Figure 6-1.** Tool Palette Transformation

## Tips for Using the Tool Window

The following are tips you can apply when using the tool window:

- Use `imaqGetLastEvent()` or register a callback with `imaqSetEventCallback()` to retrieve the draw events on a window and find the coordinates of a selected region.
- Alter the functionality of region tools by pressing certain keyboard keys while using the tool:
  - To constrain the x and y dimensions of an ROI, press <Shift> while drawing. This forces rectangles into squares, ellipses into circles, and line segments into horizontal or vertical segments.

  - To add an ROI without erasing the previous ROI elements, press <Control> when you click. The previous elements are erased if you do not use <Control> when starting a new element.
  - To produce the last point of a polygon or broken line, double-click while drawing.
- Use the selection tool to select an existing ROI by clicking its border. Once you select and ROI, you can manipulate it in the following ways:
  - To erase an ROI in an image window, select it and press <Delete>.
  - To resize a rectangle or ellipse, click in a grab handle and drag it to a new location.
  - To reposition a vertex in a broken line, polygon, or line, click in a grab handle and move it to a new location.
  - To reposition a rectangle or ellipse, click in the interior and drag it to a new location.
  - To reposition a point, click on it and drag it to a new location.
  - To reposition lines, broken lines, and polygons, click on any segment and drag it to a new location.
  - To reposition freehand lines and closed freehand lines, click anywhere on the line and drag it to a new location.

# imaqAreScrollbarsVisible

## Usage

```
int = imaqAreScrollbarsVisible(int windowNumber, int* visible)
```

## Purpose

Retrieves whether the scrollbars of the given image window are visible.

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the image window. |
| **visible** | int* | On return, this parameter is TRUE if the scrollbars are visible and FALSE if the scrollbars are hidden. This parameter is required and cannot be NULL. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqBringWindowToTop

## Usage

```
int = imaqBringWindowToTop(int windowNumber)
```

## Purpose

Makes the given image window active. This function has no effect on Windows 2000 if your application is not the active application in the system.

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the window you want to be active. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqCloseToolWindow

## Usage

```
int = imaqCloseToolWindow()
```

## Purpose

Closes the tool window and frees all associated resources.

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqCloseWindow

## Usage

```
int = imaqCloseWindow(int windowNumber)
```

## Purpose

Closes an image window and frees all associated resources.

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the image window to close. Set this parameter to `IMAQ_ALL_WINDOWS` to close all of the image windows. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqCreateOverlayFromMetafile

## Usage

```
Overlay* = imaqCreateOverlayFromMetafile(const void* metafile)
```

## Purpose

Creates a window overlay from a Windows metafile or enhanced metafile.

## Parameters

| Name | Type | Description |
|---|---|---|
| **metafile** | const void* | The Windows handle to the metafile that you want to convert into an overlay. The handle may be either an HMETAFILE or HENHMETAFILE. |

## Return Value

**Overlay***—On success, this function returns an overlay. On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with the overlay, dispose of it by calling `imaqDispose()`.

# imaqCreateOverlayFromROI

## Usage

```
Overlay* = imaqCreateOverlayFromROI(const ROI* roi)
```

## Purpose

Creates a window overlay from a region of interest.

## Parameters

| Name | Type | Description |
|---|---|---|
| **roi** | const ROI* | The region of interest to convert into an overlay. |

## Return Value

**Overlay***—On success, this function returns an overlay. On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with the overlay, dispose of it by calling `imaqDispose()`.

# imaqDisplayImage

## Usage

```
int = imaqDisplayImage(const Image* image, int windowNumber, int resize)
```

## Purpose

Displays an image in an image window. The window becomes visible when you call the function. The window is associated with the image until you close the window, dispose of the image, or call this function again with the same window number.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_COMPLEX`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image to display in the window. |
| **windowNumber** | int | The window number of the window in which to display the image. There are 16 image windows, which have window numbers 0–15. To obtain a window number not in use, call `imaqGetWindowHandle()`. |
| **resize** | int | If you set this parameter to TRUE, the window resizes to the size of the image. If you set this parameter to FALSE, the window size does not change. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetCurrentTool

## Usage

```
int = imaqGetCurrentTool(Tool* currentTool)
```

## Purpose

Returns the currently selected tool from the tool window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **currentTool** | Tool* | On return, contains the currently selected tool. This parameter is required and cannot be NULL. |

## Parameter Discussion

**currentTool**—The following options are valid:

- `IMAQ_NO_TOOL`—No tool is in the selected state.
- `IMAQ_SELECTION_TOOL`—The selection tool selects an existing ROI in an image.
- `IMAQ_POINT_TOOL`—The point tool draws a point on the image.
- `IMAQ_LINE_TOOL`—The line tool draws a line on the image.
- `IMAQ_RECTANGLE_TOOL`—The rectangle tool draws a rectangle on the image.
- `IMAQ_OVAL_TOOL`—The oval tool draws an oval on the image.
- `IMAQ_POLYGON_TOOL`—The polygon tool draws a polygon on the image.
- `IMAQ_CLOSED_FREEHAND_TOOL`—The closed freehand tool draws closed freehand shapes on the image.
- `IMAQ_ZOOM_TOOL`—The zoom tool controls the zoom of an image.
- `IMAQ_POLYLINE_TOOL`—The polyline tool draws a series of connected straight lines on the image.
- `IMAQ_FREEHAND_TOOL`—The freehand tool draws freehand lines on the image.

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetLastEvent

## Usage

```
int = imaqGetLastEvent(WindowEventType* type, int* windowNumber, Tool* tool,
                       Rect* rect)
```

## Purpose

Returns the last event that the user performed on an image window. Do not use this function if you have registered an event callback with `imaqSetEventCallback()`.

## Parameters

| Name | Type | Description |
|---|---|---|
| **type** | WindowEventType* | On return, the last event that occurred. |
| **windowNumber** | int* | On return, the window number of the window in which the last event occurred. Set this parameter to NULL if you do not need this information. |
| **tool** | Tool* | If the event was `IMAQ_DRAW_EVENT`, **tool** is the ROI tool with which the user drew. If the event was not `IMAQ_DRAW_EVENT`, the function ignores this parameter. Set this parameter to NULL if you do not need this information. |
| **rect** | Rect* | A rectangle describing the location of the event. Set this parameter to NULL if you do not need this information. |

## Parameter Discussion

For **type**, the last event can be one of the following:

- `IMAQ_NO_EVENT`—No event occurred since the last call to `imaqGetLastEvent()`.
- `IMAQ_CLICK_EVENT`—The user clicked on a window.
- `IMAQ_DRAW_EVENT`—The user drew an ROI in a window.
- `IMAQ_MOVE_EVENT`—The user moved a window.
- `IMAQ_SIZE_EVENT`—The user sized a window.
- `IMAQ_SCROLL_EVENT`—The user scrolled a window.
- `IMAQ_ACTIVATE_EVENT`—The user activated a window.
- `IMAQ_CLOSE_EVENT`—The user closed a window.
- `IMAQ_DOUBLE_CLICK_EVENT`—The user double-clicked in a window.

For **rect**, the contents of the rectangle depend on **type**, as follows:

- `IMAQ_CLICK_EVENT`—The top left corner of the rectangle is the location of the click. The width and height of the rectangle are 0.
- `IMAQ_SCROLL_EVENT`—The top left of the rectangle is the center of the displayed image. The width and height of the rectangle are 0.
- `IMAQ_DRAW_EVENT`—The rectangle is the bounding rectangle of the drawn shape.
- `IMAQ_MOVE_EVENT` or `IMAQ_SIZE_EVENT`—The rectangle is the new location of the window on the screen.

For all other events, the function ignores the rectangle.

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetLastKey

## Usage

```
int = imaqGetLastKey(char* keyPressed, int* windowNumber, int* modifiers)
```

## Purpose

Returns the last key pressed in an active image window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **keyPressed** | char* | On return, contains the last key pressed. Returns -1 if there was no new key press to retrieve. Set this parameter to NULL if you do not need this information. |
| **windowNumber** | int* | On return, contains the window number of the window in which the key press was caught. Returns –1 if there was no new key press to retrieve. Set this parameter to NULL if you do not need this information. |
| **modifiers** | int* | On return, will contain a bit-shifted value indicating what modifiers, if any, the function applied to the key press. The following are possible modifiers:<br>`IMAQ_SHIFT`<br>`IMAQ_ALT`<br>`IMAQ_CTRL`<br>`IMAQ_CAPS_LOCK`<br>Set this parameter to NULL if you do not need this information. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetMousePos

## Usage

```
int = imaqGetMousePos(Point* position, int* windowNumber)
```

## Purpose

Returns the mouse cursor coordinates and window number of the most recent instance that the mouse cursor was located over an active window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **position** | Point* | Upon return, the coordinates of the mouse in the active image window. Set this parameter to NULL if you do not need this information. |
| **windowNumber** | int* | Upon return, contains the window number of the active window. Set this parameter to NULL if you do not need this information. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetSystemWindowHandle

## Usage

```
void* = imaqGetSystemWindowHandle(int windowNumber)
```

## Purpose

Returns the Windows HWND for a given IMAQ Vision image window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the window whose HWND to retrieve. |

## Return Value

**void***—On success, this function returns the Windows HWND for the window. On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`.

# imaqGetToolWindowPos

## Usage

```
int = imaqGetToolWindowPos(Point* position)
```

## Purpose

Retrieves the current location of the tool window. The function behaves in the same manner as `imaqGetWindowPos()`.

## Parameters

| Name | Type | Description |
|---|---|---|
| **position** | Point* | Upon return, the position of the upper left corner of the tool window. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetWindowGrid

## Usage

```
int = imaqGetWindowGrid(int windowNumber, int* xResolution, int*
                        yResolution)
```

## Purpose

Retrieves the grid resolution of the image window. Grid resolution is the number of pixels between grid lines. IMAQ Vision uses the grid resolution when drawing regions of interest on the window using tools in the tool window. You can use the grid to trace a region of interest accurately.

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the image window. |
| **xResolution** | int* | On return, the number of pixels between grid lines in the *x* direction. Set this parameter to NULL if you do not need this information. |
| **yResolution** | int* | On return, the number of pixels between grid lines in the *y* direction. Set this parameter to NULL if you do not need this information. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetWindowHandle

## Usage

```
int = imaqGetWindowHandle(int* handle)
```

## Purpose

Returns an unused window number. You can use the window number in conjunction with functions such as `imaqDisplayImage()`. This function does not reserve the window number until you call a function that uses the window number.

## Parameters

| Name | Type | Description |
|---|---|---|
| **handle** | int* | On return, an unused window number. If no unused window numbers are available, the function sets this parameter to –1. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetWindowPos

## Usage

```
int = imaqGetWindowPos(int windowNumber, Point* position)
```

## Purpose

Retrieves the current location of the given image window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the image window. |
| **position** | Point* | On return, the position of the upper left corner of the given image window. This parameter is required and cannot be NULL. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetWindowSize

## Usage

```
int = imaqGetWindowSize(int windowNumber, int* width, int* height)
```

## Purpose

Retrieves the size of a given image window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the image window. |
| **width** | int* | On return, the width of the window. Set this parameter to NULL if you do not need this information. |
| **height** | int* | On return, the height of the window. Set this parameter to NULL if you do not need this information. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetWindowTitle

## Usage

```
char* = imaqGetWindowTitle(int windowNumber)
```

## Purpose

Retrieves the current title of an image window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the image window. |

## Return Value

**char***—On success, this function returns the title of the given window. On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with the title, dispose of it by calling `imaqDispose()`.

# imaqGetWindowZoom

## Usage

```
int = imaqGetWindowZoom(int windowNumber, int* xZoom, int* yZoom)
```

## Purpose

Retrieves the current zoom factors for a given image window. The zoom factor indicates an increase or decrease in the magnification of an image. A positive number indicates a magnification by the amount specified. For example, a zoom factor of 3 indicates that the image is displayed at three times its actual size (3:1). A negative number indicates that the image is decreased in magnification by the specified amount. For example, a zoom factor of –5 indicates that the image is displayed at one-fifth its actual size (1:5).

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the image window. |
| **xZoom** | int* | On return, the current zoom factor in the *x* direction for the window. |
| **yZoom** | int* | On return, the current zoom factor in the *y* direction for the window. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqIsToolWindowVisible

## Usage

```
int = imaqIsToolWindowVisible(int* visible)
```

## Purpose

Retrieves whether the tool window is visible. This function behaves in the same manner as `imaqIsWindowVisible()`.

## Parameters

| Name | Type | Description |
|---|---|---|
| **visible** | int* | On return, this parameter is TRUE if the tool window is visible and FALSE if the tool window is hidden. This parameter is required and cannot be NULL. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqIsWindowVisible

## Usage

```
int = imaqIsWindowVisible(int windowNumber, int* visible)
```

## Purpose

Retrieves whether the given window is visible.

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the image window. |
| **visible** | int* | On return, this parameter is TRUE if the given window is visible and FALSE if the window is hidden. This parameter is required and cannot be NULL. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqMoveToolWindow

## Usage

```
int = imaqMoveToolWindow(Point position)
```

## Purpose

Moves the tool window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **position** | Point | The new position, in screen coordinates, of the upper left corner of the tool window. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqMoveWindow

## Usage

```
int = imaqMoveWindow(int windowNumber, Point position)
```

## Purpose

Moves an image window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the image window. |
| **position** | Point | The new position, in screen coordinates, of the upper left corner of the window. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetCurrentTool

## Usage

```
int = imaqSetCurrentTool(Tool currentTool)
```

## Purpose

Sets the currently selected tool in the tool window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **currentTool** | Tool | The tool to make the selected region tool. |

## Parameter Discussion

**currentTool**—The following options are valid:

- IMAQ_NO_TOOL—Sets all tools to the unselected state.
- IMAQ_SELECTION_TOOL—The selection tool selects an existing ROI in an image.
- IMAQ_POINT_TOOL—The point tool draws a point on the image.
- IMAQ_LINE_TOOL—The line tool draws a line on the image.
- IMAQ_RECTANGLE_TOOL—The rectangle tool draws a rectangle on the image.
- IMAQ_OVAL_TOOL—The oval tool draws an oval on the image.
- IMAQ_POLYGON_TOOL—The polygon tool draws a polygon on the image.
- IMAQ_CLOSED_FREEHAND_TOOL—The closed freehand tool draws closed freehand shapes on the image.
- IMAQ_ZOOM_TOOL—The zoom tool controls the zoom of an image.
- IMAQ_POLYLINE_TOOL—The polyline tool draws a series of connected straight lines on the image.
- IMAQ_FREEHAND_TOOL—The freehand tool draws freehand lines on the image.

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetEventCallback

## Usage

```
int = imaqSetEventCallback(EventCallback callback, int synchronous)
```

## Purpose

Sets a callback function that IMAQ Vision calls when an event occurs in a window. When a user generates an event, IMAQ Vision calls the callback function using the following parameters: event, window number, tool, and location of the event. For a full description of these parameters, see `imaqGetLastEvent()`.

## Parameters

| Name | Type | Description |
|---|---|---|
| **callback** | EventCallback | The function to call. Set this parameter to NULL if you want to disable event processing using a callback. If you disable callbacks, you can process events using `imaqGetLastEvent()`. |
| **synchronous** | int | Set this parameter to TRUE to call the callback function in the thread that calls `imaqSetEventCallback()`. Set this parameter to FALSE to call the callback function asynchronously in a separate thread. To process callbacks synchronously, your application must have a message pump. In LabWindows/CVI, calling `RunUserInterface()` starts a message pump. The function ignores this parameter if **callback** is NULL. |

## Parameter Discussion

**callback**—The callback should have the following prototype:

```
void IMAQ_CALLBACK MyCallback(WindowEventType, int windowNumber,
                              Tool tool, Rect rect);
```

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetToolColor

## Usage

```
int = imaqSetToolColor(const RGBValue* color)
```

## Purpose

Sets the color in which the tools from the tool window draw.

## Parameters

| Name | Type | Description |
|---|---|---|
| **color** | const RGBValue* | The tool drawing color. This parameter is required and cannot be NULL. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetupToolWindow

## Usage

```
int = imaqSetupToolWindow(int showCoordinates, int maxIconsPerLine, const
                          ToolWindowOptions* options)
```

## Purpose

Configures the appearance and availability of the tools in the tool window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **showCoordinates** | int | Determines whether active pixel coordinates are visible. Set this parameter to TRUE to display the active pixel coordinates. Set this parameter to FALSE if you do not want the coordinates to show. |
| **maxIconsPerLine** | int | The maximum number of tool icons to show on each line. The tool window uses the minimum number of lines needed to display all of the tools based on this parameter and distributes the tools as evenly as possible. |
| **options** | const ToolWindowOptions* | Determines the availability of tools in the tool window. |

## Parameter Discussion

Set **options** to NULL to display all the tools. A `ToolWindowOptions` structure has the following elements:

- `showSelectionTool`—If TRUE, the selection tool becomes visible. The selection tool selects an existing ROI in an image.
- `showZoomTool`—If TRUE, the zoom tool becomes visible. The zoom tool controls the magnification of an image.
- `showPointTool`—If TRUE, the point tool becomes visible. The point tool draws a point on the image.
- `showLineTool`—If TRUE, the line tool becomes visible. The line tool draws a line on the image.
- `showRectangleTool`—If TRUE, the rectangle tool becomes visible. The rectangle tool draws a rectangle on the image.

- `showOvalTool`—If TRUE, the oval tool becomes visible. The oval tool draws an oval on the image.
- `showPolygonTool`—If TRUE, the polygon tool becomes visible. The polygon tool draws a polygon on the image.
- `showClosedFreehandTool`—If TRUE, the closed freehand tool becomes visible. The closed freehand tool draws closed freehand shapes on the image.
- `showPolyLineTool`—If TRUE, the polyline tool becomes visible. The polyline tool draws a series of connected straight lines on the image.
- `showFreehandTool`—If TRUE, the freehand tool becomes visible. The freehand tool draws freehand lines on the image.

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetupWindow

## Usage

```
int = imaqSetupWindow(int windowNumber, int configuration)
```

## Purpose

Sets the properties of an image window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the image window. |
| **configuration** | int | Any of the following flags combined together:<br>IMAQ_WIND_RESIZABLE—When present, the user may resize the window interactively. When absent, you can only resize the window programmatically.<br>IMAQ_WIND_TITLEBAR—When present, the title bar on the window is visible. When absent, the title bar on the window is not visible.<br>IMAQ_WIND_CLOSABLE—When present, the close box is available. When absent, the close box is removed. The title bar must be present for this flag to have effect.<br>IMAQ_WIND_TOPMOST—When present, the window is always on top. When absent, the window is on top only when active. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetWindowGrid

## Usage

```
int = imaqSetWindowGrid(int windowNumber, int xResolution, int yResolution)
```

## Purpose

Sets the grid resolution of the image window. Grid resolution is the number of pixels between grid lines. IMAQ Vision uses the grid resolution when drawing regions of interest on the window using tools in the tool window. You can use the grid to trace a region of interest accurately.

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the image window. |
| **xResolution** | int | The *x* resolution of the grid. |
| **yResolution** | int | The *y* resolution of the grid. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetWindowOverlay

## Usage

```
int = imaqSetWindowOverlay(int windowNumber, const Overlay* overlay)
```

## Purpose

Sets or removes the overlay from a window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the image window. |
| **overlay** | const Overlay* | The overlay. Set this parameter to NULL to remove any existing overlay from the given window. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetWindowPalette

## Usage

```
int = imaqSetWindowPalette(int windowNumber, PaletteType type, const
                           RGBValue* palette, int numColors)
```

## Purpose

Sets the color palette to use when displaying a monochrome image in an image window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the image window. |
| **type** | PaletteType | The palette type to use. The following values are valid:<br>`IMAQ_PALETTE_GRAY`<br>`IMAQ_PALETTE_BINARY`<br>`IMAQ_PALETTE_GRADIENT`<br>`IMAQ_PALETTE_RAINBOW`<br>`IMAQ_PALETTE_TEMPERATURE`<br>`IMAQ_PALETTE_USER` |
| **palette** | const RGBValue* | If **type** is `IMAQ_PALETTE_USER`, this array is the palette of colors to use with the window. If **type** is not `IMAQ_PALETTE_USER`, the function ignores this parameter, and you may set it to NULL. The maximum number of colors in a palette is 256. **palette**[`n`] maps to pixel value `n`. If there are less than 256 elements in **palette**, the function maps all pixel values past the last element in **palette** to black. |
| **numColors** | int | If **type** is `IMAQ_PALETTE_USER`, this parameter is the number of colors in the palette array. If **type** is not `IMAQ_PALETTE_USER`, the function ignores this parameter. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetWindowSize

## Usage

```
int = imaqSetWindowSize(int windowNumber, int width, int height)
```

## Purpose

Sets the size of an image window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the image window. |
| **width** | int | The new width of the window. |
| **height** | int | The new height of the window. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetWindowThreadPolicy

## Usage

```
int = imaqSetWindowThreadPolicy(WindowThreadPolicy policy)
```

## Purpose

Determines the thread in which IMAQ Vision creates windows. By default, IMAQ Vision uses `IMAQ_CALLING_THREAD`. This policy creates windows in the thread that makes the first display function call for a given window number. If that thread does not process messages, set the window thread policy to `IMAQ_SEPARATE_THREAD`. Using this policy, IMAQ Vision creates windows in a separate thread and processes messages for the windows automatically.

## Parameters

| Name | Type | Description |
|---|---|---|
| **policy** | WindowThreadPolicy | The thread policy. Valid options are `IMAQ_CALLING_THREAD` and `IMAQ_SEPARATE_THREAD`. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetWindowTitle

## Usage

```
int = imaqSetWindowTitle(int windowNumber, const char* title)
```

## Purpose

Sets the title of an image window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the image window. |
| **title** | const char* | The new title of the window. This parameter is required and cannot be NULL. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqShowScrollbars

## Usage

```
int = imaqShowScrollbars(int windowNumber, int visible)
```

## Purpose

Shows or hides the scrollbars on an image window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the image window. |
| **visible** | int | If TRUE, the scrollbars of the window are visible. If FALSE, the scrollbars of the window are hidden. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqShowToolWindow

## Usage

```
int = imaqShowToolWindow(int visible)
```

## Purpose

This function shows or hides the tool window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **visible** | int | If TRUE, the tool window becomes visible. If FALSE, the tool window becomes hidden. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqShowWindow

## Usage

```
int = imaqShowWindow(int windowNumber, int visible)
```

## Purpose

Shows or hides an image window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the image window. |
| **visible** | int | If TRUE, the given window is visible. If FALSE, the given window is hidden. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqZoomWindow

## Usage

```
int = imaqZoomWindow(int windowNumber, int xZoom, int yZoom, Point center)
```

## Purpose

Sets the current zoom factors for a given image window. The zoom factor indicates an increase or decrease in the magnification of an image. A positive number indicates a magnification by the amount specified. For example, a zoom factor of 3 indicates that the image is displayed at three times its actual size (3:1). A negative number indicates that the image is decreased in magnification by the specified amount. For example, a zoom factor of –5 indicates that the image is displayed at one-fifth its actual size (1:5).

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the image window. |
| **xZoom** | int | The zoom factor for the *x* direction. |
| **yZoom** | int | The zoom factor for the *y* direction. |
| **center** | Point | The center point around which to zoom. Set this parameter to `IMAQ_NO_POINT` to maintain the current center point. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# 7

# Regions of Interest

This chapter describes the Regions of Interest (ROI) functions in IMAQ Vision for LabWindows/CVI.

## Regions of Interest Function Panels

Table 7-1 lists the Regions of Interest functions in a tree structure. The functions in the Regions of Interest class are grouped according to the types of operations they perform. The first-level heading in the tree is the name of the class. The second-level headings are names of function subclasses. The third-level headings are names of individual function panels. Each Regions of Interest function panel represents one function.

**Table 7-1.** Regions of Interest Function Tree

| Class/Panel Name | Function Name |
|---|---|
| Regions of Interest | |
| Create ROI | `imaqCreateROI` |
| Get ROI Bounding Box | `imaqGetROIBoundingBox` |
| Get ROI Color | `imaqGetROIColor` |
| Get Window ROI | `imaqGetWindowROI` |
| Mask To ROI | `imaqMaskToROI` |
| ROI Profile | `imaqROIProfile` |
| ROI To Mask | `imaqROIToMask` |
| Set ROI Color | `imaqSetROIColor` |
| Set Window ROI | `imaqSetWindowROI` |
| Transform ROI | `imaqTransformROI` |
| Contours | |
| Add Closed Contour | `imaqAddClosedContour` |
| Add Line Contour | `imaqAddLineContour` |
| Add Open Contour | `imaqAddOpenContour` |
| Add Oval Contour | `imaqAddOvalContour` |
| Add Point Contour | `imaqAddPointContour` |
| Add Rect Contour | `imaqAddRectContour` |
| Copy Contour | `imaqCopyContour` |
| Get Contour | `imaqGetContour` |
| Get Contour Color | `imaqGetContourColor` |
| Get Contour Count | `imaqGetContourCount` |
| Get Contour Information | `imaqGetContourInfo` |
| Remove Contour | `imaqRemoveContour` |
| Set Contour Color | `imaqSetContourColor` |

# Subclass Description

Regions of Interest functions allow you to create, modify, and extract information about regions of interest. The Regions of Interest subclass description is as follows:

- Contour functions allow you to create and modify individual contours of a region of interest.

# imaqAddClosedContour

## Usage

```
ContourID = imaqAddClosedContour(ROI* roi, const Point* points,
                                 int numPoints)
```

## Purpose

Creates a new ROI contour based on the provided array of points. To make the contour, the function connects each point in the array to the next point in the array, and it connects the last point in the array to the first point in the array. The function adds the contour to the provided ROI.

## Parameters

| Name | Type | Description |
|---|---|---|
| **roi** | ROI* | The ROI to contain the new contour. |
| **points** | const Point* | An array of points describing the location and shape of the contour. This parameter is required and cannot be NULL. |
| **numPoints** | int | The number of points in the array. |

## Return Value

**ContourID**—On success, this function returns a ContourID for the added contour. You can use the ContourID to reference the contour within the containing ROI. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqAddLineContour

## Usage

```
ContourID = imaqAddLineContour(ROI* roi, Point start, Point end)
```

## Purpose

Creates a new line ROI contour and adds the line to the provided ROI.

## Parameters

| Name | Type | Description |
|---|---|---|
| **roi** | ROI* | The ROI to contain the new contour. |
| **start** | Point | The pixel location of the start of the line. |
| **end** | Point | The pixel location of the end of the line. |

## Return Value

**ContourID**—On success, this function returns a ContourID for the contour. You can use the ContourID to reference the contour within the containing ROI. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqAddOpenContour

## Usage

```
ContourID = imaqAddOpenContour(ROI* roi, const Point* points, int numPoints)
```

## Purpose

Creates a new ROI contour based on the provided array of points. To make the contour, the function connects each point in the array to the next point in the array. The function does not connect the last point in the array to the first point in the array. The function adds the contour to the provided ROI.

## Parameters

| Name | Type | Description |
|---|---|---|
| **roi** | ROI* | The ROI to contain the new contour. |
| **points** | const Point* | An array of points describing the location and shape of the contour. This parameter is required and cannot be NULL. |
| **numPoints** | int | The number of points in the array. |

## Return Value

**ContourID**—On success, this function returns a ContourID for the contour. You can use the ContourID to reference the contour within the containing ROI. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqAddOvalContour

## Usage

```
ContourID = imaqAddOvalContour(ROI* roi, Rect boundingBox)
```

## Purpose

Creates a new oval ROI contour and adds the oval to the provided ROI.

## Parameters

| Name | Type | Description |
|---|---|---|
| **roi** | ROI* | The ROI to contain the new contour. |
| **boundingBox** | Rect | The pixel location of the bounding box of the oval. |

## Return Value

**ContourID**—On success, this function returns a ContourID for the contour. You can use the ContourID to reference the contour within the containing ROI. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqAddPointContour

## Usage

```
ContourID = imaqAddPointContour(ROI* roi, Point point)
```

## Purpose

Creates a new single-point ROI contour and adds the point to the provided ROI.

## Parameters

| Name | Type | Description |
|---|---|---|
| **roi** | ROI* | The ROI to contain the new contour. |
| **point** | Point | The pixel location of the point. |

## Return Value

**ContourID**—On success, this function returns a ContourID for the contour. You can use the ContourID to reference the contour within the containing ROI. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqAddRectContour

## Usage

```
ContourID = imaqAddRectContour(ROI* roi, Rect rect)
```

## Purpose

Creates a new rectangle ROI contour and adds the rectangle to the provided ROI.

## Parameters

| Name | Type | Description |
|---|---|---|
| **roi** | ROI* | The ROI to contain the new contour. |
| **rect** | Rect | The pixel location of the rectangle. |

## Return Value

**ContourID**—On success, this function returns a ContourID for the contour. You can use the ContourID to reference the contour within the containing ROI. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqCopyContour

## Usage

```
ContourID = imaqCopyContour(ROI* destRoi, const ROI* sourceRoi, ContourID id)
```

## Purpose

Copies a contour existing in one ROI to another ROI. Copying the contour does not affect the original contour or the source ROI.

## Parameters

| Name | Type | Description |
|---|---|---|
| **destRoi** | ROI* | The ROI to which the function adds the contour. |
| **sourceRoi** | const ROI* | The ROI that contains the contour to copy. |
| **id** | ContourID | The contour to add to the ROI. |

## Return Value

**ContourID**—On success, this function returns the ContourID of the contour in the destination ROI. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqCreateROI

## Usage

```
ROI* = imaqCreateROI()
```

## Purpose

Creates a new, empty region of interest (ROI).

## Return Value

**ROI***—On success, this function returns a pointer to a new, empty ROI. On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with the ROI, dispose of the pointer by calling `imaqDispose()`.

# imaqGetContour

## Usage

```
ContourID = imaqGetContour(const ROI* roi, unsigned int index)
```

## Purpose

Returns the ContourID of the contour at the specified index location within an ROI.

## Parameters

| Name | Type | Description |
|---|---|---|
| **roi** | const ROI* | The ROI containing the desired contour. |
| **index** | unsigned int | The zero-offset index of the contour to get. |

## Return Value

**ContourID**—On success, this function returns the ContourID of the requested contour. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetContourColor

## Usage

```
int = imaqGetContourColor(const ROI* roi, ContourID id, RGBValue*
                          contourColor)
```

## Purpose

Returns the color of a contour.

## Parameters

| Name | Type | Description |
|---|---|---|
| **roi** | const ROI* | The ROI containing the contour from which the function gets color information. |
| **id** | ContourID | The ContourID of the contour from which the function gets color information. |
| **contourColor** | RGBValue* | On return, the color of the contour. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetContourCount

## Usage

```
int = imaqGetContourCount(const ROI* roi)
```

## Purpose

Returns the number of contours in an ROI.

## Parameters

| Name | Type | Description |
|---|---|---|
| **roi** | const ROI* | The ROI from which the function gets the contour count. |

## Return Value

**int**—On success, this function returns the number of contours in the ROI. On failure, this function returns –1. To get extended error information, call `imaqGetLastError()`.

# imaqGetContourInfo

## Usage

```
ContourInfo* = imaqGetContourInfo(const ROI* roi, ContourID id)
```

## Purpose

Returns information about a particular contour. The structure that the function returns contains a copy of the data from the contour. Modifications to the information in the structure do not affect the contour.

## Parameters

| Name | Type | Description |
|---|---|---|
| **roi** | const ROI* | The ROI containing the contour from which the function gets the information. |
| **id** | ContourID | The ContourID of the contour about which the function gets information. |

## Return Value

**ContourInfo*—**On success, this function returns a pointer to the structure containing information about the requested contour. The structure contains the following information:

- `type`—The type of the contour. The following options are valid:
  `IMAQ_EMPTY_CONTOUR`
  `IMAQ_POINT`
  `IMAQ_LINE`
  `IMAQ_RECT`
  `IMAQ_OVAL`
  `IMAQ_CLOSED_CONTOUR`
  `IMAQ_OPEN_CONTOUR`
- `numPoints`—The number of points that make up the contour.
- `points`—The points describing the contour.
- `contourColor`—The color of the contour.

On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with this structure, dispose of the pointer by calling `imaqDispose()`.

# imaqGetROIBoundingBox

## Usage

```
int = imaqGetROIBoundingBox(const ROI* roi, Rect* boundingBox)
```

## Purpose

Returns the bounding box for the ROI. The bounding box is the smallest rectangle that contains all of the contours that comprise the ROI.

## Parameters

| Name | Type | Description |
|---|---|---|
| **roi** | const ROI* | The ROI from which the function gets the bounding box information. |
| **boundingBox** | Rect* | On return, the bounding box. This parameter is required and cannot be NULL. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetROIColor

## Usage

```
int = imaqGetROIColor(const ROI* roi, RGBValue* roiColor)
```

## Purpose

Returns the color of an ROI.

## Parameters

| Name | Type | Description |
|---|---|---|
| **roi** | const ROI* | The ROI from which the function gets color information. |
| **roiColor** | RGBValue* | On return, the color of the ROI. This parameter is required and cannot be NULL. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetWindowROI

## Usage

```
ROI* = imaqGetWindowROI(int windowNumber)
```

## Purpose

Retrieves a copy of the ROI associated with a given image window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the image window. |

## Return Value

**ROI***—On success, this function returns a copy of the ROI associated with the given window. On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with the ROI, dispose of it by calling `imaqDispose()`.

# imaqMaskToROI

## Usage

```
ROI* = imaqMaskToROI(const Image* mask, int* withinLimit)
```

## Purpose

Transforms a mask image into a region of interest descriptor. The number of points that define the ROI cannot exceed 2,500.

## Image Type Supported

`IMAQ_IMAGE_U8`

## Parameters

| Name | Type | Description |
|---|---|---|
| **mask** | const Image* | The mask image that the function transforms into a ROI. This image must be an `IMAQ_IMAGE_U8` image. |
| **withinLimit** | int* | On return, this parameter indicates whether the ROI is a true representation of the mask. If TRUE, the number of points is within the 2,500 point limit. If FALSE, the number of points exceeds the 2,500 limit, and the ROI may not represent the mask completely. Set this parameter to NULL if you do not need this information. |

## Return Value

**ROI***—On success, this function returns a pointer to the ROI descriptor. On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with this descriptor, dispose of the pointer by calling `imaqDispose()`.

# imaqRemoveContour

## Usage

```
int = imaqRemoveContour(ROI* roi, ContourID id)
```

## Purpose

Deletes the specified contour from the specified ROI, freeing all allocated memory associated with the contour.

## Parameters

| Name | Type | Description |
|---|---|---|
| **roi** | ROI* | The ROI containing the contour to remove. |
| **id** | ContourID | The ContourID of the contour to remove. After this operation, the ContourID no longer correlates to a valid contour. Set this parameter to `IMAQ_ALL_CONTOURS` to remove all contours. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqROIProfile

## Usage

```
ROIProfile* = imaqROIProfile(const Image* image, const ROI* roi)
```

## Purpose

Calculates the profile of the pixels along the edge of each contour in an ROI.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image from which the function gets the profile. |
| **roi** | const ROI* | The ROI describing the pixels about which the function gets information. |

## Return Value

**ROIProfile***—On success, this function returns a pointer to information about the points along the edge of each contour in the ROI. The `ROIProfile` structure consists of the following elements:

- `report`—A `LineProfile` structure quantifying information about the points along the edge of each contour in the ROI. The `LineProfile` structure consists of the following elements:
  - `profileData`—An array of the pixel values along the edge of each contour in the ROI. This array has `dataCount` elements.
  - `boundingBox`—The bounding rectangle of the ROI.
  - `min`—The minimum pixel value along the edge of each contour in the ROI.
  - `max`—The maximum pixel value along the edge of each contour in the ROI.
  - `mean`—The mean of the pixel values along the edge of each contour in the ROI.
  - `stdDev`—The standard deviation of the pixel values along the edge of each contour in the ROI.
  - `dataCount`—The number of points along the edge of each contour in the ROI.
- `pixels`—An array of the points along the edge of each contour in the ROI. This array has a number of `Point` structures equal to the `dataCount` in `report`.

On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with this information, dispose of it by calling `imaqDispose()`.

# imaqROIToMask

## Usage

```
int = imaqROIToMask(Image* mask, const ROI* roi, int fillValue, const Image*
                    imageModel, int* inSpace)
```

## Purpose

Transforms an ROI into a mask image.

## Image Type Supported

`IMAQ_IMAGE_U8`

## Parameters

| Name | Type | Description |
|---|---|---|
| **mask** | Image* | The resulting mask image. |
| **roi** | const ROI* | The descriptor defining the ROI. |
| **fillValue** | int | The pixel value of the mask. All pixels inside the ROI receive this value. |
| **imageModel** | const Image* | An optional template for the destination mask image. This parameter can be any image type that IMAQ Vision supports. If you supply an **imageModel**, the mask image is the size of the model. If you set **imageModel** to NULL, the size of **mask** is equal to the size of the bounding box of the ROI, which reduces the amount of memory used. The function sets the offset of the mask image to reflect the real position of the ROI. |
| **inSpace** | int* | If you used **imageModel**, this parameter indicates on return whether the mask is a true representation of the ROI. If TRUE, the ROI data is completely within **imageModel**. If FALSE, some of the ROI data fell outside the space associated with **imageModel**. Set this parameter to NULL if you do not need this information, or if you did not use **imageModel**. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetContourColor

## Usage

```
int = imaqSetContourColor(ROI* roi, ContourID id, const RGBValue* color)
```

## Purpose

Sets the color of a contour.

## Parameters

| Name | Type | Description |
|---|---|---|
| **roi** | ROI* | The ROI containing the contour whose color the function sets. |
| **id** | ContourID | The ContourID of the contour whose color the function sets. |
| **color** | const RGBValue* | The color to which the function sets the contour. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetROIColor

## Usage

```
int = imaqSetROIColor(ROI* roi, const RGBValue* color)
```

## Purpose

Sets the color of all the contours currently in an ROI. All contours you add to the ROI become this color after you call this function. Use `imaqSetContourColor()` to change the color of individual contours within the ROI.

## Parameters

| Name | Type | Description |
|---|---|---|
| **roi** | ROI* | The ROI whose color the function sets. |
| **color** | const RGBValue* | The color to which the function sets the ROI. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSetWindowROI

## Usage

```
int = imaqSetWindowROI(int windowNumber, const ROI* roi)
```

## Purpose

Sets the ROI associated with a given window.

## Parameters

| Name | Type | Description |
|---|---|---|
| **windowNumber** | int | The window number of the image window. |
| **roi** | const ROI* | The ROI to associate with the window. Set this parameter to NULL to remove any ROIs from the window. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqTransformROI

## Usage

```
int = imaqTransformROI(ROI* roi, Point originStart, float angleStart, Point
                       originFinal, float angleFinal)
```

## Purpose

Rotates and translates an ROI from one coordinate system to another coordinate system within an image.

## Parameters

| Name | Type | Description |
|---|---|---|
| **roi** | ROI* | The ROI to transform. |
| **originStart** | Point | The origin of the starting coordinate system. |
| **angleStart** | float | The angle, in degrees, of the x-axis of the starting coordinate system relative to the x-axis of the image. |
| **originFinal** | Point | The origin of the new coordinate system. |
| **angleFinal** | float | The angle, in degrees, of the x-axis of the new coordinate system relative to the x-axis of the image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# 8

# File I/O

This chapter describes the File I/O functions in IMAQ Vision for LabWindows/CVI. File I/O functions allow you to read images from a hard drive or disk, write images to a hard drive or disk, and get information about images stored on a hard drive or disk.

## File I/O Function Panels

Table 8-1 lists the File I/O functions in a tree structure. The functions in the File I/O class are grouped in alphabetical order according to Class/Panel Name. The first-level heading in the tree is the name of the class. The second-level headings are names of individual function panels. Each File I/O function panel represents one function.

**Table 8-1.** Image Management Function Tree

| Class/Panel Name | Function Name |
|---|---|
| File I/O | |
| Get File Information | `imaqGetFileInfo` |
| Read File | `imaqReadFile` |
| Write BMP File | `imaqWriteBMPFile` |
| Write File | `imaqWriteFile` |
| Write JPEG File | `imaqWriteJPEGFile` |
| Write PNG File | `imaqWritePNGFile` |
| Write TIFF File | `imaqWriteTIFFFile` |

# imaqGetFileInfo

## Usage

```
int = imaqGetFileInfo(const char* filePath, CalibrationUnit*
                      calibrationUnit, float* calibrationX, float*
                      calibrationY, int* width, int* height, ImageType*
                      imageType)
```

## Purpose

Returns information regarding the contents of an image file. You can retrieve information from the following image file formats only: PNG, JPEG, TIFF, AIPD, and BMP.

## Parameters

| Name | Type | Description |
|---|---|---|
| **filePath** | const char* | The name of the file from which the function gets information. This parameter is required and cannot be NULL. |
| **calibrationUnit** | CalibrationUnit* | On return, the calibration unit of the image. If the file does not have calibration information, the function sets **calibrationUnit** to `IMAQ_UNDEFINED`. Set this parameter to NULL if you do not need this information. |
| **calibrationX** | float* | On return, the interpixel distance in the *x* direction. If the file does not have calibration information, the function sets **calibrationX** to 1. Set this parameter to NULL if you do not need this information. |
| **calibrationY** | float* | On return, the interpixel distance in the *y* direction. If the file does not have calibration information, the function sets **calibrationY** to 1. Set this parameter to NULL if you do not need this information. |
| **width** | int* | On return, the width of the image. Set this parameter to NULL if you do not need this information. |
| **height** | int* | On return, the height of the image. Set this parameter to NULL if you do not need this information. |
| **imageType** | ImageType* | On return, the type of the image. Set this parameter to NULL if you do not need this information. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqReadFile

## Usage

```
int = imaqReadFile(Image* image, const char* fileName, RGBValue* colorTable,
                   int* numColors)
```

## Purpose

Creates an IMAQ Vision image from the information in a file. The file can be in one of the following formats: PNG, JPEG, TIFF, AIPD, or BMP.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_COMPLEX`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | Image* | The image in which the function stores data it reads from the file. |
| **fileName** | const char* | The name of the file to read. This parameter is required and cannot be NULL. |
| **colorTable** | RGBValue* | An optional array of up to 256 elements. If the file has a color table, the function fills this parameter with the color table values. If the file does not contain a color table, the function returns an empty array. Set this parameter to NULL if you do not want the function to load the color table. |
| **numColors** | int* | If **colorTable** is not NULL, the function fills this parameter with the number of colors in the color table. If **colorTable** is NULL, the function ignores this parameter. Set this parameter to NULL if you do not need this information. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqWriteBMPFile

## Usage

```
int = imaqWriteBMPFile(const Image* image, const char* fileName, int
                       compress, const RGBValue* colorTable)
```

## Purpose

Writes an image to a BMP file. The function also stores a `CalibrationUnit` of `IMAQ_METER` only. If you pass an image to this function that has a `CalibrationUnit` other than `IMAQ_METER`, the function converts `xStep` and `yStep` from the supplied unit into meters.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_RGB`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image to write to a file. |
| **fileName** | const char* | The name of the file to write. This parameter is required and cannot be NULL. |
| **compress** | int | Set this parameter to TRUE to compress the BMP. Set this parameter to FALSE to write an uncompressed BMP. |
| **colorTable** | const RGBValue* | An optional color table to associate with 8-bit images. If you want to provide a color table, the table must have 256 elements. Set this parameter to NULL to write a grayscale palette to the file. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqWriteFile

## Usage

```
int = imaqWriteFile(const Image* image, const char* fileName, const RGBValue*
                    colorTable)
```

## Purpose

This function writes an image to a file. In addition to writing pixel information, the function writes calibration information if the file format supports calibration information. The following list details file formats that support calibration information.

- AIPD Files—Stores any type of `CalibrationUnit` but stores only one step size. The function stores the `xStep` from the supplied image as the step size.
- BMP Files—Stores a `CalibrationUnit` of `IMAQ_METER` only. If you pass an image to this function that has a `CalibrationUnit` other than `IMAQ_METER`, the function converts `xStep` and `yStep` from the supplied unit into meters.
- JPEG and TIFF Files—Stores a `CalibrationUnit` of `IMAQ_CENTIMETER` or `IMAQ_INCH` only. If you pass an image to this function that has a metric `CalibrationUnit` other than `IMAQ_CENTIMETER`, the function converts `xStep` and `yStep` from the supplied unit into centimeters. If you pass an image to this function that has an English `CalibrationUnit` other than `IMAQ_INCH`, the function converts `xStep` and `yStep` from the supplied unit into inches.
- PNG Files—Stores any type of `CalibrationUnit`.

To write specific file types with more flexibility, use `imaqWriteJPEGFile()`, `imaqWriteBMPFile()`, `imaqWriteTIFFFile()`, or `imaqWritePNGFile()`.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_COMPLEX`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image to write to a file. The function cannot write all image types to all file types. The following are the supported image types for each file type:<br>**Image Types** — **File Types**<br>AIPD — all image types<br>BMP, JPEG, TIFF — 8-bit, RGB<br>PNG — 8-bit, 16-bit, RGB |
| **fileName** | const char* | The name of the file. The file type is determined by the extension, as follows:<br>**Extension** — **File Type**<br>`.aipd` or `.apd` — AIPD<br>`.bmp` — BMP<br>`.jpg` or `.jpeg` — JPEG<br>`.png` — PNG<br>`.tif` or `.tiff` — TIFF<br>This parameter is required and cannot be NULL. |
| **colorTable** | const RGBValue* | An optional color table to associate with 8-bit images. If you provide a color table, the table must have 256 elements. Set this parameter to NULL to write a grayscale palette to the image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqWriteJPEGFile

## Usage

```
int = imaqWriteJPEGFile(const Image* image, const char* fileName, unsigned
                        int quality, const RGBValue* colorTable)
```

## Purpose

Writes an image to a JPEG file. The function also stores a `CalibrationUnit` of `IMAQ_CENTIMETER` or `IMAQ_INCH` only. If you pass an image to this function that has a metric `CalibrationUnit` other than `IMAQ_CENTIMETER`, the function converts `xStep` and `yStep` from the supplied unit into centimeters. If you pass an image to this function that has an English `CalibrationUnit` other than `IMAQ_INCH`, the function converts `xStep` and `yStep` from the supplied unit into inches.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_RGB`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image to write to a file. |
| **fileName** | const char* | The name of the file to write. This parameter is required and cannot be NULL. |
| **quality** | unsigned int | Represents the quality of the image. As quality increases, the function uses less lossy compression. Acceptable values range from 0 to 1,000 with 750 as the default. Note: This function uses lossy compression even if you set the quality to 1,000. |
| **colorTable** | const RGBValue* | An optional color table to associate with 8-bit images. If you provide a color table, the table must have 256 elements. Set this parameter to NULL to write a grayscale palette to the image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqWritePNGFile

## Usage

```
int = imaqWritePNGFile(const Image* image, const char* fileName, unsigned int
                       compressionSpeed, const RGBValue* colorTable)
```

## Purpose

Writes an image to a PNG file. This function stores any type of `CalibrationUnit` in a format that IMAQ Vision can read. This function also converts calibration information into meters, which any PNG file reader can interpret.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_RGB`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image to write to a file. |
| **fileName** | const char* | The name of the file to write. This parameter is required and cannot be NULL. |
| **compressionSpeed** | unsigned int | Represents the relative speed of the compression algorithm. As this value increases, the function spends less time compressing the image. Acceptable values range from 0 to 1,000 with 750 as the default. PNG format always stores images in a lossless fashion. |
| **colorTable** | const RGBValue* | An optional color table to associate with 8-bit images. If you provide a color table, the table must have 256 elements. Set this parameter to NULL to write a grayscale palette to the image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqWriteTIFFFile

## Usage

```
int = imaqWriteTIFFFile(const Image* image, const char* fileName, const
                        TIFFFileOptions* options, const RGBValue*
                        colorTable)
```

## Purpose

Writes an image to a TIFF file. This function stores a `CalibrationUnit` of `IMAQ_CENTIMETER` or `IMAQ_INCH` only. If you pass an image to this function that has a metric `CalibrationUnit` other than `IMAQ_CENTIMETER`, the function converts `xStep` and `yStep` from the supplied unit into centimeters. If you pass an image to this function that has an English `CalibrationUnit` other than `IMAQ_INCH`, the function converts `xStep` and `yStep` from the supplied unit into inches.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_RGB`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image to write to a file. |
| **fileName** | const char* | The name of the file to write. This parameter is required and cannot be NULL. |
| **options** | const TIFFFileOptions* | A structure defining the specific options to use while writing the TIFF file. |
| **colorTable** | const RGBValue* | An optional color table to associate with 8-bit images. If you provide a color table, the table must have 256 elements. Set this parameter to NULL to write a grayscale palette to the image. |

## Parameter Discussion

**options**—The `TIFFFileOptions` structure contains three fields:

- `rowsPerStrip`—Indicates the number of rows that the function writes per strip.
- `photoInterp`—Designates which photometric interpretation to use. The following options are valid:

  `IMAQ_WHITE_IS_ZERO`

  `IMAQ_BLACK_IS_ZERO`

- compressionType—Indicates the type of compression to use on the TIFF. The following options are valid:

  IMAQ_NO_COMPRESSION

  IMAQ_JPEG

  IMAQ_RUN_LENGTH

  IMAQ_ZIP

Set the **options** parameter to NULL to use the default options, as follows:

- rowsPerStrip—0, which writes all data in one strip
- photoInterp—IMAQ_BLACK_IS_ZERO
- compressionType—IMAQ_NO_COMPRESSION

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# 9

# Image Analysis

This chapter describes the Image Analysis functions in IMAQ Vision for LabWindows/CVI. Image Analysis functions allow you to calculate various statistics about the pixels of an image.

## Image Analysis Function Panels

Table 9-1 lists the Image Analysis functions in a tree structure. The functions in the Image Analysis class are grouped in alphabetical order according to Class/Panel Name. The first-level heading in the tree is the name of the class. The second-level headings are names of individual function panels. Each Image Analysis function panel represents one function.

**Table 9-1.** Image Analysis Function Tree

| Class/Panel Name | Function Name |
|---|---|
| Image Analysis | |
| Centroid | `imaqCentroid` |
| Histogram | `imaqHistogram` |
| Linear Averages | `imaqLinearAverages` |
| Line Profile | `imaqLineProfile` |
| Quantify | `imaqQuantify` |

# imaqCentroid

## Usage

```
int = imaqCentroid(const Image* image, PointFloat* centroid, const Image*
                   mask)
```

## Purpose

Computes the centroid of an image.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image whose centroid the function calculates. |
| **centroid** | PointFloat* | On return, the *x* and *y* coordinates of the centroid. This parameter is required, and may not be NULL. |
| **mask** | const Image* | An optional mask image. This image must be an `IMAQ_IMAGE_U8` image. In the calculation, the function uses only those source pixels whose corresponding mask pixels are non-zero. Set this parameter to NULL if you want the function to use the whole image in the centroid calculation. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqHistogram

## Usage

```
HistogramReport* = imaqHistogram(const Image* image, int numClasses, float
                          min, float max, const Image* mask)
```

## Purpose

Calculates the histogram, or pixel value distribution, of an image.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image whose histogram the function calculates. |
| **numClasses** | int | The number of classes into which the function separates the pixels. |
| **min** | float | The minimum pixel value to consider for the histogram. The function does not count pixels with values less than `min`. |
| **max** | float | The maximum pixel value to consider for the histogram. The function does not count pixels with values greater than `max`. |
| **mask** | const Image* | An optional mask image. This image must be an `IMAQ_IMAGE_U8` image. When calculating the histogram, the function considers only those pixels in **image** whose corresponding pixels in **mask** are non-zero. Set this parameter to NULL if you want the function to perform a histogram on the whole image. |

## Return Value

**HistogramReport***—On success, this function returns a report describing the pixel value classification. The `HistogramReport` structure contains the following elements:

- `histogram`—An array describing the number of pixels that fell into each class.
- `histogramCount`—The number of elements in the `histogram` array. The number of elements equals the value you provided in **numClasses**.
- `min`—The smallest pixel value that the function classified.
- `max`—The largest pixel value that the function classified.
- `start`—The smallest pixel value that fell into the first class.

- `width`—The size of each class.
- `mean`—The mean value of the pixels that the function classified.
- `stdDev`—The standard deviation of the pixels that the function classified.
- `numPixels`—The number of pixels that the function classified. The **mask** and the given **min** and **max** influence this element.

On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with the report, dispose of it by calling `imaqDispose()`.

# imaqLinearAverages

## Usage

```
LinearAverages* = imaqLinearAverages(const Image* image, Rect rect)
```

## Purpose

Computes the mean line profile of an image.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image on which the function calculates pixel value averages. |
| **rect** | Rect | Sets the rectangular area in which the function calculates the averages. Set this parameter to `IMAQ_NO_RECT` to calculate the averages on the whole image. |

## Return Value

**LinearAverages***—On success, this function returns a structure containing the linear averages of the image. The `LinearAverages` structure has the following elements:

- `columnAverages`—An array containing the mean pixel value of each column.
- `columnCount`—The number of elements in the `columnAverages` array.
- `rowAverages`—An array containing the mean pixel value of each row.
- `rowCount`—The number of elements in the `rowAverages` array.
- `risingDiagAverages`—An array containing the mean pixel value of each diagonal running from the lower left to the upper right of the area you set in **rect**.
- `risingDiagCount`—The number of elements in the `risingDiagAverages` array.
- `fallingDiagAverages`—An array containing the mean pixel value of each diagonal running from the upper left to the lower right of the area you set in **rect**.
- `fallingDiagCount`—The number of elements in the `fallingDiagAverages` array.

On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with this information, dispose of the pointer by calling `imaqDispose()`.

# imaqLineProfile

## Usage

```
LineProfile* = imaqLineProfile(const Image* image, Point start, Point end)
```

## Purpose

Computes the profile of a line of pixels.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| image | const Image* | The image containing a line whose profile the function computes. |
| start | Point | The first point of the line. |
| end | Point | The last point of the line. |

## Return Value

**LineProfile***—On success, this function returns a report containing information about the line. The `LineProfile` structure contains the following elements:

- `profileData`—An array containing the value of each pixel in the line.
- `boundingRect`—The bounding rectangle of the line.
- `min`—The smallest pixel value in the line profile.
- `max`—The largest pixel value in the line profile.
- `mean`—The mean value of the pixels in the line profile.
- `stdDev`—The standard deviation of the line profile.
- `dataCount`—The size of the `profileData` array.

On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with the line profile, dispose of it by calling `imaqDispose()`.

# imaqQuantify

## Usage

```
QuantifyReport* = imaqQuantify(const Image* image, const Image* mask)
```

## Purpose

Calculates statistical parameters on an image.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image to quantify. |
| **mask** | const Image* | If provided, a labeled version of the source image specifying the regions to quantify. This image must be an IMAQ_IMAGE_U8 image. Set this parameter to NULL if you want the function to quantify the whole image as one region. |

## Return Value

**QuantifyReport***—On success, this function returns a pointer to a report describing the statistical parameters of the image. The QuantifyReport structure consists of the following parameters:

- global—A QuantifyData structure containing statistical data of the whole image.
- regions—An array of QuantifyData structures containing statistical data of each region of the image. See the **mask** parameter for more information on the regions.
- regionCount—The number of regions.

The elements global and regions are QuantifyData structures. A QuantifyData structure consists of the following elements:

- mean—The mean value of the pixel values.
- stdDev—The standard deviation of the pixel values.
- min—The smallest pixel value.
- max—The largest pixel value.
- calibratedArea—The area, calibrated to the calibration information of the image.
- pixelArea—The area, in number of pixels.

- `relativeSize`—The proportion, expressed as a percentage, of the associated region relative to the whole image.

On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with this information, dispose of the pointer by calling `imaqDispose()`.

# 10

# Grayscale Processing

This chapter describes the Grayscale Processing functions in IMAQ Vision for LabWindows/CVI.

## Grayscale Processing Function Panels

Table 10-1 lists the Grayscale Processing functions in a tree structure. The functions in the Grayscale Processing class are grouped according to the types of operations they perform. The first-level heading in the tree is the name of the class. The second-level headings are names of function subclasses. The third-level headings are names of individual function panels. Each Grayscale Processing function panel represents one function.

**Table 10-1.** Grayscale Processing Function Tree

| Class/Panel Name | Function Name |
|---|---|
| Grayscale Processing | |
| Threshold | |
| Automatic Threshold | `imaqAutoThreshold` |
| Magic Wand | `imaqMagicWand` |
| Multithreshold | `imaqMultithreshold` |
| Threshold | `imaqThreshold` |
| Transform | |
| BCG Transform | `imaqBCGTransform` |
| Equalize | `imaqEqualize` |
| Inverse | `imaqInverse` |
| Lookup | `imaqLookup` |
| Math Transform | `imaqMathTransform` |
| Spatial Filters | |
| Canny Edge Filter | `imaqCannyEdgeFilter` |
| Convolve | `imaqConvolve` |
| Correlate | `imaqCorrelate` |
| Edge Filter | `imaqEdgeFilter` |
| Lowpass | `imaqLowpass` |
| Median Filter | `imaqMedianFilter` |
| Nth Order Filter | `imaqNthOrderFilter` |
| Morphology | |
| Gray Morphology | `imaqGrayMorphology` |

# Subclass Descriptions

Grayscale Processing functions enhance grayscale images for viewing or further processing. These functions usually modify the pixel values of the images they enhance. Grayscale Processing subclass descriptions are as follows:

- Threshold functions allow you to convert a grayscale image to a binary image.
- Transform functions allow you to replace each pixel in an image using a transfer function.
- Spatial Filter functions allow you to modify an image using neighborhood functions.
- Morphology functions allow you to apply standard morphological transformations, such as dilations and erosions.

# imaqAutoThreshold

## Usage

```
ThresholdData* = imaqAutoThreshold(Image* dest, Image* source, int
                         numClasses, ThresholdMethod method)
```

## Purpose

Automatically thresholds an image into multiple classes.

## Image Type Supported

IMAQ_IMAGE_U8

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | Image* | The image to threshold. |
| **numClasses** | int | The number of classes into which to threshold the image. Valid values range from 2 to 256. |
| **method** | ThresholdMethod | The method for binary thresholding. If **numClasses** is 2 (a binary threshold), **method** specifies how to calculate the classes, as follows:<br>IMAQ_THRESH_CLUSTERING<br>IMAQ_THRESH_ENTROPY<br>IMAQ_THRESH_METRIC<br>IMAQ_THRESH_MOMENTS<br>IMAQ_THRESH_INTERCLASS<br>If **numClasses** is not 2, the function ignores this parameter. For a description of how each method works, see the *IMAQ Vision User Manual*. |

## Return Value

**ThresholdData***—On success, this function returns an array of structures providing information about the threshold ranges that the function applied. The array contains a number of ThresholdData structures equal to **numClasses**. Each ThresholdData element in the array contains the following fields:

- rangeMin—The lower boundary of the range to keep.
- rangeMax—The upper boundary of the range to keep.

- useNewValue—If TRUE, the function sets pixel values within [rangeMin, rangeMax] to the value specified in newValue.
- newValue—The replacement value for pixels within the range.

On failure, this function returns NULL. To get extended error information, call imaqGetLastError(). When you are finished with the array, dispose of it by calling imaqDispose().

# imaqBCGTransform

## Usage

```
int = imaqBCGTransform(Image* dest, const Image* source, const BCGOptions*
                       options, const Image* mask)
```

## Purpose

Applies brightness, contrast, and gamma correction to an image by computing and applying a lookup table. The function computes the lookup table based on the values in BCGOptions.

## Image Type Supported

`IMAQ_IMAGE_U8`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The image to transform. |
| **options** | const BCGOptions* | The parameters to use in the transform. This parameter is required and cannot be NULL. |
| **mask** | const Image* | An optional mask image. This image must be an `IMAQ_IMAGE_U8` image. The function applies the transform to only those source pixels whose corresponding mask pixels are non-zero. All other pixel remain unchanged. Set this parameter to NULL to apply the transform to the whole source image. |

## Parameter Description

**options**—A `BCGOptions` structure contains the following elements:

- `brightness`—Adjusts the brightness of the image. A value of 128 leaves the brightness unchanged. Values below 128 darken the image, and values above 128 brighten the image.
- `contrast`—Adjusts the contrast of the image. A value of 45 leaves the contrast unchanged. Values below 45 decrease the contrast, and values above 45 increase the contrast.
- `gamma`—Performs gamma correction. A value of 1.0 leaves the image unchanged. Values below 1.0 enhance contrast for darker pixel at the expense of the brighter pixels. Values above 1.0 enhance contrast for brighter pixels at the expense of darker pixels.

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqCannyEdgeFilter

## Usage

```
int = imaqCannyEdgeFilter(Image* dest, const Image* source, const
                          CannyOptions* options)
```

## Purpose

Outlines edges in an image using the Canny algorithm, which is well-suited to images with poor signal-to-noise ratios.

## Image Type Supported

`IMAQ_IMAGE_U8`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The image whose edges the function outlines. |
| **options** | const CannyOptions* | A description of filter parameters to use in the algorithm. |

## Parameter Discussion

**options**—A `CannyOptions` structure contains the following elements:

- `sigma`—The sigma of the Gaussian smoothing filter that the function applies to the image before edge detection.
- `upperThreshold`—The upper fraction of pixel values in the image from which the function chooses a seed or starting point of an edge segment. This value must be between 0 and 1.
- `lowerThreshold`—The function multiplies this value by `upperThreshold` to determine the lower threshold for all the pixels in an edge segment.
- `windowSize`—The window size of the Gaussian filter that the function applies to the image. This value must be odd.

Set the **options** parameter to NULL to use the default options, as follows:

- `sigma`—1.00
- `upperThreshold`—0.70
- `lowerThreshold`—0.20
- `windowSize`—9

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqConvolve

## Usage

```
int = imaqConvolve(Image* dest, Image* source, const float* kernel, int
                   matrixRows, int matrixCols, float normalize,
                   const Image* mask)
```

## Purpose

Applies a linear filter to an image by convolving the image with a filtering kernel. The convolution kernel must have an odd width and height.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | Image* | The image to filter. This function modifies the border of the source image. The border must be at least half as large as the larger dimension of the kernel. |
| **kernel** | const float* | The matrix representing the linear filter. This parameter is required and cannot be NULL. |
| **matrixRows** | int | The number of rows in the kernel matrix. This number must be odd. |
| **matrixCols** | int | The number of columns in the kernel matrix. This number must be odd. |
| **normalize** | float | The normalization factor. After performing the convolution, the function divides each pixel value by this value. Set this parameter to 0 to divide by the sum of the elements of the kernel. |
| **mask** | const Image* | An optional mask image. This image must be an IMAQ_IMAGE_U8 image. The function filters only those pixels in the source image whose corresponding pixels in the mask are non-zero. Set this parameter to NULL to filter the entire image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqCorrelate

## Usage

```
int = imaqCorrelate(Image* dest, Image* source, const Image* templateImage,
                    Rect rect)
```

## Purpose

Computes the normalized cross-correlation between a source image and a template image. This operation is time-intensive. To reduce the correlation time, use a small template, reduce the search area by using the area rectangle, and make the template image width a multiple of 4.

## Image Type Supported

`IMAQ_IMAGE_U8`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | Image* | The source image. The correlation modifies the border of the source image. The border must be at least half as large as the larger dimension of the template image. |
| **templateImage** | const Image* | The template image to correlate against the source. |
| **rect** | Rect | The area of the source image on which to perform the correlation. Set this parameter to `IMAQ_NO_RECT` to perform the correlation on the whole source image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqEdgeFilter

## Usage

```
int = imaqEdgeFilter(Image* dest, Image* source, OutlineMethod method, const
                     Image* mask)
```

## Purpose

Applies a nonlinear filter to highlight edges.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | Image* | The image of which the function highlights edges. The convolution modifies the border of the source image. The border must be at least one pixel wide. |
| **method** | OutlineMethod | Method to use when outlining the edges. The following options are valid:<br>`IMAQ_EDGE_DIFFERENCE`<br>`IMAQ_EDGE_GRADIENT`<br>`IMAQ_EDGE_PREWITT`<br>`IMAQ_EDGE_ROBERTS`<br>`IMAQ_EDGE_SIGMA`<br>`IMAQ_EDGE_SOBEL`<br>For more information about filtering, see the *IMAQ Vision User Manual*. |
| **mask** | const Image* | An optional mask image. This image must be an `IMAQ_IMAGE_U8` image. The function filters only those pixels in the source image whose corresponding pixels in the mask are non-zero. Set this parameter to NULL to filter the whole source image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqEqualize

## Usage

```
int = imaqEqualize(Image* dest, const Image* source, float min, float max,
                   const Image* mask)
```

## Purpose

Calculates the histogram of an image and redistributes pixel values across the desired range to maintain the same pixel value distribution. Pixels whose values are the same before the redistribution also have common pixel values after the redistribution.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **min** | float | The minimum value of the range to which the function equalizes the image. |
| **max** | float | The maximum value of the range to which the function equalizes the image. |
| **mask** | const Image* | An optional mask image. This image must be an `IMAQ_IMAGE_U8` image. The function equalizes only those pixels in the source image whose corresponding pixels in the mask are non-zero. Set this parameter to NULL to equalize the whole image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGrayMorphology

## Usage

```
int = imaqGrayMorphology(Image* dest, Image* source, MorphologyMethod
                         method, const StructuringElement*
                         structuringElement)
```

## Purpose

Applies morphological transformations to gray level images.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | Image* | The image on which the function performs the morphological operation. The calculation modifies the border of the source image. The border must be at least half as large as the larger of the dimensions of the structuring element. |
| **method** | MorphologyMethod | The morphological transformation to apply. For more information about morphological transformations, see the *IMAQ Vision User Manual*. |
| **structuringElement** | const StructuringElement* | The structuring element that the function uses in the operation. Set this parameter to NULL if you do not want a custom structuring element. For more information about structuring elements, see Chapter 1, *Basic Concepts*, or see the *IMAQ Vision User Manual*. |

## Parameter Discussion

**method**—The following options are valid: IMAQ_AUTOM, IMAQ_CLOSE, IMAQ_DILATE, IMAQ_ERODE, IMAQ_GRADIENT, IMAQ_GRADIENTOUT, IMAQ_GRADIENTIN, IMAQ_HITMISS, IMAQ_OPEN, IMAQ_PCLOSE, IMAQ_POPEN, IMAQ_THICK, IMAQ_THIN.

**structuringElement**—The following fields are valid:

- `matrixCols`—Number of columns in the matrix.
- `matrixRows`—Number of rows in the matrix.
- `hexa`—Set this field to TRUE if you specify a hexagonal structuring element in `kernel`. Set this field to FALSE if you specify a square structuring element in `kernel`.
- `kernel`—The values of the structuring element. Specify these values in order from the top-left of the kernel to the bottom-right of the kernel. For example, to set up the $3 \times 3$ kernel.

| | | |
|---|---|---|
| –1 | 0 | 1 |
| –1 | 0 | 1 |
| –1 | 0 | 1 |

called `myStructuringElement`, use the following syntax:

```
myStructuringElement.matrixCols = 3
myStructuringElement.matrixRows = 3
myStructuringElement.hexa = FALSE
myStructuringElement.kernel = malloc(9 * sizeof(int))
myStructuringElement.kernel[0] = –1
myStructuringElement.kernel[1] = 0
myStructuringElement.kernel[2] = 1
myStructuringElement.kernel[3] = –1
myStructuringElement.kernel[4] = 0
myStructuringElement.kernel[5] = 1
myStructuringElement.kernel[6] = –1
myStructuringElement.kernel[7] = 0
myStructuringElement.kernel[8] = 1
```

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqInverse

## Usage

```
int = imaqInverse(Image* dest, const Image* source, const Image* mask)
```

## Purpose

Inverts the pixel intensities of an image using the following equation:

$$f(p) = dynamicMax - p + dynamicMin$$

*where* *p* represents the value of a pixel.
*dynamicMin* represents 0 (8-bit images) or the smallest pixel value in the source image (16-bit and floating point images).
*dynamicMax* represents 255 (8-bit images) or the largest pixel value in the source image (16-bit and floating point images).

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The image to invert. |
| **mask** | const Image* | An optional mask image. This image must be an `IMAQ_IMAGE_U8` image. The function inverts only those pixels in the source image whose corresponding pixels in the mask are non-zero. Set this parameter to NULL to invert the pixel intensities of the entire source image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqLookup

## Usage

```
int = imaqLookup(Image* dest, const Image* source, const short* table, const
                         Image* mask)
```

## Purpose

Performs a transformation on an image by replacing each pixel value with the lookup table entry corresponding to that value.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **table** | const short* | The lookup table. For 8-bit images, the lookup table must contain 256 elements. The function replaces each pixel value `v` with `table[v]`. For 16-bit images, the lookup table must contain 65,536 elements. The function replaces each non-negative pixel value `v` with `table[v]` and replaces each negative pixel value `v` with `table[65536+v]`. |
| **mask** | const Image* | An optional mask image. This image must be an `IMAQ_IMAGE_U8` image. The function applies the lookup only to those source pixels whose corresponding mask pixels are non-zero. All other pixels remain unchanged. Set this parameter to NULL to apply the lookup to the entire source image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqLowpass

## Usage

```
int = imaqLowpass(Image* dest, Image* source, int width, int height, float
                          tolerance, const Image* mask)
```

## Purpose

Filters an image using a non-linear filter. For each pixel, the algorithm considers the neighborhood specified by the given filter sizes. If the current pixel value varies from the value of its neighbors more than the specified tolerance, the function sets the pixel value to the average value of its neighborhood. If the current pixel value varies from the value of its neighbors less than the specified tolerance, the function does not change the value of the pixel.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | Image* | The image to filter. The filter modifies the border of the source image. The border must be at least half as large as the larger of the neighborhood dimensions. |
| **width** | int | The width of the rectangular neighborhood around the pixel on which the function operates. This number must be odd. |
| **height** | int | The height of the rectangular neighborhood around the pixel on which the function operates. This number must be odd. |
| **tolerance** | float | The maximum allowable variance. |
| **mask** | const Image* | An optional mask image. This image must be an `IMAQ_IMAGE_U8` image. The function filters only those pixels in the source image whose corresponding pixels in the mask are non-zero. Set this parameter to NULL to apply the filter to the entire source image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqMagicWand

## Usage

```
int = imaqMagicWand(Image* dest, const Image* source, Point coord, float
                    tolerance, int connectivity8, float replaceValue)
```

## Purpose

Creates a mask of a particle in an image by selecting a particle at the given location and setting all the pixels of that particle to a specified value. The function sets all other pixel values to 0.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | On return, the mask image. This image must be an `IMAQ_IMAGE_U8` image. |
| **source** | const Image* | The source image containing the particle to mask. |
| **coord** | Point | The coordinates of the reference point in the particle to mask. |
| **tolerance** | float | Specifies the pixel value tolerance that the function uses to determine whether neighbors of the reference point are part of the particle. |
| **connectivity8** | int | Set this parameter to TRUE to use connectivity-8 to determine whether pixels are part of the same particle. Set this parameter to FALSE to use connectivity-4 to determine whether pixels are part of the same particle. For more information about connectivity, see Chapter 1, *Basic Concepts*, or see the *IMAQ Vision User Manual*. |
| **replaceValue** | float | The value to which the function sets pixels in the selected particle. The function sets pixels not in the particle to 0. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqMathTransform

## Usage

```
int = imaqMathTransform(Image* dest, const Image* source,
                        MathTransformMethod method, float rangeMin, float
                        rangeMax, float power, const Image* mask)
```

## Purpose

Transforms an image by applying a transfer function to the value of each pixel. The function applies the transform T(*x*) over a specified input range [rangeMin, rangeMax] in the following manner:

T(*x*) = dynamicMin if *x* <= rangeMin

*f*(*x*) if rangeMin < *x* <= rangeMax

dynamicMax if *x* > rangeMax

*where* dynamicMin = 0 (8-bit images) or the smallest initial pixel value (16-bit and floating point images)

dynamicMax = 255 (8-bit images) or the largest initial pixel value (16-bit and floating point images)

dynamicRange = dynamicMax – dynamicMin

The function scales *f*(*x*) so that *f*(rangeMin) = dynamicMin and *f*(rangeMax) = dynamicMax. *f*(*x*) behaves on (rangeMin, rangeMax) according to the method you select.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **method** | MathTransformMethod | The transform function to use. |
| **rangeMin** | float | The smallest pixel value on which the function applies the transform. |

| Name | Type | Description |
|---|---|---|
| **rangeMax** | float | The largest pixel value on which the function applies the transform. |
| **power** | float | If you set **method** to `IMAQ_TRANSFORM_POWX` or `IMAQ_TRANSFORM_POW1X`, **power** specifies the power to which the function raises the value. |
| **mask** | const Image* | An optional mask. This image must be an `IMAQ_IMAGE_U8` image. The function transforms only those source pixels whose corresponding mask pixels are non-zero. All other pixels remain unchanged. Set this parameter to NULL to apply the transform to the entire source image. |

## Parameter Discussion

**method**—The following options are valid:

- `IMAQ_TRANSFORM_LINEAR`—Linear remapping.
- `IMAQ_TRANSFORM_LOG`—Logarithmic remapping. Enhances contrast for small pixel values and reduces contrast for large pixel values.
- `IMAQ_TRANSFORM_EXP`—Exponential remapping. Enhances contrast for large pixel values and reduces contrast for small pixel values.
- `IMAQ_TRANSFORM_SQR`—Square remapping. Similar to exponential remapping but with a more gradual effect.
- `IMAQ_TRANSFORM_SQRT`—Square root remapping. Similar to logarithmic remapping but with a more gradual effect.
- `IMAQ_TRANSFORM_POWX`—Power X remapping. Causes variable effect depending on `power`.
- `IMAQ_TRANSFORM_POW1X`—Power 1/X remapping. Causes variable effect depending on `power`.

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqMedianFilter

## Usage

```
int = imaqMedianFilter(Image* dest, Image* source, int width, int height,
                       const Image* mask)
```

## Purpose

Filters an image using a non-linear filter. For each pixel, the algorithm takes the neighborhood specified by the given filter sizes and replaces the pixel with the median value of the neighborhood.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | Image* | The image to filter. The filter modifies the border of the source image. The border must be at least half as large as the larger of the neighborhood dimensions. |
| **width** | int | The width of the rectangular neighborhood around the pixel on which the function operates. This number must be odd. |
| **height** | int | The height of the rectangular neighborhood around the pixel on which the function operates. This number must be odd. |
| **mask** | const Image* | An optional mask. This image must be an `IMAQ_IMAGE_U8` image. The function filters only those source pixels whose corresponding mask pixels are non-zero. Set this parameter to NULL to apply the filter to the entire source image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqMultithreshold

## Usage

```
int = imaqMultithreshold(Image* dest, const Image* source, const
                         ThresholdData* ranges, int numRanges)
```

## Purpose

Thresholds an image into multiple classes. The function classifies each pixel into the first threshold range of which it is a member. If a pixel is not a member of any of the given ranges, the function sets it to 0. For example, given two threshold ranges:

| rangeMin | rangeMax | useNewValue | newValue |
|---|---|---|---|
| 80 | 150 | TRUE | 10 |
| 120 | 200 | FALSE | <ignored> |

The function operates as follows:

- The function replaces pixel values below 80 with 0.
- The function replaces pixel values from 80 to 150 with 10.
- The function does not change pixel values from 151 to 200.
- The function replaces pixel values above 200 with 0.

For more information on thresholding, see `imaqThreshold()`.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **ranges** | const ThresholdData* | An array of threshold data. This array is required and cannot be NULL. |
| **numRanges** | int | The number of elements in the `ranges` array. |

## Parameter Discussion

**ranges**—Each element in the **ranges** array is a structure with the following fields:

- `rangeMin`—The lower boundary of the range of pixel values to keep.
- `rangeMax`—The upper boundary of the range of pixel values to keep.
- `useNewValue`—Set this field to TRUE to set the pixel values within [`rangeMin`, `rangeMax`] to the value specified in `newValue`. Set this field to FALSE to leave the pixel values unchanged.
- `newValue`—If you set `useNewValue` to TRUE, `newValue` is the replacement value for pixels within the range. If you set `useNewField` to FALSE, the function ignores this field.

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqNthOrderFilter

## Usage

```
int = imaqNthOrderFilter(Image* dest, Image* source, int width, int height,
                         int n, const Image* mask)
```

## Purpose

Filters an image using a non-linear filter. For each pixel, the algorithm takes the neighborhood specified by the given filter sizes and replaces the pixel with the *n*th smallest value in the neighborhood.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | Image* | The image to filter. The filter modifies the border of the source image. The border must be at least half as large as the larger of the neighborhood dimensions. |
| **width** | int | The width of the rectangular neighborhood around the pixel on which the function operates. This number must be odd. |
| **height** | int | The height of the rectangular neighborhood around the pixel on which the function operates. This number must be odd. |
| **n** | int | Specifies which value in the neighborhood to place in the destination. Set **n** to 0 to select the smallest value in the neighborhood, set **n** to 1 to select the next smallest value, and so on. |
| **mask** | const Image* | An optional image mask. This image must be an `IMAQ_IMAGE_U8` image. The function filters only those pixels in the source image whose corresponding pixels in the mask are non-zero. Set this parameter to NULL to filter the entire source image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqThreshold

## Usage

```
int = imaqThreshold(Image* dest, const Image* source, float rangeMin, float
                    rangeMax, int useNewValue, float newValue)
```

## Purpose

Thresholds an image. The function sets pixels values outside of the given range to 0. The function sets pixel values within the range to a given value or leaves the values unchanged.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **rangeMin** | float | The lower boundary of the range of pixel values to keep. |
| **rangeMax** | float | The upper boundary of the range of pixel values to keep. |
| **useNewValue** | int | Set this parameter to TRUE to set the pixel values within [**rangeMin**, **rangeMax**] to the value specified in **newValue**. Set this field to FALSE to leave the pixel values unchanged. |
| **newValue** | float | If you set **useNewValue** to TRUE, **newValue** is the replacement value for pixels within the range. If you set **useNewValue** to FALSE, the function ignores this parameter. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# 11

# Binary Processing

This chapter describes the Binary Processing functions in IMAQ Vision for LabWindows/CVI. Use Binary Processing functions on binary and labeled images for applications in which the size, number, or shape of the objects in the image are important. Binary images have only two pixel values, unless you label the image.

**Note** Apply a threshold to a grayscale image to make an image binary. For more information about thresholding an image, see `imaqThreshold()` in Chapter 10, *Grayscale Processing*.

## Binary Processing Function Panels

Table 11-1 lists the Binary Processing functions in a tree structure. The functions in the Binary Processing class are grouped according to the types of operations they perform. The first-level heading in the tree is the name of the class. The second-level headings are names of function subclasses. The third-level headings are names of individual function panels. Each Binary Processing function panel represents one function.

**Table 11-1.** Binary Processing Function Tree

| Class/Panel Name | Function Name |
|---|---|
| Binary Processing | |
| Morphology | |
| Circles | `imaqCircles` |
| Convex | `imaqConvex` |
| Danielsson Distance | `imaqDanielssonDistance` |
| Fill Holes | `imaqFillHoles` |
| Label | `imaqLabel` |
| Morphology | `imaqMorphology` |
| Reject Border | `imaqRejectBorder` |
| Segmentation | `imaqSegmentation` |
| Separation | `imaqSeparation` |
| Simple Distance | `imaqSimpleDistance` |
| Size Filter | `imaqSizeFilter` |
| Skeleton | `imaqSkeleton` |
| Shape Matching | |
| Match Shape | `imaqMatchShape` |

**Table 11-1.** Binary Processing Function Tree (Continued)

| | |
|---|---|
| Blob Analysis | |
| Calculate Coefficient | `imaqCalcCoeff` |
| Get Particle Information | `imaqGetParticleInfo` |
| Particle Filter | `imaqParticleFilter` |
| Select Particles | `imaqSelectParticles` |

# Subclass Descriptions

Binary Processing subclass descriptions are as follows:

- Morphology functions allow you to apply standard morphological transformations. For a description of many of these transformations, see the *IMAQ Vision User Manual*.
- Shape Matching functions allow you to find shapes in an image.
- Blob Analysis functions allow you to calculate information about particles (or blobs) in an image and select particles using the information.

# imaqCalcCoeff

## Usage

```
int = imaqCalcCoeff(const Image* image, const ParticleReport* report,
                    MeasurementValue parameter, float* coefficient)
```

## Purpose

Returns a coefficient associated with a particle. Call `imaqGetParticleInfo()` before calling `imaqCalcCoeff()` to get the particle reports you need to pass to this function.

## Image Type Supported

`IMAQ_IMAGE_U8`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image containing the particle. |
| **report** | const ParticleReport* | The particle report you want to use to calculate the coefficient. You must generate this report using `imaqGetParticleInfo()` with the **mode** parameter set to `IMAQ_ALL_INFO`. This parameter is required and cannot be NULL. |
| **parameter** | MeasurementValue | The coefficient to calculate. |
| **coefficient** | float* | On return, the coefficient you request. This parameter is required and cannot be NULL. |

## Parameter Discussion

**parameter**—The following options are valid:

- `IMAQ_AREA`—Surface area of the particle in pixels.
- `IMAQ_AREA_CALIBRATED`—Surface area of the particle in calibrated units.
- `IMAQ_NUM_HOLES`—Number of holes in the particle.
- `IMAQ_AREA_OF_HOLES`—Surface area of the holes in calibrated units.
- `IMAQ_TOTAL_AREA`—Total surface area (holes and particle) in calibrated units.
- `IMAQ_IMAGE_AREA`—Surface area of the entire image in calibrated units.
- `IMAQ_TOTAL_TO_IMAGE`—Ratio, expressed as a percent, of the total surface area (holes and particle) in relation to the image.

- IMAQ_PARTICLE_TO_IMAGE—Ratio, expressed as a percent, of the surface area of a particle in relation to the image area.
- IMAQ_CENTER_MASS_X—X-coordinate of the center of gravity.
- IMAQ_CENTER_MASS_Y—Y-coordinate of the center of gravity.
- IMAQ_LEFT_COLUMN—Left edge of the bounding rectangle.
- IMAQ_TOP_ROW—Top edge of the bounding rectangle.
- IMAQ_RIGHT_COLUMN—Right edge of the bounding rectangle.
- IMAQ_BOTTOM_ROW—Bottom edge of the bounding rectangle.
- IMAQ_WIDTH—Width of bounding rectangle in calibrated units.
- IMAQ_HEIGHT—Height of bounding rectangle in calibrated units.
- IMAQ_MAX_SEGMENT_LENGTH—Length of longest horizontal line segment.
- IMAQ_MAX_SEGMENT_LEFT_COLUMN—Leftmost x-coordinate of longest horizontal line segment.
- IMAQ_MAX_SEGMENT_TOP_ROW—Y-coordinate of longest horizontal line segment.
- IMAQ_PERIMETER—Outer perimeter of the particle.
- IMAQ_PERIMETER_OF_HOLES—Perimeter of all holes within the particle.
- IMAQ_SIGMA_X—Sum of the particle pixels on the x-axis.
- IMAQ_SIGMA_Y—Sum of the particle pixels on the y-axis.
- IMAQ_SIGMA_XX—Sum of the particle pixels on the x-axis squared.
- IMAQ_SIGMA_YY—Sum of the particle pixels on the y-axis squared.
- IMAQ_SIGMA_XY—Sum of the particle pixels on the x-axis and y-axis.
- IMAQ_PROJ_X—Projection corrected in X.
- IMAQ_PROJ_Y—Projection corrected in Y.
- IMAQ_INERTIA_XX—Inertia matrix coefficient in XX.
- IMAQ_INERTIA_YY—Inertia matrix coefficient in YY.
- IMAQ_INERTIA_XY—Inertia matrix coefficient in XY.
- IMAQ_MEAN_H—Mean length of horizontal segments.
- IMAQ_MEAN_V—Mean length of vertical segments.
- IMAQ_MAX_INTERCEPT—Length of longest segment of the convex hull.
- IMAQ_MEAN_INTERCEPT—Mean length of the chords in a particle perpendicular to its max intercept.
- IMAQ_ORIENTATION—Direction of the longest segment.
- IMAQ_EQUIV_ELLIPSE_MINOR—Total length of the axis of the ellipse having the same area as the particle and a major axis equal to half the max intercept.

- IMAQ_ELLIPSE_MAJOR—Total length of the major axis having the same area and perimeter as the particle in calibrated units.
- IMAQ_ELLIPSE_MINOR—Total length of the minor axis having the same area and perimeter as the particle in calibrated units.
- IMAQ_ELLIPSE_RATIO—Fraction of the major axis to the minor axis.
- IMAQ_RECT_LONG_SIDE—Length of the long side of a rectangle having the same area and perimeter as the particle in calibrated units.
- IMAQ_RECT_SHORT_SIDE—Length of the short side of a rectangle having the same area and perimeter as the particle in calibrated units.
- IMAQ_RECT_RATIO—Ratio of rectangle long side to rectangle short side.
- IMAQ_ELONGATION—Max intercept/mean perpendicular intercept.
- IMAQ_COMPACTNESS—Particle area/(height × width).
- IMAQ_HEYWOOD—Particle perimeter/perimeter of the circle having the same area as the particle.
- IMAQ_TYPE_FACTOR—A complex factor relating the surface area to the moment of inertia.
- IMAQ_HYDRAULIC—Particle area/particle perimeter.
- IMAQ_WADDLE_DISK—Diameter of the disk having the same area as the particle in user units.
- IMAQ_DIAGONAL—Diagonal of an equivalent rectangle in user units.

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqCircles

## Usage

```
CircleReport* = imaqCircles(Image* dest, const Image* source, float
                            minRadius, float maxRadius, int* numCircles)
```

## Purpose

Separates overlapping circular objects and classifies them depending on their radii. This function also draws the detected circles into the destination image.

## Image Types Supported

IMAQ_IMAGE_U8

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | On return, an image containing circles that the function located. |
| **source** | const Image* | The image in which the function finds circles. |
| **minRadius** | float | The smallest radius (in pixels) to be detected. Circles with radii smaller than this value do not appear in the destination image. These circles are in the returned report array, but the function reports their radii as negative. |
| **maxRadius** | float | The largest radius (in pixels) to be detected. Circles with radii larger than this value do not appear in the destination image. These circles are in the returned report array, but the function reports their radii as negative. |
| **numCircles** | int* | On return, the number of circles that the function detected in the image. If any circles fall outside the radius range, **numCircles** will be greater than the number of circles that the function draws in the destination image. Set this parameter to NULL if you do not need this information. |

## Return Value

**CircleReport***—On success, this function returns an array of structures containing information about each of the found circles. Each `CircleReport` structure consists of the following elements:

- `center`—The coordinate point of the center of the circle.
- `radius`—The radius of the circle, in pixels. If the radius of the circle is not within the given range, this value is negative.
- `area`—The area of the circle, in pixels.

On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with the array, dispose of it by calling `imaqDispose()`.

# imaqConvex

## Usage

```
int = imaqConvex(Image* dest, const Image* source)
```

## Purpose

Computes the convex envelope for each labeled particle in the source image. If the source image contains more than one particle, you must label each particle with `imaqLabel()` before calling this function.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The image containing the labeled particles whose convex envelopes the function calculates. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqDanielssonDistance

## Usage

```
int = imaqDanielssonDistance(Image* dest, Image* source)
```

## Purpose

Creates a very accurate distance map based on the Danielsson distance algorithm. The function encodes the pixel value of a particle as a function of the distance of the pixel from the particle perimeter. For a faster but less precise algorithm, use `imaqSimpleDistance()`.

## Image Type Supported

`IMAQ_IMAGE_U8`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | Image* | The image that the function uses to compute the distance map. The function modifies the border of the source image. The border must be at least one pixel wide. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqFillHoles

## Usage

```
int = imaqFillHoles(Image* dest, const Image* source, int connectivity8)
```

## Purpose

Fills holes in particles. The function fills the holes with a pixel value of 1. The function does not fill areas touching the edge of the image that appear to be holes because these areas could be either holes or areas of concavity.

## Image Type Supported

`IMAQ_IMAGE_U8`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The image containing particles with holes. |
| **connectivity8** | int | Set this parameter to TRUE to use connectivity-8 to determine whether particles are touching. Set this parameter to FALSE to use connectivity-4 to determine whether particles are touching. For more information about connectivity, see Chapter 1, *Basic Concepts*, or see the *IMAQ Vision User Manual*. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetParticleInfo

## Usage

```
ParticleReport* = imaqGetParticleInfo(Image* image, int connectivity8,
                          ParticleInfoMode mode, int* reportCount)
```

## Purpose

Calculates various measurements of particles in a binary image.

## Image Type Supported

`IMAQ_IMAGE_U8`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | Image* | The binary image containing particles on which the function gets particle information. The calculation modifies the border of the image. |
| **connectivity8** | int | Set this parameter to TRUE to use connectivity-8 to determine whether particles are touching. Set this parameter to FALSE to use connectivity-4 to determine whether particles are touching. For more information about connectivity, see Chapter 1, *Basic Concepts*, or see the *IMAQ Vision User Manual*. |
| **mode** | ParticleInfoMode | Controls the extent of particle information that the function returns. Set this parameter to `IMAQ_ALL_INFO` to return all the information about each particle. Set this parameter to `IMAQ_BASIC_INFO` to get faster results by filling in only the following elements of each report: area, calibratedArea, boundingRect. |
| **reportCount** | int* | On return, filled with the number of reports in the array returned by the function. Set this parameter to NULL if you do not need this information. |

## Return Value

**ParticleReport***—On success, this function returns a pointer to an array of reports about the particles in the image. Each report consists of the following elements:

- `area`—The number of pixels in the particle.
- `calibratedArea`—The size of the particle, calibrated to the calibration information of the image.
- `perimeter`—The length of the perimeter, calibrated to the calibration information of the image.
- `numHoles`—The number of holes in the particle.
- `areaOfHoles`—The total surface area, in pixels, of all the holes in a particle.
- `perimeterOfHoles`—The length of the perimeter of all the holes in the particle calibrated to the calibration information of the image.
- `boundingRect`—The smallest rectangle that encloses the particle.
- `sigmaX`—The sum of the particle pixels on the x-axis.
- `sigmaY`—The sum of the particle pixels on the y-axis.
- `sigmaXX`—The sum of the particle pixels on the x-axis squared.
- `sigmaYY`—The sum of the particle pixels on the y-axis squared.
- `sigmaXY`—The sum of the particle pixels on the x-axis and y-axis.
- `longestLength`—The length, in pixels, of the longest segment in the convex hull of the particle.
- `longestPoint`—The location of the leftmost pixel of the longest segment in the particle.
- `projectionX`—Equals half of the sum of the horizontal segments in a particle that do not overlap another adjacent horizontal segment.
- `projectionY`—Equals half of the sum of the vertical segments in a particle that do not overlap another adjacent vertical segment.

On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with this data, dispose of the array by calling `imaqDispose()`.

# imaqLabel

## Usage

```
int = imaqLabel(Image* dest, Image* source, int connectivity8, int*
                          particleCount)
```

## Purpose

Labels the particles in a binary image by applying a unique value to all pixels within a particle. This value is encoded in 8 or 16 bits, depending on the image type. The function can label 255 particles in an 8-bit image and 65,535 particles in a 16-bit image.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | Image* | The source image. The labeling process modifies the border of the source image. The border must be at least one pixel wide if you use connectivity-4 or two pixels wide if you use connectivity-8. |
| **connectivity8** | int | Set this parameter to TRUE to use connectivity-8 to determine whether particles are touching. Set this parameter to FALSE to use connectivity-4 to determine whether particles are touching. For more information about connectivity, see Chapter 1, *Basic Concepts*, or see the *IMAQ Vision User Manual*. |
| **particleCount** | int* | On return, the number of particles that the function found. Set this parameter to NULL if you do not need this information. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqMatchShape

## Usage

```
ShapeReport* = imaqMatchShape(Image* dest, const Image* source, const Image*
                              templateImage, int scaleInvariant, int
                              connectivity8, double tolerance, int* numMatches)
```

## Purpose

Finds a shape in an image. In most cases, use `imaqMatchPattern()` instead of this function. For information about when to use this function, see the *IMAQ Vision User Manual*.

## Image Type Supported

`IMAQ_IMAGE_U8`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | On return, the objects in the source image that match the object in the template image. |
| **source** | const Image* | The image in which the function searches for shapes. |
| **templateImage** | const Image* | The image containing the shape to find. |
| **scaleInvariant** | int | Set this parameter to TRUE to search for shapes regardless of size. Set this parameter to FALSE to search for shapes that are ± 10% the same size as the template shape. |
| **connectivity8** | int | Set this parameter to TRUE to use connectivity-8 to determine whether particles are touching. Set this parameter to FALSE to use connectivity-4 to determine whether particles are touching. For more information about connectivity, see Chapter 1, *Basic Concepts*, or see the *IMAQ Vision User Manual*. |
| **tolerance** | double | Indicates the allowable difference between the template shape and similar shapes in the image. The difference is expressed as a value from 0 to 1. |
| **numMatches** | int* | On return, the number of matches to the template image. Set this parameter to NULL if you do not need this information. |

## Return Value

**ShapeReport***—On success, this function returns an array of reports describing the matches to the given template shape. `ShapeReport` contains the following elements:

- `coordinates`—A `Rect` that contains the bounding rectangle of the object.
- `centroid`—A `Point` containing the coordinate location of the centroid of the object.
- `size`—The size (in pixels) of the object.
- `score`—A value ranging between 1 and 1,000 that specifies how similar the object in the image is to the template. A score of 1,000 indicates a perfect match.

On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with the array, dispose of it by calling `imaqDispose()`.

# imaqMorphology

## Usage

```
int = imaqMorphology(Image* dest, Image* source, MorphologyMethod method,
                     const StructuringElement* structuringElement)
```

## Purpose

Applies morphological transformations to binary images.

## Image Type Supported

IMAQ_IMAGE_U8

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | Image* | The image on which the function performs the morphological operations. The calculation modifies the border of the source image. The border must be at least half as large as the larger dimension of the structuring element. |
| **method** | MorphologyMethod | The morphological transform to apply. For a description of the morphological transformations, see the *IMAQ Vision User Manual*. |
| **structuringElement** | const StructuringElement* | The structuring element used in the operation. Set this parameter to NULL if you do not want a custom structuring element. For more information on structuring elements, see Chapter 1, *Basic Concepts*, or see the *IMAQ Vision User Manual*. |

## Parameter Discussion

**method**—The following operations are valid: IMAQ_AUTOM, IMAQ_CLOSE, IMAQ_DILATE, IMAQ_ERODE, IMAQ_GRADIENT, IMAQ_GRADIENTOUT, IMAQ_GRADIENTIN, IMAQ_HITMISS, IMAQ_OPEN, IMAQ_PCLOSE, IMAQ_POPEN, IMAQ_THICK, IMAQ_THIN.

**structuringElement**—A `StructuringElement` consists of the following elements:

- `matrixCols`—Number of columns in the matrix.
- `matrixRows`—Number of rows in the matrix.
- `hexa`—Set this field to TRUE if you specify a hexagonal structuring element in `kernel`. Set this field to FALSE if you specify a square structuring element in `kernel`.
- `kernel`—The values of the structuring element. Specify these values in order from the top-left of the kernel to the bottom-right of the kernel. For example, to set up the $3 \times 3$ kernel.

| | | |
|---|---|---|
| –1 | 0 | 1 |
| –1 | 0 | 1 |
| –1 | 0 | 1 |

called `myStructuringElement`, use the following syntax:

```
myStructuringElement.matrixCols = 3
myStructuringElement.matrixRows = 3
myStructuringElement.hexa = FALSE
myStructuringElement.kernel = malloc(9 * sizeof(int))
myStructuringElement.kernel[0] = –1
myStructuringElement.kernel[1] = 0
myStructuringElement.kernel[2] = 1
myStructuringElement.kernel[3] = –1
myStructuringElement.kernel[4] = 0
myStructuringElement.kernel[5] = 1
myStructuringElement.kernel[6] = –1
myStructuringElement.kernel[7] = 0
myStructuringElement.kernel[8] = 1
```

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqParticleFilter

## Usage

```
int = imaqParticleFilter(Image* dest, Image* source, const
                         ParticleFilterCriteria* criteria, int
                         criteriaCount, int keepMatches, int
                         connectivity8)
```

## Purpose

Filters particles based on their morphological measurements.

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | Image* | The image on which to perform the particle filter. The calculation modifies the border of the source image. The border must be at least one pixel wide if you use connectivity-4 or two pixels wide if you use connectivity-8. |
| **criteria** | const ParticleFilterCriteria* | The array of criteria for the filter. This parameter is required and cannot be NULL. |
| **criteriaCount** | int | Specifies the number of entries in the criteria array. |
| **keepMatches** | int | Set this parameter to TRUE to transfer only those particles that meet all the criteria to the destination. Set this parameter to FALSE to transfer only those particles that do not meet all the criteria. |
| **connectivity8** | int | Set this parameter to TRUE to use connectivity-8 to determine whether particles are touching. Set this parameter to FALSE to use connectivity-4 to determine whether particles are touching. For more information about connectivity, see Chapter 1, *Basic Concepts*, or see the *IMAQ Vision User Manual*. |

## Parameter Discussion

**criteria**—Each element in a `criteria` array consists of the following:

- `parameter`—The morphological measurement that the function uses for filtering. The following measurements are valid:
  - `IMAQ_AREA`—Surface area of the particle in pixels.
  - `IMAQ_AREA_CALIBRATED`—Surface area of the particle in calibrated units.
  - `IMAQ_NUM_HOLES`—Number of holes in the particle.
  - `IMAQ_AREA_OF_HOLES`—Surface area of the holes in calibrated units.
  - `IMAQ_TOTAL_AREA`—Total surface area (holes and particle) in calibrated units.
  - `IMAQ_IMAGE_AREA`—Surface area of the entire image in calibrated units.
  - `IMAQ_TOTAL_TO_IMAGE`—Ratio, expressed as a percent, of the total surface area (holes and particle) in relation to the image area.
  - `IMAQ_PARTICLE_TO_IMAGE`—Ratio, expressed as a percent, of the surface area of a particle in relation to the image area.
  - `IMAQ_CENTER_MASS_X`—X-coordinate of the center of gravity.
  - `IMAQ_CENTER_MASS_Y`—Y-coordinate of the center of gravity.
  - `IMAQ_LEFT_COLUMN`—Left edge of the bounding rectangle.
  - `IMAQ_TOP_ROW`—Top edge of the bounding rectangle.
  - `IMAQ_RIGHT_COLUMN`—Right edge of the bounding rectangle.
  - `IMAQ_BOTTOM_ROW`—Bottom edge of bounding rectangle.
  - `IMAQ_WIDTH`—Width of bounding rectangle in calibrated units.
  - `IMAQ_HEIGHT`—Height of bounding rectangle in calibrated units.
  - `IMAQ_MAX_SEGMENT_LENGTH`—Length of longest horizontal line segment.
  - `IMAQ_MAX_SEGMENT_LEFT_COLUMN`—Leftmost x-coordinate of longest horizontal line segment.
  - `IMAQ_MAX_SEGMENT_TOP_ROW`—Y-coordinate of longest horizontal line segment.
  - `IMAQ_PERIMETER`—Outer perimeter of the particle.
  - `IMAQ_PERIMETER_OF_HOLES`—Perimeter of all holes within the particle.
  - `IMAQ_SIGMA_X`—Sum of the particle pixels on the x-axis.
  - `IMAQ_SIGMA_Y`—Sum of the particle pixels on the y-axis.
  - `IMAQ_SIGMA_XX`—Sum of the particle pixels on the x-axis squared.
  - `IMAQ_SIGMA_YY`—Sum of the particle pixels on the y-axis squared.
  - `IMAQ_SIGMA_XY`—Sum of the particle pixels on the x-axis and y-axis.
  - `IMAQ_PROJ_X`—Projection corrected in X.

  - IMAQ_PROJ_Y—Projection corrected in Y.
  - IMAQ_INERTIA_XX—Inertia matrix coefficient in XX.
  - IMAQ_INERTIA_YY—Inertia matrix coefficient in YY.
  - IMAQ_INERTIA_XY—Inertia matrix coefficient in XY.
  - IMAQ_MEAN_H—Mean length of horizontal segments.
  - IMAQ_MEAN_V—Mean length of vertical segments.
  - IMAQ_MAX_INTERCEPT—Length of longest segment of the convex hull.
  - IMAQ_MEAN_INTERCEPT—Mean length of the chords in an object perpendicular to its max intercept.
  - IMAQ_ORIENTATION—Direction of the longest segment.
  - IMAQ_EQUIV_ELLIPSE_MINOR—Total length of the axis of the ellipse having the same area as the particle and a major axis equal to half the max intercept.
  - IMAQ_ELLIPSE_MAJOR—Total length of major axis having the same area and perimeter as the particle in calibrated units.
  - IMAQ_ELLIPSE_MINOR—Total length of minor axis having the same area and perimeter as the particle in calibrated units.
  - IMAQ_ELLIPSE_RATIO—Fraction of major axis to minor axis.
  - IMAQ_RECT_LONG_SIDE—Length of the long side of a rectangle having the same area and perimeter as the particle in calibrated units.
  - IMAQ_RECT_SHORT_SIDE—Length of the short side of a rectangle having the same area and perimeter as the particle in calibrated units.
  - IMAQ_RECT_RATIO—Ratio of rectangle long side to rectangle short side.
  - IMAQ_ELONGATION—Max intercept / mean perpendicular intercept.
  - IMAQ_COMPACTNESS—Particle area / (height × width).
  - IMAQ_HEYWOOD—Particle perimeter / perimeter of the circle having the same area as the particle.
  - IMAQ_TYPE_FACTOR—A complex factor relating the surface area to the moment of inertia.
  - IMAQ_HYDRAULIC—Particle area / particle perimeter.
  - IMAQ_WADDLE_DISK—Diameter of the disk having the same area as the particle in user units.
  - IMAQ_DIAGONAL—Diagonal of an equivalent rectangle in user units.
- lower—The lower bound of the criteria range.
- upper—The upper bound of the criteria range.

- `exclude`—Set this element to TRUE to indicate that a match occurs when the value is outside the criteria range. Set this element to FALSE to indicate that a match occurs when the value is inside the criteria range.

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqRejectBorder

## Usage

```
int = imaqRejectBorder(Image* dest, Image* source, int connectivity8)
```

## Purpose

Eliminates particles touching the border of the image.

## Image Type Supported

`IMAQ_IMAGE_U8`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | Image* | The source image. If the image has a border, the function sets all border pixel values to 0. |
| **connectivity8** | int | Set this parameter to TRUE to use connectivity-8 to determine whether particles are touching. Set this parameter to FALSE to use connectivity-4 to determine whether particles are touching. For more information about connectivity, see Chapter 1, *Basic Concepts*, or see the *IMAQ Vision User Manual*. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSegmentation

## Usage

```
int = imaqSegmentation(Image* dest, Image* source)
```

## Purpose

Calculates zones of influence around particles in a labeled image. The function finds the nearest particle of each pixel in the source image and sets the pixel value to the labeled value of that particle. Before calling `imaqSegmentation()`, you must label the particles with `imaqLabel()`.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | Image* | The image to segment. The segmentation modifies the border of the source image. The border must be at least one pixel wide. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSelectParticles

## Usage

```
ParticleReport* = imaqSelectParticles(const Image* image, const
                          ParticleReport* reports, int* reportCount, int
                          rejectBorder, const SelectParticleCriteria*
                          criteria, int* criteriaCount)
```

## Purpose

Filters an array of particle reports based on criteria ranges. Call `imaqGetParticleInfo()` before calling `imaqSelectParticle()`.

## Image Type Supported

`IMAQ_IMAGE_U8`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image from which `imaqGetParticleInfo()` generated the particle reports. |
| **reports** | const ParticleReport* | The array of reports that `imaqGetParticleInfo()` returned. The function makes a copy of the reports that match the criteria. |
| **reportCount** | int | The number of reports in the input **reports** array. |
| **rejectBorder** | int | Set this parameter to TRUE to filter out particles touching the border of the image. Set this parameter to FALSE to process all particles. |
| **criteria** | const SelectParticleCriteria* | The array of criteria for the filter. This parameter is required and cannot be NULL. |
| **criteriaCount** | int | The number of criteria in the criteria array. |
| **selectedCount** | int* | On return, the number of reports that matched the criteria. Set this parameter to NULL if you do not need this information. |

## Parameter Discussion

**criteria**—Each `SelectParticleCriteria` structure consists of the following elements:

- `parameter`—The morphological measurement that the function uses for filtering. The following measurements are valid. If you set the mode of `imaqGetParticleInfo()` to `IMAQ_BASIC_INFO`, the `criteria` array must contain only the values denoted as *basic*.
  - `IMAQ_AREA`—Surface area of the particle in pixels (basic).
  - `IMAQ_AREA_CALIBRATED`—Surface area of the particle in calibrated units (basic).
  - `IMAQ_NUM_HOLES`—Number of holes in the particle.
  - `IMAQ_AREA_OF_HOLES`—Surface area of the holes in calibrated units.
  - `IMAQ_LEFT_COLUMN`—Left edge of the bounding rectangle (basic).
  - `IMAQ_TOP_ROW`—Top edge of the bounding rectangle (basic).
  - `IMAQ_RIGHT_COLUMN`—Right edge of the bounding rectangle (basic).
  - `IMAQ_BOTTOM_ROW`—Bottom edge of the bounding rectangle (basic).
  - `IMAQ_MAX_SEGMENT_LENGTH`—Length of longest horizontal line segment.
  - `IMAQ_MAX_SEGMENT_LEFT_COLUMN`—Leftmost x-coordinate of longest horizontal line segment.
  - `IMAQ_MAX_SEGMENT_TOP_ROW`—Y-coordinate of longest horizontal line segment.
  - `IMAQ_PERIMETER`—Outer perimeter of the particle.
  - `IMAQ_PERIMETER_OF_HOLES`—Perimeter of all holes.
  - `IMAQ_SIGMA_X`—Sum of the particle pixels on the x-axis.
  - `IMAQ_SIGMA_Y`—Sum of the particle pixels on the y-axis.
  - `IMAQ_SIGMA_XX`—Sum of the particle pixels on the x-axis squared.
  - `IMAQ_SIGMA_YY`—Sum of the particle pixels on the y-axis squared.
  - `IMAQ_SIGMA_XY`—Sum of the particle pixels on the x-axis and y-axis.
  - `IMAQ_PROJ_X`—Projection corrected in x.
  - `IMAQ_PROJ_Y`—Projection corrected in y.
- `lower`—The lower boundary of the criteria range
- `upper`—The upper boundary of the criteria range.

## Return Value

**ParticleReport***—On success, this function returns an array of the input reports that match the filtration criteria. On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with this information, dispose of it by calling `imaqDispose()`. For more information on the ParticleReport structure, see `imaqGetParticleInfo()`.

# imaqSeparation

## Usage

```
int = imaqSeparation(Image* dest, Image* source, int erosions, const
                     StructuringElement* structuringElement)
```

## Purpose

Separates touching particles by eroding the image to remove small isthmuses between the particles. After performing the erosion, the algorithm reconstructs the image.

## Image Type Supported

`IMAQ_IMAGE_U8`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | Image* | The image containing particles to separate. The separation modifies the border of the source image. The border must be at least half as large as the larger dimension of the structuring element. |
| **erosions** | int | The number of erosions to perform. |
| **structuringElement** | const StructuringElement* | The structuring element used in the operation. Set this parameter to NULL if you do not want a custom structuring element. For more information on structuring elements, see Chapter 1, *Basic Concepts*, or see the *IMAQ Vision User Manual*. |

## Parameter Discussion

**structuringElement**—A `StructuringElement` consists of the following elements:

- `matrixCols`—Number of columns in the matrix.
- `matrixRows`—Number of rows in the matrix.
- `hexa`—Set this field to TRUE if you specify a hexagonal structuring element in `kernel`. Set this field to FALSE if you specify a square structuring element in `kernel`.

- `kernel`—The values of the structuring element. Specify these values in order from the top-left of the kernel to the bottom-right of the kernel. For example, to set up the $3 \times 3$ kernel.

| | | |
|---|---|---|
| –1 | 0 | 1 |
| –1 | 0 | 1 |
| –1 | 0 | 1 |

called `myStructuringElement`, use the following syntax:

```
myStructuringElement.matrixCols = 3
myStructuringElement.matrixRows = 3
myStructuringElement.hexa = FALSE
myStructuringElement.kernel = malloc(9 * sizeof(int))
myStructuringElement.kernel[0] = –1
myStructuringElement.kernel[1] = 0
myStructuringElement.kernel[2] = 1
myStructuringElement.kernel[3] = –1
myStructuringElement.kernel[4] = 0
myStructuringElement.kernel[5] = 1
myStructuringElement.kernel[6] = –1
myStructuringElement.kernel[7] = 0
myStructuringElement.kernel[8] = 1
```

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSimpleDistance

## Usage

```
int = imaqSimpleDistance(Image* dest, Image* source, const
                         StructuringElement* structuringElement)
```

## Purpose

Creates a distance map. The function encodes the pixel value of a particle as a function of the distance of the pixel from the particle border. For a more precise but slower algorithm, use `imaqDanielssonDistance()`.

## Image Type Supported

`IMAQ_IMAGE_U8`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | Image* | The image that the function uses to compute the distance map. The function modifies the border of the source image. The border must be at least half as large as the larger of the structuring element dimensions. |
| **structuringElement** | const StructuringElement* | The structuring element used in the operation. Set this parameter to NULL if you do not want a custom structuring element. For more information on structuring elements, see Chapter 1, *Basic Concepts*, or see the *IMAQ Vision User Manual*. |

## Parameter Discussion

**structuringElement**—A `StructuringElement` consists of the following elements:

- `matrixCols`—Number of columns in the matrix.
- `matrixRows`—Number of rows in the matrix.
- `hexa`—Set this field to TRUE if you specify a hexagonal structuring element in `kernel`. Set this field to FALSE if you specify a square structuring element in `kernel`.

- `kernel`—The values of the structuring element. Specify these values in order from the top-left of the kernel to the bottom-right of the kernel. For example, to set up the $3 \times 3$ kernel.

| | | |
|---|---|---|
| –1 | 0 | 1 |
| –1 | 0 | 1 |
| –1 | 0 | 1 |

called `myStructuringElement`, use the following syntax:

```
myStructuringElement.matrixCols = 3
myStructuringElement.matrixRows = 3
myStructuringElement.hexa = FALSE
myStructuringElement.kernel = malloc(9 * sizeof(int))
myStructuringElement.kernel[0] = –1
myStructuringElement.kernel[1] = 0
myStructuringElement.kernel[2] = 1
myStructuringElement.kernel[3] = –1
myStructuringElement.kernel[4] = 0
myStructuringElement.kernel[5] = 1
myStructuringElement.kernel[6] = –1
myStructuringElement.kernel[7] = 0
myStructuringElement.kernel[8] = 1
```

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSizeFilter

## Usage

```
int = imaqSizeFilter(Image* dest, Image* source, int connectivity8, int
                     erosions, SizeType keepSize, const
                     StructuringElement* structuringElement)
```

## Purpose

Filters particles based on their size. The algorithm erodes the image a specified number of times and either keeps or discards the particles from the original image that remain in the eroded image.

## Image Type Supported

`IMAQ_IMAGE_U8`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | Image* | The image on which the function applies the filter. The calculation modifies the border of the source image. The border must be at least half as large as the larger of the structuring element dimensions. |
| **connectivity8** | int | Set this parameter to TRUE to use connectivity-8 to determine whether particles are touching. Set this parameter to FALSE to use connectivity-4 to determine whether particles are touching. For more information about connectivity, see Chapter 1, *Basic Concepts*, or see the *IMAQ Vision User Manual*. |
| **erosions** | int | The number of erosions to perform. |
| **keepSize** | SizeType | Set this parameter to `IMAQ_KEEP_LARGE` to keep the particles remaining after the erosion. Set this parameter to `IMAQ_KEEP_SMALL` to keep the particles eliminated by the erosion. |
| **structuringElement** | const StructuringElement* | The structuring element used in the operation. Set this parameter to NULL if you do not want a custom structuring element. For more information on structuring elements, see Chapter 1, *Basic Concepts*, or see the *IMAQ Vision User Manual*. |

## Parameter Discussion

**structuringElement**—A `StructuringElement` consists of the following elements:

- `matrixCols`—Number of columns in the matrix.
- `matrixRows`—Number of rows in the matrix.
- `hexa`—Set this field to TRUE if you specify a hexagonal structuring element in `kernel`. Set this field to FALSE if you specify a square structuring element in `kernel`.
- `kernel`—The values of the structuring element. Specify these values in order from the top-left of the kernel to the bottom-right of the kernel. For example, to set up the $3 \times 3$ kernel.

| | | |
|---|---|---|
| –1 | 0 | 1 |
| –1 | 0 | 1 |
| –1 | 0 | 1 |

called `myStructuringElement`, use the following syntax:

```
myStructuringElement.matrixCols = 3
myStructuringElement.matrixRows = 3
myStructuringElement.hexa = FALSE
myStructuringElement.kernel = malloc(9 * sizeof(int))
myStructuringElement.kernel[0] = –1
myStructuringElement.kernel[1] = 0
myStructuringElement.kernel[2] = 1
myStructuringElement.kernel[3] = –1
myStructuringElement.kernel[4] = 0
myStructuringElement.kernel[5] = 1
myStructuringElement.kernel[6] = –1
myStructuringElement.kernel[7] = 0
myStructuringElement.kernel[8] = 1
```

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSkeleton

## Usage

```
int = imaqSkeleton(Image* dest, Image* source, SkeletonMethod method)
```

## Purpose

Calculates the skeleton of the particles inside the image. The skeleton is made up of lines separating the zones of influence in the image.

## Image Type Supported

IMAQ_IMAGE_U8

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | Image* | The image whose skeleton the function derives. The calculation modifies the border of the source image. The border must be at least one pixel wide. |
| **method** | SkeletonMethod | The method that the function uses to calculate the skeleton. The following options are valid:<br>IMAQ_SKELETON_L<br>IMAQ_SKELETON_M<br>IMAQ_SKELETON_INVERSE<br>For more information about skeleton functions, see the *IMAQ Vision User Manual*. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# 12

# Color Processing

This chapter describes the Color Processing functions in IMAQ Vision for LabWindows/CVI. Color Processing functions allow you to analyze and process color images in different color spaces. Use Color Processing functions with applications in which color information is important. These functions work with color images in the Red, Green, Blue (RGB) domain and the Hue, Saturation, and Luminance (HSL) domain. For more information about color domains, see the *IMAQ Vision User Manual*.

## Color Processing Function Panels

Table 12-1 lists the Color Processing functions in a tree structure. The functions in the Color Processing class are grouped according to the types of operations they perform. The first-level heading in the tree is the name of the class. The second-level headings are names of function subclasses. The third-level headings are names of individual function panels. Each Color Processing function panel represents one function.

**Table 12-1.** Color Processing Function Tree

| Class/Panel Name | Function Name |
|---|---|
| Color Processing | |
| Change Color Space | `imaqChangeColorSpace` |
| Color BCG Transform | `imaqColorBCGTransform` |
| Color Equalize | `imaqColorEqualize` |
| Color Histogram | `imaqColorHistogram` |
| Color Lookup | `imaqColorLookup` |
| Color Threshold | `imaqColorThreshold` |
| Color Matching | |
| Learn Color | `imaqLearnColor` |
| Match Color | `imaqMatchColor` |

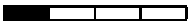
## Subclass Description

The Color Processing subclass description is as follows:

- Color Matching functions allow you to learn information about the colors in a template image and compare that information with the colors in other images.

# Color Spaces

An RGB or HSL image is a color image coded in three parts. For RGB, these parts are red, green, and blue. For HSL, these parts are hue, saturation, and luminance. A pixel encoded in 32 bits is actually four planes:

alpha plane (not used) 

red or hue plane 

green or saturation plane 

blue or luminance plane 

A color image is encoded in memory in either the form RGB or HSL. However, there are a number of other color spaces, such as HSI and HSV. The HSI model is composed as hue, saturation, and intensity. The HSV model is composed as hue, saturation, and value.

To determine the values for hue, saturation, luminance, intensity, or value, a measurement is made from the red, green, and blue components. These extractions are approximate. In effect, a color converted between two of the different color spaces (for instance, RGB to HSL) and then reconverted back to the original color space, does not have exactly the same values as the original image. This difference results from the 8-bit encoding of the image planes, which causes some loss of data.

The following formulas illustrate converting an RGB color value to an HSL color value:

$$H = \frac{256\tan^{-1}\left(\frac{\sqrt{3}(G-B)}{2R-G-B}\right)}{2\pi}$$

$$S = 256\left(1-\frac{3\min(R, G, B)}{R+G+B}\right)$$

$$L = 0.299R + 0.587G + 0.114B$$

In the HSV and HSI modes, H and S are computed as approximations to the model shown above. V and I are computed as follows:

$$V = \frac{\max(R, G, B) - \min(R, G, B)}{2}$$

$$I = \max(R, G, B)$$

# imaqChangeColorSpace

## Usage

```
Color = imaqChangeColorSpace(const Color* sourceColor, ColorMode
                             sourceSpace, ColorMode destSpace)
```

## Purpose

Maps the value of a color in one color space into the value of the same color in another color space.

## Parameters

| Name | Type | Description |
|---|---|---|
| **sourceColor** | const Color* | The color in the source color space. This parameter is required and cannot be NULL. |
| **sourceSpace** | ColorMode | The source color space. Valid options are `IMAQ_RGB`, `IMAQ_HSL`, `IMAQ_HSV`, and `IMAQ_HSI`. For more information about color spaces, see the *IMAQ Vision User Manual*. |
| **destSpace** | ColorMode | The destination color space. Valid options are `IMAQ_RGB`, `IMAQ_HSL`, `IMAQ_HSV`, and `IMAQ_HSI`. For more information about color spaces, see the *IMAQ Vision User Manual*. |

## Return Value

**Color**—On success, this function returns the value of the color in the destination color space. On failure, this function returns black. To get extended error information, call `imaqGetLastError()`.

# imaqColorBCGTransform

## Usage

```
int = imaqColorBCGTransform(Image* dest, const Image* source, const
                            BCGOptions* redOptions, const BCGOptions*
                            greenOptions, const BCGOptions* blueOptions,
                            const Image* mask)
```

## Purpose

Applies brightness, contrast, and gamma correction to each plane of a color image.

## Image Type Supported

`IMAQ_IMAGE_RGB`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The image to transform. |
| **redOptions** | const BCGOptions* | The parameters to use in the transform of the red plane. Set this parameter to NULL to leave the red plane unchanged. |
| **greenOptions** | const BCGOptions* | The parameters to use in the transform of the green plane. Set this parameter to NULL to leave the green plane unchanged. |
| **blueOptions** | const BCGOptions* | The parameters to use in the transform of the blue plane. Set this parameter to NULL to leave the blue plane unchanged. |
| **mask** | const Image* | An optional mask image. This image must be an `IMAQ_IMAGE_U8` image. The function transforms only those pixel in the source image whose corresponding pixels in the mask image are non-zero. Set this parameter to NULL to transform the whole image. |

## Parameter Discussion

**redOptions**, **greenOptions**, **blueOptions**—A `BCGOptions` structure has the following elements:

- `brightness`—Adjusts the brightness of the image. A value of 128 leaves the brightness unchanged. Values below 128 darken the image, and values above 128 brighten the image.

- `contrast`—Adjusts the contrast of the image. A value of 45 leaves the contrast unchanged. Values below 45 decrease the contrast, and values above 45 increase the contrast.
- `gamma`—Performs gamma correction. A value of 1.0 leaves the image unchanged. Values below 1.0 enhance contrast for darker pixel at the expense of the brighter pixels. Values above 1.0 enhance contrast for brighter pixels at the expense of darker pixels.

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqColorEqualize

## Usage

```
int = imaqColorEqualize(Image* dest, const Image* source, int
                        colorEqualization)
```

## Purpose

Calculates the histogram of each plane of a color image and redistributes pixel values across the desired range while maintaining pixel value groupings.

## Image Types Supported

`IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The image to equalize. |
| **colorEqualization** | int | Set this parameter to TRUE to equalize all three planes of the image. Set this parameter to FALSE to equalize only the luminance plane. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqColorHistogram

## Usage

```
ColorHistogramReport* = imaqColorHistogram(Image* image, int numClasses,
                        ColorMode mode, const Image* mask)
```

## Purpose

Calculates the histogram, or pixel distribution, of a color image.

## Image Types Supported

IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | Image* | The image whose histogram the function calculates. |
| **numClasses** | int | The number of classes into which the function separates the pixels. |
| **mode** | ColorMode | The color space in which to perform the histogram. Valid options are IMAQ_RGB, IMAQ_HSL, IMAQ_HSV, and IMAQ_HSI. For more information about color spaces, see the *IMAQ Vision User Manual*. |
| **mask** | const Image* | An optional mask image. This image must be an IMAQ_IMAGE_U8 image. The function calculates the histogram using only those pixels in the image whose corresponding pixels in the mask are non-zero. Set this parameter to NULL to calculate the histogram of the entire image. |

## Return Value

**ColorHistogramReport***—On success, this function returns a report describing the classification of each plane in a HistogramReport. The ColorHistogramReport structure contains the following elements:

- plane1—The histogram report of the first color plane.
- plane2—The histogram report of the second plane.
- plane3—The histogram report of the third plane.

The data of each plane depends on the **mode** parameter, as follows:

- IMAQ_RGB—plane1 is the red plane, plane2 is the green plane, plane3 is the blue plane.
- IMAQ_HSL—plane1 is the hue plane, plane2 is the saturation plane, plane3 is the luminance plane.
- IMAQ_HSV—plane1 is the hue plane, plane2 is the saturation plane, plane3 is the value plane.
- IMAQ_HSI—plane1 is the hue plane, plane2 is the saturation plane, plane3 is the intensity plane.

Each HistogramReport structure contains the following elements:

- histogram—An array describing the number of pixels that fell into each class.
- histogramCount—The number of elements in the histogram array. The number of elements equals the value you provided in **numClasses**.
- min—The smallest pixel value that the function classified.
- max—The largest pixel value that the function classified.
- start—The smallest pixel value that fell into the first class. The smallest pixel value may be smaller than **min**.
- width—The size of each class.
- mean—The mean value of the pixels that the function classified.
- stdDev—The standard deviation of the pixels that the function classified.
- numPixels—The number of pixels that the function classified. The **mask** and the given **min** and **max** influence this element.

On failure, this function returns NULL. To get extended error information, call imaqGetLastError(). When you are finished with the report, dispose of it by calling imaqDispose().

# imaqColorLookup

## Usage

```
int = imaqColorLookup(Image* dest, const Image* source, ColorMode mode, const
                      Image* mask, const short* plane1, const short*
                      plane2, const short* plane3)
```

## Purpose

Performs a transformation on an image by replacing each pixel value in a given color plane with the lookup table entry corresponding to that value.

## Image Types Supported

IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The image on which to apply the lookup. |
| **mode** | ColorMode | The color space in which to apply the lookup. Valid options are IMAQ_RGB, IMAQ_HSL, IMAQ_HSV, and IMAQ_HSI. If the image is not in the color space you specify, the function converts the pixels into the specified color space, performs the lookup, and converts the pixels back to their original color space. For more information on color spaces, see the *IMAQ Vision User Manual*. |
| **mask** | const Image* | An optional mask image. This image must be an IMAQ_IMAGE_U8 image. The function applies the lookup to only those pixels in the source image whose corresponding pixels in the mask image are non-zero. Set this parameter to NULL to apply the lookup to the whole image. |
| **plane1** | const short* | The lookup table for the first plane of the image. If you set this parameter, the table must contain 256 values. Set this parameter to NULL to leave the first plane unchanged. The color plane depends on mode, as follows:<br><br>IMAQ_RGB—red plane<br>IMAQ_HSL—hue plane<br>IMAQ_HSV—hue plane<br>IMAQ_HSI—hue plane |

| Name | Type | Description |
|---|---|---|
| **plane2** | const short* | The lookup table for the second plane of the image. If you set this parameter, the table must contain 256 values. Set this parameter to NULL to leave the second plane unchanged. The color plane depends on mode, as follows:<br><br>`IMAQ_RGB`—green plane<br>`IMAQ_HSL`—saturation plane<br>`IMAQ_HSV`—saturation plane<br>`IMAQ_HSI`—saturation plane |
| **plane3** | const short* | The lookup table for the third plane of the image. If you set this parameter, the table must contain 256 values. Set this parameter to NULL to leave the third plane unchanged. The color plane depends on mode, as follows:<br><br>`IMAQ_RGB`—blue plane<br>`IMAQ_HSL`—luminance plane<br>`IMAQ_HSV`—value plane<br>`IMAQ_HSI`—intensity plane |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqColorThreshold

## Usage

```
int = imaqColorThreshold(Image* dest, const Image* source, int replaceValue,
                         ColorMode mode, const Range* plane1Range, const
                         Range* plane2Range, const Range* plane3Range)
```

## Purpose

Thresholds a color image. The function selects a pixel if all three color components fall within the specified range. The function replaces the value of selected pixels with the given replacement value and sets the value of unselected pixels to 0.

## Image Types Supported

`IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. This image must be an IMAQ_IMAGE_U8 image. |
| **source** | const Image* | The image to threshold. |
| **replaceValue** | int | The value to which the function sets selected pixels. |
| **mode** | ColorMode | The color space in which to perform the threshold. Valid options are `IMAQ_RGB`, `IMAQ_HSL`, `IMAQ_HSV`, and `IMAQ_HSI`. For more information about color spaces, see the *IMAQ Vision User Manual*. |
| **plane1Range** | const Range* | The selection range for the first plane of the image. Set this parameter to NULL to use a selection range from 0 to 255. |
| **plane2Range** | const Range* | The selection range for the second plane of the image. Set this parameter to NULL to use a selection range from 0 to 255. |
| **plane3Range** | const Range* | The selection range for the third plane of the image. Set this parameter to NULL to use a selection range from 0 to 255. |

## Parameter Discussion

**plane1Range—**The color plane depends on the **mode**, as follows:

- `IMAQ_RGB`—red plane
- `IMAQ_HSL`—hue plane

- IMAQ_HSV—hue plane
- IMAQ_HSI—hue plane

**plane2Range**—The color plane depends on the **mode**, as follows:

- IMAQ_RGB—green plane
- IMAQ_HSL—saturation plane
- IMAQ_HSV—saturation plane
- IMAQ_HSI—saturation plane

**plane3Range—**The color plane depends on the **mode**, as follows:

- IMAQ_RGB—blue plane
- IMAQ_HSL—luminance plane
- IMAQ_HSV—value plane
- IMAQ_HSI—intensity plane

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqLearnColor

## Usage

```
ColorInformation* = imaqLearnColor(const Image* image, const ROI* roi,
                                   ColorComplexity complexity, int saturation)
```

## Purpose

Extracts the color features of an image. Use these features for color matching with `imaqMatchColor()`.

## Image Types Supported

`IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image containing the color information to learn. |
| **roi** | const ROI* | The region about which the function learns the color information. Set this parameter to NULL to learn color information about the whole image. |
| **complexity** | ColorComplexity | Specifies the complexity the color information in the image. The following options are valid:<br>`IMAQ_COMPLEXITY_LOW`<br>`IMAQ_COMPLEXITY_MED`<br>`IMAQ_COMPLEXITY_HIGH`<br>In most cases, set this parameter to `IMAQ_COMPLEXITY_LOW`. However, set this parameter to `IMAQ_COMPLEXITY_HIGH` to learn more information and better distinguish colors in highly complex images. |
| **saturation** | int | Sets a threshold value which the function uses to separate colors with similar hues. The function classifies colors below the given saturation value separately from colors above the given saturation value. |

## Return Value

**ColorInformation***—On success, this function returns a color information structure which you can pass into `imaqMatchColor()`. On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with this structure, dispose of it by calling `imaqDispose()`.

# imaqMatchColor

## Usage

```
int* = imaqMatchColor(const Image* image, const ColorInformation* info, const
                      ROI* roi, int* numScores)
```

## Purpose

Determines how closely colors in an image match colors in the given color information.

## Image Types Supported

`IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image containing colors you want to compare with the given color information. |
| **info** | const ColorInformation* | The color information. Call `imaqLearnColor()` to get the color information. This parameter is required and cannot be NULL. |
| **roi** | const ROI* | The region of the image in which to compare the colors. Set this parameter to NULL to compare colors in the entire image. |
| **numScores** | int* | On return, contains the number of values in the score array. Set this parameter to NULL if you do not need this information. |

## Return Value

**int***—On success, this function returns an array filled with scores describing the closeness of a match between each contour in the ROI and the color information. A score of 1,000 indicates a perfect match. On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with this array, dispose of it by calling `imaqDispose()`.

# 13

# Pattern Matching

This chapter describes the Pattern Matching functions in IMAQ Vision for LabWindows/CVI. Pattern Matching functions allow you to search for templates in an image.

## Pattern Matching Function Panels

Table 13-1 lists the Pattern Matching functions in a tree structure. The functions in the Pattern Matching class are grouped according to the types of operations they perform. The first-level heading in the tree is the name of the class. The second-level headings are names of individual function panels. Each Pattern Matching function panel represents one function.

**Table 13-1.** Pattern Matching Function Tree

| Class/Panel Name | Function Name |
|---|---|
| Pattern Matching | |
| Learn Pattern | `imaqLearnPattern` |
| Load Pattern | `imaqLoadPattern` |
| Match Pattern | `imaqMatchPattern` |
| Save Pattern | `imaqSavePattern` |

# imaqLearnPattern

## Usage

```
int = imaqLearnPattern(Image* image, LearningMode learningMode)
```

## Purpose

Prepares an image for use as a pattern for `imaqMatchPattern()`. If you change the pattern image after calling this function, you must call the function again to learn the modified image.

## Image Type Supported

`IMAQ_IMAGE_U8`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | Image* | The image about which the function learns pattern matching information. The function appends the pattern matching information to the image. |
| **learningMode** | LearningMode | The mode in which the function learns the pattern image. The following options are valid:<br>`IMAQ_LEARN_ALL`—Learns information about the image that `imaqMatchPattern()` requires for shift-invariant and rotation-invariant pattern matching.<br>`IMAQ_LEARN_SHIFTINVARIANT`—Learns information about the image that `imaqMatchPattern()` requires for shift-invariant pattern matching.<br>`IMAQ_LEARN_ROTATIONINVARIANT`—Learns information about the image that `imaqMatchPattern()` requires for rotation-invariant pattern matching. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqLoadPattern

## Usage

```
int = imaqLoadPattern(Image* pattern, const char* fileName)
```

## Purpose

Loads a pattern image previously saved with the `imaqSavePattern()` function.

## Image Type Supported

`IMAQ_IMAGE_U8`

## Parameters

| Name | Type | Description |
|---|---|---|
| **pattern** | Image* | The pattern image. |
| **fileName** | const char* | The name of the image file to load. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqMatchPattern

## Usage

```
PatternMatch* = imaqMatchPattern(const Image* image, Image* pattern, const
                          MatchPatternOptions* options, Rect searchRect,
                          int* numMatches)
```

## Purpose

Searches for areas in an image that match a given pattern image.

## Image Type Supported

IMAQ_IMAGE_U8

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image in which the function finds matches to the pattern image. |
| **pattern** | Image* | The pattern image to find in the image. IMAQ Vision must learn this pattern image in `imaqLearnPattern()` before using it in this function. |
| **options** | const MatchPatternOptions* | Describes how to search for the pattern image. |
| **searchRect** | Rect | Specifies a rectangle in the image in which to search for the pattern image. Set this parameter to IMAQ_NO_RECT to search for the pattern image in the entire image. |
| **numMatches** | int* | On return, the number of matches to the pattern image that the function found. Set this parameter to NULL if you do not need this information. |

## Parameter Discussion

**options**—The following options are valid:

- `mode`—Specifies the method to use when looking for the pattern in the image. The following modes are valid:
  - `IMAQ_MATCH_SHIFT_INVARIANT`—Searches for occurrences of the pattern image anywhere in the searchRect, assuming that the pattern is not rotated more than ± 4°.

  - IMAQ_MATCH_ROTATION_INVARIANT—Searches for occurrences of the pattern in the image with no restriction on the rotation of the pattern.
- minContrast—Specifies the minimum contrast expected in the image.
- subpixelAccuracy—Set this parameter to TRUE to return areas in the image that match the pattern area with subpixel accuracy.
- angleRanges—An array of angle ranges, in degrees, where each range specifies how much you expect the pattern to be rotated in the image. To decrease the search time, limit the degrees of rotation in which you expect to find the pattern image.
- numRanges—Number of angle ranges in the angleRanges array.
- numMatchesRequested—Number of valid matches expected.
- matchFactor—Controls the number of potential matches that the function examines. Acceptable values range from 0 to 1,000. For most applications, set matchFactor to 0, which optimizes the speed of the algorithm. If you are not getting all of the numMatchesRequested, increasing this factor may increase the number of matches you receive but decreases the speed of the algorithm. Normally, increasing matchFactor is necessary only when looking for more than 200 matches per image.
- minMatchScore—The minimum score a match can have for the function to consider the match valid.

Set the **options** parameter to NULL to use the default options, as follows:

- mode—IMAQ_MATCH_SHIFT_INVARIANT
- minContrast—10
- subpixelAccuracy—FALSE
- angleRanges—NULL (all angles allowed)
- numRanges—0
- numMatchesRequested—1
- matchFactor—0
- minMatchScore—800

## Return Value

**PatternMatch***—On success, this function returns an array of information about each match found. Each PatternMatch structure contains the following information:

- position—The location of the center of the match.
- rotation—The rotation of the match relative to the pattern image, in degrees.
- scale—The ratio of the size of the match relative to the pattern image.

- `score`—The accuracy of the match. A 1,000 indicates a perfect match, and a 0 indicates no match.
- `corner`—An array of four points describing the rectangle surrounding the pattern image.

On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with this information, dispose of the pointer by calling `imaqDispose()`.

# imaqSavePattern

## Usage

```
int = imaqSavePattern(const Image* pattern, const char* filename)
```

## Purpose

Saves a pattern image to disk in PNG format. If you alter the contents of this file, IMAQ Vision may not be able to use the file as a pattern image.

## Image Type Supported

`IMAQ_IMAGE_U8`

## Parameters

| Name | Type | Description |
|---|---|---|
| **pattern** | const Image* | The pattern image to save. |
| **filename** | const char* | The name in which the function saves the pattern image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# 14

# Caliper

This chapter describes the Caliper functions in IMAQ Vision for LabWindows/CVI. Caliper functions allow you to detect and measure features, such as edges and angles, along a path in an image.

## Caliper Function Panels

Table 14-1 lists the Caliper functions in a tree structure. The functions in the Caliper class are grouped according to the types of operations they perform. The first-level heading in the tree is the name of the class. The second-level headings are names of individual function panels. Each Caliper function panel represents one function.

**Table 14-1.** Caliper Function Tree

| Class/Panel Name | Function Name |
|---|---|
| Caliper | |
| Caliper Tool | `imaqCaliperTool` |
| Detect Rotation | `imaqDetectRotation` |
| Edge Tool | `imaqEdgeTool` |
| Line Gauge Tool | `imaqLineGaugeTool` |
| Simple Edge | `imaqSimpleEdge` |

# imaqCaliperTool

## Usage

```
CaliperReport* = imaqCaliperTool(const Image* image, const Point* points, int
                          numPoints, const EdgeOptions* edgeOptions, const
                          CaliperOptions* caliperOptions, int*
                          numEdgePairs)
```

## Purpose

Finds edges along a path in an image, chooses pairs of the edges, and measures the distance between them.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image in which the function finds edges. |
| **points** | const Point* | The path along which the function detects edges. This parameter is required and cannot be NULL. |
| **numPoints** | int | The number of points in the points array. |
| **edgeOptions** | const EdgeOptions* | Describes how you want the function to find an edge. This parameter is required and cannot be NULL. |
| **caliperOptions** | const CaliperOptions* | Describes how you want the function to choose edge pairs. This parameter is required and cannot be NULL. |
| **numEdgePairs** | int* | On return, the number of edge pairs that the function found. Set this parameter to NULL if you do not need this information. |

## Parameter Discussion

**edgeOptions**—The `EdgeOptions` structure consists of the following entries:

- `threshold`—Specifies the threshold for the contrast of the edge. The function identifies only edges with a contrast greater than this value in the detection process.
- `width`—The number of pixels that the function averages to find the contrast at either side of the edge.

- steepness—The span, in pixels, of the slope of the edge projected along the path specified by the input points.
- subpixelType—The method for interpolating. Valid options are IMAQ_ZERO_ORDER, IMAQ_QUADRATIC, and IMAQ_CUBIC_SPLINE.
- subpixelDivisions—The number of samples the function obtains from a pixel. For example, set subpixelDivisions to 4 to split each pixel into four subpixels. The maximum number of subpixel divisions is 12.

**caliperOptions**—The CaliperOptions structure consists of the following entries:

- polarity—Specifies the edge polarity of the edge pairs. The following options are valid:
  - IMAQ_NONE—The function ignores the polarity of the edges.
  - IMAQ_RISING_FALLING—The polarity of the first edge is rising (dark to light) and the polarity of the second edge is falling (light to dark).
  - IMAQ_RISING_RISING—The polarity of the first edge is rising (dark to light) and the polarity of the second edge is rising (dark to light).
  - IMAQ_FALLING_FALLING—The polarity of the first edge is falling (light to dark) and the polarity of the second edge is falling (light to dark).
  - IMAQ_FALLING_RISING—The polarity of the first edge is falling (light to dark) and the polarity of the second edge is rising (dark to light).
- separation—The distance between edge pairs. If the edge pair has a separation greater than this value ± the separationDeviation, the function ignores the edge pair. Set this parameter to 0 to find all edge pairs.
- separationDeviation—Sets the range around the separation value. If you set separation to 0, the function ignores separationDeviation.

## Return Value

**CaliperReport***—On success, this function returns an array of information about each edge. This structure consists of the following values:

- edge1Contrast—The contrast of the first edge.
- edge1Coord—The coordinates of the first edge.
- edge2Contrast—The contrast of the second edge.
- edge2Coord—The coordinates of the second edge.
- separation—The distance between the two edges.

On failure, this function returns NULL. To get extended error information, call imaqGetLastError(). When you are finished with this information, dispose of this function by calling imaqDispose().

# imaqDetectRotation

## Usage

```
int = imaqDetectRotation(const Image* referenceImage, const Image*
                         testImage, PointFloat referenceCenter, PointFloat
                         testCenter, int radius, float precision, double*
                         angle)
```

## Purpose

Detects the rotational shift between two images, usually a reference image containing a part at a known orientation and another image containing the part in an unknown position. This function extracts pixel values around a circular region in the reference image and compares these values to the same region in the test image. The algorithm looks for the rotational shift between those two samples.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL

## Parameters

| Name | Type | Description |
|---|---|---|
| **referenceImage** | const Image* | The reference image. |
| **testImage** | const Image* | The test image. |
| **referenceCenter** | PointFloat | The center point of the circular region in the reference image. |
| **testCenter** | PointFloat | The center point of the circular region in the test image. |
| **radius** | int | The radius of the circles used to detect rotation. |
| **precision** | float | The sampling period, in degrees, of the pixel values that the function extracts from the circular region. The sampling period directly affects the speed of the function. If the sampling period is high (the number of samples along the circular region are few), the processing speed of the function increases at the cost of reduced accuracy in the computed rotational shift. In many cases, you do not need a precision higher than 5 degrees to position the regions of inspection of a part. If the sampling period is less than or equal to 0, the function returns an error. |
| **angle** | double* | On return, the angle, in degrees, between the two images. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqEdgeTool

## Usage

```
EdgeReport* = imaqEdgeTool(const Image* image, const Point* points, int
                           numPoints, const EdgeOptions* options, int*
                           numEdges)
```

## Purpose

Finds edges along a path in an image.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image in which to find edges. |
| **points** | const Point* | The path along which the function detects edges. This parameter is required and cannot be NULL. |
| **numPoints** | int | The number of points in the points array. |
| **options** | const EdgeOptions* | Describes how you want the function to find edges. This parameter is required and cannot be NULL. |
| **numEdges** | int* | On return, the number of edges that the function found. Set this parameter to NULL if you do not need this information. |

## Parameter Discussion

**options**—The `EdgeOptions` structure consists of the following entries:

- `threshold`—Specifies the threshold for the contrast of the edge. The function identifies only edges with a contrast greater than this value in the detection process.
- `width`—The number of pixels that the function averages to find the contrast at either side of the edge.
- `steepness`—The span, in pixels, of the slope of the edge projected along the path specified by the input points.
- `subpixelType`—The method for interpolating. Valid options include `IMAQ_QUADRATIC` and `IMAQ_CUBIC_SPLINE`.

- `subpixelDivisions`—The number of samples the function obtains from a pixel. For example, set subpixelDivisions to 4 to split each pixel into four subpixels. The maximum number of subpixels is 12.

## Return Value

**EdgeReport***—On success, this function returns an array of information about each edge. This structure consists of the following values:

- `position`—The position of the edge from the first point in the points array. This is a subpixel interpolated distance.
- `contrast`—The contrast at the edge.
- `polarity`—The polarity of the edge. The following values are valid:
  - `IMAQ_EDGE_RISING`—Indicates a rising edge (dark to light).
  - `IMAQ_EDGE_FALLING`—Indicates a falling edge (light to dark).
- `coordinates`—The coordinates of the edge.

On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with this information, dispose of this function by calling `imaqDispose()`.

# imaqLineGaugeTool

## Usage

```
int = imaqLineGaugeTool(const Image* image, Point start, Point end,
                        LineGaugeType method, const EdgeOptions*
                        edgeOptions, const CoordinateTransform*
                        reference, float* distance)
```

## Purpose

Measures the distance between selected edges of a line with high-precision subpixel accuracy. This function has the ability to work on a rotated or translated particle. If you supply coordinate transform information, the function transforms the line information you pass from the coordinate system of the particle to the coordinate system of the image.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image in which the function measures the distance between edges. |
| **start** | Point | The starting point of the line. |
| **end** | Point | The ending point of the line. |
| **method** | LineGaugeMethod | The measurement method. |
| **edgeOptions** | const EdgeOptions* | Describes how you want the function to find edges. If you set `method` to `IMAQ_POINT_TO_POINT`, the function ignores `edgeOptions`. If you set `method` to anything other than `IMAQ_POINT_TO_POINT`, this parameter is required and cannot be NULL. |
| **reference** | const CoordinateTransform* | The transform between the coordinate system of the particle and the image. See the purpose for a discussion of using a transform. Set this parameter to NULL if you do not want to apply a transform. |
| **distance** | float* | On return, the distance between edges and/or points. |

## Parameter Discussion

**method**—The following options are valid:

- `IMAQ_EDGE_TO_EDGE`—Measures from the first edge on the line to the last edge on the line.
- `IMAQ_EDGE_TO_POINT`—Measures from the first edge on the line to the end point of the line.
- `IMAQ_POINT_TO_EDGE`—Measures from the start point of the line to the first edge on the line.
- `IMAQ_POINT_TO_POINT`—Measures from the start point of the line to the end point of the line.

**edgeOptions**—The `EdgeOptions` structure consists of the following entries:

- `threshold`—Specifies the threshold value for the contrast of the edge. The function identifies only edges with a contrast greater than this value in the detection process.
- `width`—The number of pixels that the function averages to find the contrast at either side of the edge.
- `steepness`—The span, in pixels, of the slope of the edge projected along the path specified by the input points.
- `subpixelType`—The method for interpolating. Valid options include `IMAQ_QUADRATIC` and `IMAQ_CUBIC_SPLINE`.
- `subpixelDivisions`—The number of samples the function obtains from a pixel. For example, set `subpixelDivisions` to 4 to split each pixel into four subpixels. The maximum number of subpixel divisions is 12.

**reference**—The `CoordinateTransform` structure contains the following elements:

- `initialOrigin`—The origin of the initial coordinate system.
- `initialAngle`—The angle, in degrees, of the initial coordinate system.
- `finalOrigin`—The origin of the final coordinate system.
- `finalAngle`—The angle, in degrees, of the final coordinate system.

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqSimpleEdge

## Usage

```
PointFloat* = imaqSimpleEdge(const Image* image, const Point* points, int
                             numPoints, const SimpleEdgeOptions* options, int*
                             numEdges)
```

## Purpose

Finds prominent edges along an array of pixel coordinates. This function can return the first edge, both the first and the last edges, or all the edges.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image in which to find edges. |
| **points** | const Point* | The path along which the function detects edges. This parameter is required and cannot be NULL. |
| **numPoints** | int | The number of points in the points array. |
| **options** | const SimpleEdgeOptions* | Describes how you want the function to find edges. This parameter is required and cannot be NULL. |
| **numEdges** | int* | On return, the number of edges that the function found. Set this parameter to NULL if you do not need this information. |

## Parameter Discussion

**options**—The `SimpleEdgeOptions` structure consists of the following entries:

- `type`—Set this parameter to `IMAQ_ABSOLUTE` to make the values in `threshold` and `hysteresis` absolute values. Set this parameter to `IMAQ_RELATIVE` to make the values in `threshold` and `hysteresis` relative to the dynamic range of the given path.
- `threshold`—The pixel value at which an edge occurs.
- `hysteresis`—A value that helps determine edges in noisy images. If a pixel value crosses the given `threshold` value but does not exceed the value by the `hysteresis` value, the function does not consider the pixel to be part of an edge.
- `process`—Defines which edges to find. Valid options include the `IMAQ_FIRST`, `IMAQ_FIRST_AND_LAST`, or `IMAQ_ALL`.

- `subpixel`—Set this parameter to TRUE to find edges with subpixel accuracy by interpolating between points to find the crossing of the given threshold. Set this parameter to FALSE to report an edge as the point nearest the threshold crossing.

## Return Value

**PointFloat***—On success, this function returns an array of points indicating the location of the edges. On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with this information, dispose of the array by calling `imaqDispose()`.

# 15

# Operators

This chapter describes the Operator functions in IMAQ Vision for LabWindows/CVI.

## Operator Function Panels

Table 15-1 lists the Operator functions in a tree structure. The functions in the Operator class are grouped according to the types of operations they perform. The first-level heading in the tree is the name of the class. The second-level headings are names of function subclasses. The third-level headings are names of individual function panels. Each Operator function panel represents one function.

**Table 15-1.** Operator Function Tree

| Class/Panel Name | Function Name |
|---|---|
| Operators | |
| Arithmetic | |
| Add | `imaqAdd` |
| Add Constant | `imaqAddConstant` |
| Average | `imaqAverage` |
| Average Constant | `imaqAverageConstant` |
| Divide | `imaqDivide` |
| Divide Constant | `imaqDivideConstant` |
| Max | `imaqMax` |
| Max Constant | `imaqMaxConstant` |
| Min | `imaqMin` |
| Min Constant | `imaqMinConstant` |
| Modulo | `imaqModulo` |
| Modulo Constant | `imaqModuloConstant` |
| Multiply Divide | `imaqMulDiv` |
| Multiply | `imaqMultiply` |
| Multiply Constant | `imaqMultiplyConstant` |
| Subtract | `imaqSubtract` |
| Subtract Constant | `imaqSubtractConstant` |
| Logical | |
| And | `imaqAnd` |
| And Constant | `imaqAndConstant` |
| Compare | `imaqCompare` |
| Compare Constant | `imaqCompareConstant` |
| Logical Difference | `imaqLogicalDifference` |
| Logical Difference Constant | `imaqLogicalDifferenceConstant` |

**Table 15-1.** Operator Function Tree (Continued)

| Class/Panel Name | Function Name |
|---|---|
| Nand | `imaqNand` |
| Nand Constant | `imaqNandConstant` |
| Nor | `imaqNor` |
| Nor Constant | `imaqNorConstant` |
| Or | `imaqOr` |
| Or Constant | `imaqOrConstant` |
| Xnor | `imaqXnor` |
| Xnor Constant | `imaqXnorConstant` |
| Xor | `imaqXor` |
| Xor Constant | `imaqXorConstant` |

# Subclass Descriptions

Operator subclass descriptions are as follows:

- Arithmetic functions allow you to perform arithmetic operations between two images or between an image and a constant.
- Logical functions allow you to perform logic operations between two images or between an image and a constant.

# imaqAdd

## Usage

```
int = imaqAdd(Image* dest, const Image* sourceA, const Image* sourceB)
```

## Purpose

Adds two images.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **sourceA** | const Image* | The first image to add. |
| **sourceB** | const Image* | The second image to add. The image type of **sourceB** depends on the image type of **sourceA**, as follows:<br><br>If **sourceA** is IMAQ_IMAGE_U8, **sourceB** must be IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, or IMAQ_IMAGE_RGB.<br><br>If **sourceA** is IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, or IMAQ_IMAGE_COMPLEX, **sourceB** must be IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, or IMAQ_IMAGE_COMPLEX.<br><br>If **sourceA** is IMAQ_IMAGE_RGB, **sourceB** must be IMAQ_IMAGE_RGB or IMAQ_IMAGE_U8. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqAddConstant

## Usage

```
int = imaqAddConstant(Image* dest, const Image* source, PixelValue value)
```

## Purpose

Adds a constant value to each pixel in an image.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The image to which the function adds a scalar constant. |
| **value** | PixelValue | The value to add to the source image. **value** must correspond to the image type, as follows:<br>If the image is IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, or IMAQ_IMAGE_SGL, use the grayscale value of the PixelValue union.<br>If the image is IMAQ_IMAGE_COMPLEX, use the complex value of the PixelValue union.<br>If the image is IMAQ_IMAGE_RGB, use the rgb value of the PixelValue union. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqAnd

## Usage

```
int = imaqAnd(Image* dest, const Image* sourceA, const Image* sourceB)
```

## Purpose

Computes a bitwise AND between two images.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **sourceA** | const Image* | The first source image. |
| **sourceB** | const Image* | The second source image, which must be the same type of image as **sourceA**. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqAndConstant

## Usage

```
int = imaqAndConstant(Image* dest, const Image* source, PixelValue value)
```

## Purpose

Performs a bitwise AND between an image and a constant.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **value** | PixelValue | The value to AND to the source image. **value** must correspond to the image type, as follows:<br>If the image is IMAQ_IMAGE_U8 or IMAQ_IMAGE_I16, use the grayscale value of the PixelValue union.<br>If the image is IMAQ_IMAGE_HSL, use the hsl value of the PixelValue union.<br>If the image is IMAQ_IMAGE_RGB, use the rgb value of the PixelValue union. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqAverage

## Usage

```
int = imaqAverage(Image* dest, const Image* sourceA, const Image* sourceB)
```

## Purpose

Computes the average of two source images and places the result in the destination image.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **sourceA** | const Image* | The first source image. |
| **sourceB** | const Image* | The second source image, which must be the same type of image as **sourceA**. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqAverageConstant

## Usage

```
int = imaqAverageConstant(Image* dest, const Image* source, PixelValue value)
```

## Purpose

Computes the average of a source image and a constant and places the result into a destination image.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **value** | PixelValue | The value to average with the source image. Use the `grayscale` value of the PixelValue union. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqCompare

## Usage

```
int = imaqCompare(Image* dest, const Image* source, const Image*
                          compareImage, ComparisonFunction compare)
```

## Purpose

Copies the source image to the destination image in the following manner: If the given comparison between the source pixel value and its corresponding comparison image pixel value is true, the function sets the destination pixel value to 0. If the comparison is false, the function copies the source pixel value to the destination pixel.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **compareImage** | const Image* | The image to which the function compares the source image. This image must be the same type of image as **source**. |
| **compare** | ComparisonFunction | The method in which the function compares images. The following options are valid:<br>`IMAQ_CLEAR_LESS`—The comparison is true if the source pixel value is less than the comparison image pixel value.<br>`IMAQ_CLEAR_LESS_OR_EQUAL`—The comparison is true if the source pixel value is less than or equal to the comparison image pixel value.<br>`IMAQ_CLEAR_EQUAL`—The comparison is true if the source pixel value is equal to the comparison image pixel value.<br>`IMAQ_CLEAR_GREATER_OR_EQUAL`—The comparison is true if the source pixel value is greater than or equal to the comparison image pixel value.<br>`IMAQ_CLEAR_GREATER`—The comparison is true if the source pixel value is greater than the comparison image pixel value. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqCompareConstant

## Usage

```
int = imaqCompareConstant(Image* dest, const Image* source, PixelValue value,
                          ComparisonFunction compare)
```

## Purpose

Copies the source image to the destination image in the following manner: If the given comparison between the source pixel value and the given constant is true, the function sets the destination pixel value to 0. If the comparison is false, the function copies the source pixel value to the destination.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **value** | PixelValue | The value to compare to the source image. Use the `grayscale` value of the PixelValue union. |
| **compare** | `ComparisonFunction` | The method in which the function compares images. The following options are valid:<br>`IMAQ_CLEAR_LESS`—The comparison is true if the source pixel value is less than the comparison image pixel value.<br>`IMAQ_CLEAR_LESS_OR_EQUAL`—The comparison is true if the source pixel value is less than or equal to the comparison image pixel value.<br>`IMAQ_CLEAR_EQUAL`—The comparison is true if the source pixel value is equal to the comparison image pixel value.<br>`IMAQ_CLEAR_GREATER_OR_EQUAL`—The comparison is true if the source pixel value is greater than or equal to the comparison image pixel value.<br>`IMAQ_CLEAR_GREATER`—The comparison is true if the source pixel value is greater than the comparison image pixel value. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqDivide

## Usage

```
int = imaqDivide(Image* dest, const Image* sourceA, const Image* sourceB)
```

## Purpose

Divides two images.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **sourceA** | const Image* | The first image to divide. |
| **sourceB** | const Image* | The second image to divide. The image type of **sourceB** depends on the image type of **sourceA**, as follows:<br>If **sourceA** is IMAQ_IMAGE_I16, **sourceB** must be IMAQ_IMAGE_I16 or IMAQ_IMAGE_U8.<br>If **sourceA** is IMAQ_IMAGE_SGL, **sourceB** must be IMAQ_IMAGE_SGL, IMAQ_IMAGE_16, or IMAQ_IMAGE_U8.<br>If **sourceA** is IMAQ_IMAGE_COMPLEX, **sourceB** must be IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_SGL, IMAQ_IMAGE_16, or IMAQ_IMAGE_U8.<br>If **sourceA** is IMAQ_IMAGE_RGB, **sourceB** must be IMAQ_IMAGE_RGB or IMAQ_IMAGE_U8.<br>Otherwise **sourceB** must be the same type as **sourceA**. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqDivideConstant

## Usage

```
int = imaqDivideConstant(Image* dest, const Image* source, PixelValue value)
```

## Purpose

Divides each pixel in an image by a constant.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_COMPLEX`, `IMAQ_IMAGE_RGB`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The image by which the function divides a scalar constant. |
| **value** | PixelValue | The value by which the function divides the source image. **value** must correspond to the image type, as follows:<br>If the image is `IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, or `IMAQ_IMAGE_SGL`, use the `grayscale` value of the PixelValue union.<br>If the image is `IMAQ_IMAGE_COMPLEX`, use the `complex` value of the PixelValue union.<br>If the image is `IMAQ_IMAGE_RGB`, use the `rgb` value of the PixelValue union. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqLogicalDifference

## Usage

```
int = imaqLogicalDifference(Image* dest, const Image* sourceA, const Image*
                            sourceB)
```

## Purpose

Computes a bitwise logical difference (A AND NOT B) between two images.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **sourceA** | const Image* | The first source image. |
| **sourceB** | const Image* | The second source image, which must be the same type of image as **sourceA**. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqLogicalDifferenceConstant

## Usage

```
int = imaqLogicalDifferenceConstant(Image* dest, const Image* source,
                                    PixelValue value)
```

## Purpose

Performs a bitwise logical difference (A AND NOT B) between an image and a constant.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **value** | PixelValue | The value to AND NOT to the source image. **value** must correspond to the image type, as follows:<br>If the image is IMAQ_IMAGE_U8 or IMAQ_IMAGE_I16, use the grayscale value of the PixelValue union.<br>If the image is IMAQ_IMAGE_HSL, use the hsl value of the PixelValue union.<br>If the image is IMAQ_IMAGE_RGB, use the rgb value of the PixelValue union. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqMax

## Usage

```
int = imaqMax(Image* dest, const Image* sourceA, const Image* sourceB)
```

## Purpose

Copies the larger pixel value of the two sources into the destination for each pixel.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **sourceA** | const Image* | The first source image. |
| **sourceB** | const Image* | The second source image, which must be the same type of image as **sourceA**. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqMaxConstant

## Usage

```
int = imaqMaxConstant(Image* dest, const Image* source, PixelValue value)
```

## Purpose

Copies the source image to the destination in the following manner: If the source image pixel value is greater than the given constant, the function copies the source pixel to the destination. Otherwise the function copies the constant to the destination.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The first source image |
| **value** | PixelValue | The value to use in the computation. Use the `grayscale` value of the PixelValue union. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqMin

## Usage

```
int = imaqMin(Image* dest, const Image* sourceA, const Image* sourceB)
```

## Purpose

Copies the smaller pixel value of the two source pixels into the destination for each pixel.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **sourceA** | const Image* | The first source image. |
| **sourceB** | const Image* | The second source image, which must be the same type of image as **sourceA**. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqMinConstant

## Usage

```
int = imaqMinConstant(Image* dest, const Image* source, PixelValue value)
```

## Purpose

Copies the source image to the destination in the following manner: If the source image pixel value is less than the given constant, the function copies the source pixel to the destination. Otherwise the function copies the constant to the destination.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The first source image |
| **value** | PixelValue | The value to use in the computation. Use the `grayscale` value of the PixelValue union. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqModulo

## Usage

```
int = imaqModulo(Image* dest, const Image* sourceA, const Image* sourceB)
```

## Purpose

Modulo divides two images.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_RGB

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **sourceA** | const Image* | The first image to modulo divide. |
| **sourceB** | const Image* | The second image to modulo divide. The image type of **sourceB** depends on the image type of **sourceA**, as follows:<br>If **sourceA** is IMAQ_IMAGE_I16, **sourceB** must be IMAQ_IMAGE_I16 or IMAQ_IMAGE_U8.<br>If **sourceA** is IMAQ_IMAGE_SGL, **sourceB** must be IMAQ_IMAGE_SGL, IMAQ_IMAGE_16, or IMAQ_IMAGE_U8.<br>If **sourceA** is IMAQ_IMAGE_RGB, **sourceB** must be IMAQ_IMAGE_RGB or IMAQ_IMAGE_U8.<br>Otherwise **sourceB** must be the same type as **sourceA**. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqModuloConstant

## Usage

```
int = imaqModuloConstant(Image* dest, const Image* source, PixelValue value)
```

## Purpose

Performs a modulo division operation with each pixel in an image by a constant.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_RGB

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The image to be modulo divided by the scalar constant. |
| **value** | PixelValue | The value to use as the divisor in the operation. **value** must correspond to the image type, as follows:<br>If the image is IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, or IMAQ_IMAGE_SGL, use the grayscale value of the PixelValue union.<br>If the image is IMAQ_IMAGE_RGB, use the rgb value of the PixelValue union. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqMulDiv

## Usage

```
int = imaqMulDiv(Image* dest, const Image* sourceA, const Image* sourceB,
                 float value)
```

## Purpose

Computes a ratio between the two source images. You find the ratio by multiplying each pixel value in the first source image by the constant value you supply. This result is divided by the corresponding pixel in the second source, and the final result is stored in the destination image. You can use this function to correct a background if the background is lighter than the image. In a background correction, the first source image is the acquired image and the second source image is the background image.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_RGB`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **sourceA** | const Image* | The first source image. |
| **sourceB** | const Image* | The second source image, which must be the same type of image as **sourceA**. |
| **value** | float | The value by which the function multiplies the first image. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqMultiply

## Usage

```
int = imaqMultiply(Image* dest, const Image* sourceA, const Image* sourceB)
```

## Purpose

Multiplies two images.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_RGB

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **sourceA** | const Image* | The first image to multiply. |
| **sourceB** | const Image* | The second image to multiply. The image type of **sourceB** depends on the image type of **sourceA**, as follows:<br>If **sourceA** is IMAQ_IMAGE_U8, **sourceB** must be IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_COMPLEX, or IMAQ_IMAGE_RGB.<br>If **sourceA** is IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, or IMAQ_IMAGE_COMPLEX, **sourceB** must be IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, or IMAQ_IMAGE_COMPLEX.<br>If **sourceA** is IMAQ_IMAGE_RGB, **sourceB** must be IMAQ_IMAGE_RGB or IMAQ_IMAGE_U8. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqMultiplyConstant

## Usage

```
int = imaqMultiplyConstant(Image* dest, const Image* source, PixelValue
                           value)
```

## Purpose

Multiplies each pixel in an image by a constant.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_COMPLEX`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The image by which the function multiplies a scalar constant. |
| **value** | PixelValue | The value by which to multiply. **value** must correspond to the image type, as follows:<br>If the image is `IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, or `IMAQ_IMAGE_SGL`, use the `grayscale` value of the PixelValue union.<br>If the image is `IMAQ_IMAGE_COMPLEX`, use the `complex` value of the PixelValue union.<br>If the image is `IMAQ_IMAGE_RGB`, use the `rgb` value of the PixelValue union. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqNand

## Usage

```
int = imaqNand(Image* dest, const Image* sourceA, const Image* sourceB)
```

## Purpose

Computes a bitwise NAND between two images.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **sourceA** | const Image* | The first source image. |
| **sourceB** | const Image* | The second source image, which must be the same type of image as **sourceA**. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqNandConstant

## Usage

```
int = imaqNandConstant(Image* dest, const Image* source, PixelValue value)
```

## Purpose

Performs a bitwise NAND between an image and a constant.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **value** | PixelValue | The value to NAND with the source image. **value** must correspond to the image type, as follows:<br>If the image is IMAQ_IMAGE_U8 or IMAQ_IMAGE_I16, use the grayscale value of the PixelValue union.<br>If the image is IMAQ_IMAGE_HSL, use the hsl value of the PixelValue union.<br>If the image is IMAQ_IMAGE_RGB, use the rgb value of the PixelValue union. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqNor

## Usage

```
int = imaqNor(Image* dest, const Image* sourceA, const Image* sourceB)
```

## Purpose

Computes a bitwise NOR between two images.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **sourceA** | const Image* | The first source image. |
| **sourceB** | const Image* | The second source image, which must be the same type of image as **sourceA**. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqNorConstant

## Usage

```
int = imaqNorConstant(Image* dest, const Image* source, PixelValue value)
```

## Purpose

Performs a bitwise NOR between an image and a constant.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **value** | PixelValue | The value to NOR with the source image. **value** must correspond to the image type, as follows:<br>If the image is IMAQ_IMAGE_U8 or IMAQ_IMAGE_I16, use the grayscale value of the PixelValue union.<br>If the image is IMAQ_IMAGE_HSL, use the hsl value of the PixelValue union.<br>If the image is IMAQ_IMAGE_RGB, use the rgb value of the PixelValue union. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqOr

## Usage

```
int = imaqOr(Image* dest, const Image* sourceA, const Image* sourceB)
```

## Purpose

Computes a bitwise OR between two images.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **sourceA** | const Image* | The first source image. |
| **sourceB** | const Image* | The second source image, which must be the same type of image as **sourceA**. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqOrConstant

## Usage

```
int = imaqOrConstant(Image* dest, const Image* source, PixelValue value)
```

## Purpose

Performs a bitwise OR between an image and a constant.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **value** | PixelValue | The value to OR to the source image. **value** must correspond to the image type, as follows:<br>If the image is IMAQ_IMAGE_U8 or IMAQ_IMAGE_I16, use the grayscale value of the PixelValue union.<br>If the image is IMAQ_IMAGE_HSL, use the hsl value of the PixelValue union.<br>If the image is IMAQ_IMAGE_RGB, use the rgb value of the PixelValue union. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqSubtract

## Usage

```
int = imaqSubtract(Image* dest, const Image* sourceA, const Image* sourceB)
```

## Purpose

Subtracts two images.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL, IMAQ_IMAGE_RGB, IMAQ_IMAGE_COMPLEX

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **sourceA** | const Image* | The first image to subtract. |
| **sourceB** | const Image* | The second image to subtract. The type of the **sourceB** image depends on the type of the **sourceA**, as follows:<br>If **sourceA** is IMAQ_IMAGE_I16, **sourceB** must be IMAQ_IMAGE_I16 or IMAQ_IMAGE_U8.<br>If **sourceA** is IMAQ_IMAGE_SGL, **sourceB** must be IMAQ_IMAGE_SGL, IMAQ_IMAGE_16, or IMAQ_IMAGE_U8.<br>If **sourceA** is IMAQ_IMAGE_RGB, **sourceB** must be IMAQ_IMAGE_RGB or IMAQ_IMAGE_U8.<br>If **sourceA** is IMAQ_IMAGE_COMPLEX, **sourceB** must be IMAQ_IMAGE_COMPLEX, IMAQ_IMAGE_SGL, IMAQ_IMAGE_I16, or IMAQ_IMAGE_U8.<br>Otherwise **sourceB** must be the same type as **sourceA**. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqSubtractConstant

## Usage

```
int = imaqSubtractConstant(Image* dest, const Image* source, PixelValue
                           value)
```

## Purpose

Subtracts each pixel in an image by a constant.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_COMPLEX`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The image from which the function subtracts a scalar constant. |
| **value** | PixelValue | The value to subtract from the source image pixels. **value** must correspond to the image type, as follows:<br>If the image is `IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, or `IMAQ_IMAGE_SGL`, use the `grayscale` value of the PixelValue union.<br>If the image is `IMAQ_IMAGE_COMPLEX`, use the `complex` value of the PixelValue union.<br>If the image is `IMAQ_IMAGE_RGB`, use the `rgb` value of the PixelValue union. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqXnor

## Usage

```
int = imaqXnor(Image* dest, const Image* sourceA, const Image* sourceB)
```

## Purpose

Computes a bitwise XNOR between two images.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_RGB`, `IMAQ_IMAGE_HSL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **sourceA** | const Image* | The first source image. |
| **sourceB** | const Image* | The second source image, which must be the same type of image as **sourceA**. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqXnorConstant

## Usage

```
int = imaqXnorConstant(Image* dest, const Image* source, PixelValue value)
```

## Purpose

Performs a bitwise XNOR between an image and a constant.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **value** | PixelValue | The value to XNOR with the source image. **value** must correspond to the image type, as follows:<br>If the image is IMAQ_IMAGE_U8 or IMAQ_IMAGE_I16, use the grayscale value of the PixelValue union.<br>If the image is IMAQ_IMAGE_HSL, use the hsl value of the PixelValue union.<br>If the image is IMAQ_IMAGE_RGB, use the rgb value of the PixelValue union. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# imaqXor

## Usage

```
int = imaqXor(Image* dest, const Image* sourceA, const Image* sourceB)
```

## Purpose

Computes a bitwise XOR between two images.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **sourceA** | const Image* | The first source image. |
| **sourceB** | const Image* | The second source image, which must be the same type of image as **sourceA**. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqXorConstant

## Usage

```
int = imaqXorConstant(Image* dest, const Image* source, PixelValue value)
```

## Purpose

Performs a bitwise XOR between an image and a constant.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_RGB, IMAQ_IMAGE_HSL

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image. |
| **value** | PixelValue | The value to XOR with the source image. **value** must correspond to the image type, as follows:<br>If the image is IMAQ_IMAGE_U8 or IMAQ_IMAGE_I16, use the grayscale value of the PixelValue union.<br>If the image is IMAQ_IMAGE_HSL, use the hsl value of the PixelValue union.<br>If the image is IMAQ_IMAGE_RGB, use the rgb value of the PixelValue union. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call imaqGetLastError().

# 16

# Analytic Geometry

This chapter describes the Analytic Geometry functions in IMAQ Vision for LabWindows/CVI. Analytic Geometry functions allow you to perform analytical geometry operations, such as obtaining points on a contour within an image or obtaining the angle between two lines.

## Analytic Geometry Function Panels

Table 16-1 lists the Analytic Geometry functions in a tree structure. The functions in the Analytic Geometry class are grouped according to the types of operations they perform. The first-level heading in the tree is the name of the class. The second-level headings are names of individual function panels. Each Analytic Geometry function panel represents one function.

**Table 16-1.** Analytic Geometry Function Tree

| Class/Panel Name | Function Name |
|---|---|
| Analytic Geometry | |
| Best Circle | `imaqBestCircle` |
| Coordinate Reference | `imaqCoordinateReference` |
| Get Angle | `imaqGetAngle` |
| Get Points On Contour | `imaqGetPointsOnContour` |
| Get Points On Line | `imaqGetPointsOnLine` |
| Interpolate Points | `imaqInterpolatePoints` |

# imaqBestCircle

## Usage

```
int = imaqBestCircle(const PointFloat* points, int numPoints, PointFloat*
                     center, double* radius)
```

## Purpose

Returns the circle that best fits the given points.

## Parameters

| Name | Type | Description |
|---|---|---|
| **points** | const PointFloat* | The array of points to fit to the edge of the circle. |
| **numPoints** | int | The number of points in the supplied array. You must supply at least three points. |
| **center** | PointFloat* | On return, filled with the coordinates of the center of the circle. Set this parameter to NULL if you do not need this information. |
| **radius** | double* | On return, filled with the radius of the center of the circle. Set this parameter to NULL if you do not need this information. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqCoordinateReference

## Usage

```
int = imaqCoordinateReference(const Point* points, ReferenceMode mode,
                              Point* origin, float* angle)
```

## Purpose

Builds a reference for any arbitrary coordinate system with respect to the image plane. The reference of the coordinate system is specified as the position of the origin of the coordinate system and the orientation of its x-axis with respect to that of the image plane. Figure 16-1 illustrates how the function determines the reference of a coordinate system.

**Note** The y-axis of an image is inverted relative to the Cartesian coordinate system.

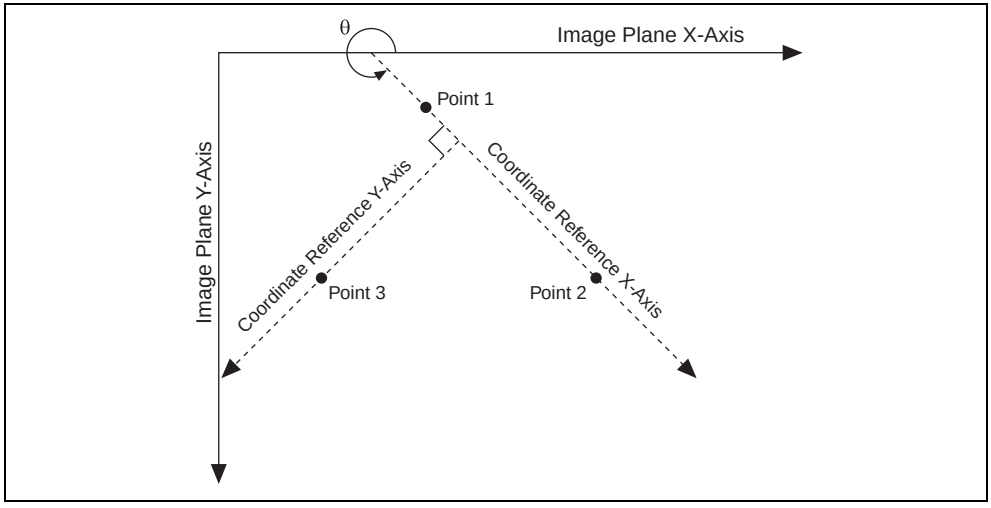



**Figure 16-1.** Reference of a Coordinate System

## Parameters

| Name | Type | Description |
|---|---|---|
| **points** | const Point* | An array of points defining the coordinate reference. If **mode** is `IMAQ_COORD_X_Y`, the **points** array must have three points. If **mode** is `IMAQ_COORD_ORIGIN_X`, the **points** array must have two points. |
| **mode** | ReferenceMode | Specifies the method that the function uses to calculate the coordinate reference.<br>Set this parameter to `IMAQ_COORD_X_Y` to require three elements in the **points** array. The first two points are on the x-axis of the coordinate reference, and the third point is on the y-axis.<br>Set this parameter to `IMAQ_COORD_ORIGIN_X` to require two elements in the **points** array. The first point is the origin of the coordinate reference, and the second point is on the x-axis. |
| **origin** | Point* | On return, the origin of the coordinate system. Set this parameter to NULL if you do not need this information. |
| **angle** | float* | On return, the angle, in degrees, of the x-axis of the coordinate system relative to the x-axis of an image. Set this parameter to NULL if you do not need this information. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetAngle

## Usage

```
int = imaqGetAngle(PointFloat start1, PointFloat end1, PointFloat start2,
                   PointFloat end2, float* angle)
```

## Purpose

Returns the angle, in degrees, between two lines. The returned angle represents the rotation around **start1** required so the line from **start1** to **end1** is parallel with the line from **start2** to **end2**. Figure 16-2 illustrates how the function calculates the angle between two lines.

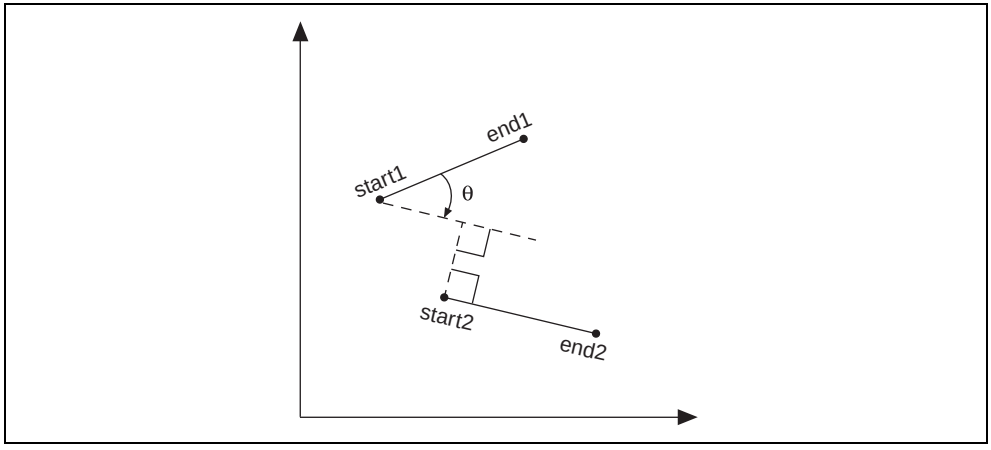



**Figure 16-2.** Calculating the Angle Between Two Lines

## Parameters

| Name | Type | Description |
|---|---|---|
| **start1** | PointFloat | The start point of the first line. |
| **end1** | PointFloat | The end point of the first line. |
| **start2** | PointFloat | The start point of the second line. |
| **end2** | PointFloat | The end point of the second line. |
| **angle** | float* | On return, the angle, in degrees, between the lines. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqGetPointsOnContour

## Usage

```
SegmentInfo* = imaqGetPointsOnContour(const Image* image, int* numSegments)
```

## Purpose

Finds the number of edge segments in an image with outlined edges and returns the points of each segment.

## Image Types Supported

IMAQ_IMAGE_U8

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image in which to find the segments. |
| **numSegments** | int* | On return, the number of segments found. Set this parameter to NULL if you do not need this information. |

## Return Value

SegmentInfo*—On success, this function returns an array of information about the segments. The `SegmentInfo` structure contains the following elements:

- `numberOfPoints`—The number of points in the segment.
- `isOpen`—If TRUE, the contour is open. If FALSE, the contour is closed.
- `weight`—The significance of the edge in terms of the gray values that constitute the edge.
- `points`—The points of the segment. This is a `ContourPoint`.

The `ContourPoint` structure contains the following information:

- `x`—The x-coordinate value in the image.
- `y`—The y-coordinate value in the image.
- `curvature`—The change in slope at this edge point of the segment.
- `xDisplacement`—The x displacement of the current edge pixel from a cubic spline fit of the current edge segment.
- `yDisplacement`—The y displacement of the current edge pixel from a cubic spline fit of the current edge segment.

On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with the information, dispose of it by calling `imaqDispose()`.

# imaqGetPointsOnLine

## Usage

```
Point* = imaqGetPointsOnLine(Point start, Point end, int* numPoints)
```

## Purpose

Given the endpoints of a line, this function returns all the points comprising the line.

## Parameters

| Name | Type | Description |
|---|---|---|
| **start** | Point | The first point of the line. |
| **end** | Point | The last point of the line. |
| **numPoints** | int* | On return, the number of points put into the returned array. Set this parameter to NULL if you do not need this information. |

## Return Value

**Point***—On success, this function returns an array of the points on the line. On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with this array, dispose of it by calling `imaqDispose()`.

# imaqInterpolatePoints

## Usage

```
float* = imaqInterpolatePoints(const Image* image, const Point* points, int
                               numPoints, InterpolationMethod method, int
                               subpixel, int* interpCount)
```

## Purpose

Interpolates the pixel values of an image over specified points.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image containing the values to interpolate. |
| **points** | const Point* | The points over which to interpolate. All the points in this array must be within the image. This parameter is required and cannot be NULL. |
| **numPoints** | int | The number of points in the input points array. |
| **method** | InterpolationMethod | Specifies the method for the interpolation. The following options are valid:<br>IMAQ_BILINEAR<br>IMAQ_QUADRATIC<br>IMAQ_CUBIC_SPLINE |
| **subpixel** | int | The number of subdivisions into which to interpolate. For example, a value of 0 causes the function to return only the pixel values at the given points, whereas a value of 1 returns the pixel values at the given points and at the midpoint of each pair. |
| **interpCount** | int* | On return, the number of interpolated values in the array returned by the function. Set this parameter to NULL if you do not need this information. |

## Return Value

**float***—On success, this function returns an array of the interpolated values. On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with this array, dispose of it by calling `imaqDispose()`.

# 17

# Frequency Domain Analysis

This chapter describes the Frequency Domain Analysis functions in IMAQ Vision for LabWindows/CVI. Frequency Domain Analysis functions allow you to convert images between the spatial and frequency domains and to analyze images in the frequency domain.

## Frequency Domain Analysis Function Panels

Table 17-1 lists the Frequency Domain Analysis functions in a tree structure. The functions in the Frequency Domain Analysis class are grouped according to the types of operations they perform. The first-level heading in the tree is the name of the class. The second-level headings are names of individual function panels. Each Frequency Domain Analysis function panel represents one function.

**Table 17-1.** Frequency Domain Analysis Function Tree

| Class/Panel Name | Function Name |
|---|---|
| Frequency Domain Analysis | |
| Attenuate | `imaqAttenuate` |
| Conjugate | `imaqConjugate` |
| FFT | `imaqFFT` |
| Flip Frequencies | `imaqFlipFrequencies` |
| Inverse FFT | `imaqInverseFFT` |
| Truncate | `imaqTruncate` |

# imaqAttenuate

## Usage

```
int = imaqAttenuate(Image* dest, const Image* source, AttenuateMode highlow)
```

## Purpose

Attenuates the frequencies of a complex image.

## Image Type Supported

`IMAQ_IMAGE_COMPLEX`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The image to attenuate. |
| **highlow** | AttenuateMode | The frequencies to attenuate. Set this parameter to `IMAQ_ATTENUATE_LOW` to attenuate low frequencies. Set this parameter to `IMAQ_ATTENUATE_HIGH` to attenuate high frequencies. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqConjugate

## Usage

```
int = imaqConjugate(Image* dest, const Image* source)
```

## Purpose

Computes the conjugate of a complex image.

## Image Type Supported

`IMAQ_IMAGE_COMPLEX`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The source image whose conjugate the function calculates. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqFFT

## Usage

```
int = imaqFFT(Image* dest, const Image* source)
```

## Purpose

Computes the Fourier transform of an image. The image can be any size, but the function works faster if the image dimensions are powers of 2. The destination image must be different than the source image to perform this operation.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, `IMAQ_IMAGE_COMPLEX`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. The destination image must be a complex image and must be different than source image. |
| **source** | const Image* | The image whose Fourier transform the function computes. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqFlipFrequencies

## Usage

```
int = imaqFlipFrequencies(Image* dest, const Image* source)
```

## Purpose

Transposes the high and low frequencies of a complex image.

## Image Type Supported

IMAQ_IMAGE_COMPLEX

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The complex image whose frequencies the function flips. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqInverseFFT

## Usage

```
int = imaqInverseFFT(Image* dest, const Image* source)
```

## Purpose

Takes the inverse Fourier transform of an image. The destination image must be different than the source image to perform this operation.

## Image Type Supported

`IMAQ_IMAGE_COMPLEX`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. Valid image types are `IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`, or `IMAQ_IMAGE_COMPLEX`. The destination image must be different from the source image. |
| **source** | const Image* | The image whose inverse Fourier transform the function takes. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqTruncate

## Usage

```
int = imaqTruncate(Image* dest, const Image* source, TruncateMode highlow,
                   float ratioToKeep)
```

## Purpose

Truncates the frequencies of a complex image.

## Image Type Supported

`IMAQ_IMAGE_COMPLEX`

## Parameters

| Name | Type | Description |
|---|---|---|
| **dest** | Image* | The destination image. |
| **source** | const Image* | The image whose frequencies the function truncates. |
| **highlow** | TruncateMode | Set this parameter to `IMAQ_TRUNCATE_LOW` to truncate low frequencies. Set this parameter to `IMAQ_TRUNCATE_HIGH` to truncate high frequencies. |
| **ratioToKeep** | float | Specifies the ratio of frequencies that the function retains. For example, set this parameter to 0.1 to retain 10% of the frequencies and attenuate 90% of the frequencies. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# 18

# Barcode

This chapter describes the Barcode function in IMAQ Vision for LabWindows/CVI. The Barcode function, `imaqReadBarcode`, allows you to read a barcode from an image.

## Barcode Function Panels

Table 18-1 lists the Barcode function in a tree structure. The Barcode function panel represents one function.

**Table 18-1.** Barcode Function Tree

| **Class/Panel Name** | **Function Name** |
|---|---|
| Barcode I/O | |
| Read Barcode | `imaqReadBarcode` |

# imaqReadBarcode

## Usage

```
BarcodeInfo* = imaqReadBarcode(const Image* image, BarcodeType type,
                               const ROI* roi, int validate)
```

## Purpose

This function reads a barcode from an image.

## Image Types Supported

IMAQ_IMAGE_U8, IMAQ_IMAGE_I16, IMAQ_IMAGE_SGL

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image containing the barcode to read. |
| **type** | BarcodeType | The type of the barcode to read, as follows:<br>IMAQ_I2_OF_5 IMAQ_EAN8<br>IMAQ_CODE39 IMAQ_EAN13<br>IMAQ_CODE93 IMAQ_CODABAR<br>IMAQ_CODE128 IMAQ_MSI<br>IMAQ_UPCA |
| **roi** | const ROI* | An ROI specifying the location of the barcode in the image. Set this parameter to NULL to use the entire image. |
| **validate** | int | If **type** is IMAQ_I2_OF_5, IMAQ_CODE39, or IMAQ_CODABAR, set **validate** to TRUE to use the error correction information of the barcode to validate the data. If **type** is not IMAQ_I2_OF_5, IMAQ_CODE39, or IMAQ_CODABAR, or if you set **validate** to FALSE, no validation occurs. |

## Return Value

**BarcodeInfo***—On success, this function returns a pointer to a BarcodeInfo structure. This structure contains the following elements:

- outputString—A string containing the decoded barcode data.
- outputChar1—The contents of this character depend on the barcode type, as follows:
  - IMAQ_CODABAR—The start character.
  - IMAQ_CODE128—The FNC value.
  - IMAQ_EAN8 and IMAQ_EAN13—The first country code.

For all other barcode types, the function sets `outputChar1` to 0.

- `outputChar2`—The contents of this character depend on the barcode type, as follows:
  - `IMAQ_CODABAR`—The stop character.
  - `IMAQ_EAN8` and `IMAQ_EAN13`—The second country code.
  - `IMAQ_UPCA`—The system number.

  For all other barcode types, the function sets `outputChar2` to 0.
- `confidenceLevel`—A quality measure of the decoded barcode ranging from 0 to 100, with 100 being the best. This value weighs the error in the widths of the bars and spaces with the size of the character in the barcode. In general, a confidence level of less than 80 means the decoded string is suspect.

**Note** The **confidenceLevel** is particularly useful in decoding `IMAQ_EAN13` barcodes because twelve of the thirteen data values are encoded as characters in the barcode, and the thirteenth value is encoded by the parity of the first 12 encoded characters.

- `type`—The type of barcode.

On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with this structure, dispose of it by calling `imaqDispose()`.

# 19

# LCD

This chapter describes the LCD functions in IMAQ Vision for LabWindows/CVI. LCD functions allow you to isolate and read the value of a seven-segment LCD.

## LCD Function Panels

Table 19-1 lists the LCD functions in a tree structure. The functions in the LCD class are grouped according to the types of operations they perform. The first-level heading in the tree is the name of the class. The second-level headings are names of individual function panels. Each LCD function panel represents one function.

**Table 19-1.** LCD Function Tree

| **Class/Panel Name** | **Function Name** |
|---|---|
| LCD | |
| Find LCD Segments | `imaqFindLCDSegments` |
| Read LCD | `imaqReadLCD` |

# imaqFindLCDSegments

## Usage

```
int = imaqFindLCDSegments(ROI* roi, const Image* image, const
                          LCDOptions* options)
```

## Purpose

Takes an ROI around a seven-segment LCD and transforms the ROI to be a set of rectangles around each LCD digit. You can then process this modified ROI with the `imaqReadLCD()` function.

**Note** All segments of the LCD must be on for this function to properly find the digits.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **roi** | ROI* | The region of interest to transform. |
| **image** | const Image* | The image containing the LCD. All segments of the LCD must be lit. |
| **options** | const LCDOptions* | Controls how the function performs the search. |

## Parameter Discussion

**options**—The `LCDOptions` structure contains the following elements:

- `litSegments`—Set this parameter to TRUE if the segments are brighter than the background. Set this parameter to FALSE if the segments are darker than the background.
- `threshold`—Determines whether a segment is ON or OFF. A segment is ON if the standard deviation of the pixels along a line profile across the segment is greater than this threshold. Increase the threshold value when using images with high contrast. Decrease the threshold value when using images with low contrast.
- `sign`—Indicates whether the function must read the sign of the indicator. Set this parameter to TRUE to search for the sign. Set this parameter to FALSE to not search for the sign.
- `decimalPoint`—Determines whether to look for a decimal separator after each digit. Set this parameter to TRUE to search for the separator. Set this parameter to FALSE to not search for the separator.

Set the **options** parameter to NULL to use the default options, as follows:

- `litSegments`—FALSE
- `threshold`—8
- `sign`—FALSE
- `decimalPoint`—FALSE

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# imaqReadLCD

## Usage

```
LCDReport* = imaqReadLCD(const Image* image, const ROI* roi, const
                         LCDOptions* options)
```

## Purpose

Reads the numeric value of a seven-segment LCD.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image containing the LCD to read. |
| **roi** | const ROI* | An ROI consisting of rectangles around each of the digits of the LCD. Generate this ROI by calling `imaqFindLCDSegments()`. |
| **options** | const LCDOptions* | Controls how the LCD is read. |

## Parameter Discussion

**options**—The `LCDOptions` structure contains the following elements:

- `litSegments`—Set this parameter to TRUE if the segments are brighter than the background. Set this parameter to FALSE if the segments are darker than the background.
- `threshold`—Determines whether a segment is ON or OFF. A segment is ON if the standard deviation of the pixels along a line profile across the segment is greater than this threshold. Increase the threshold value when using images with high contrast. Decrease the threshold value when using images with low contrast.
- `sign`—Indicates whether the function must read the sign of the indicator. Set this parameter to TRUE to search for the sign. Set this parameter to FALSE to not search for the sign.
- `decimalPoint`—Determines whether to look for a decimal separator after each digit. Set this parameter to TRUE to search for the separator. Set this parameter to FALSE to not search for the separator.

Set the **options** parameter to NULL to use the default options, as follows:

- `litSegments`—FALSE
- `threshold`—8

- sign—FALSE
- decimalPoint—FALSE

## Return Value

LCDReport*—On success, this function returns a structure describing the state of the LCD. The LCDReport structure consists of the following elements:

- text—A string of the characters of the LCD.
- segmentInfo—An array of LCDSegment structures describing which segments of each digit are on.
- numCharacters—The number of characters that the function read. This describes the number of elements in the segmentInfo array.

The LCDSegment structure consists of the following elements. See Figure 19-1 for more information.

- a—True if the **a** segment is on.
- b—True if the **b** segment is on.
- c—True if the **c** segment is on.
- d—True if the **d** segment is on.
- e—True if the **e** segment is on.
- f—True if the **f** segment is on.
- g—True if the **g** segment is on.

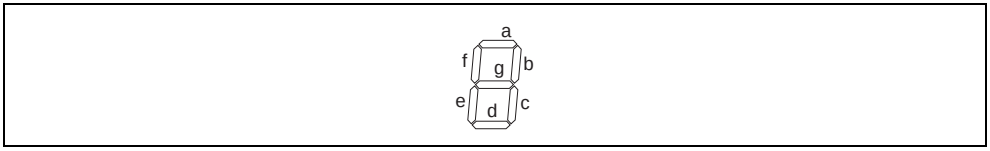



**Figure 19-1.** Segments of an LCD

On failure, this function returns NULL. To get extended error information, call imaqGetLastError().

# 20

# Meter

This chapter describes the Meter functions in IMAQ Vision for LabWindows/CVI. Meter functions allow you to identify the arc information of and then read a meter.

## Meter Function Panels

Table 20-1 lists the Meter functions in a tree structure. The functions in the Meter class are grouped according to the types of operations they perform. The first-level heading in the tree is the name of the class. The second-level headings are names of individual function panels. Each Meter function panel represents one function.

**Table 20-1.** Meter Function Tree

| **Class/Panel Name** | **Function Name** |
|---|---|
| Meter | |
| Get Meter Arc | `imaqGetMeterArc` |
| Read Meter | `imaqReadMeter` |

# imaqGetMeterArc

## Usage

```
MeterArc* = imaqGetMeterArc(int lightNeedle, MeterArcMode mode, const ROI*
                            roi, PointFloat base, PointFloat start,
                            PointFloat end)
```

## Purpose

Returns the arc information of a meter. The `imaqReadMeter()` function uses this information to read a meter.

## Parameters

| Name | Type | Description |
|---|---|---|
| **lightNeedle** | int | Set this parameter to TRUE to find a light-colored needle on a dark background. Set this parameter to FALSE to find a dark-colored needle on a light background. |
| **mode** | MeterArcMode | Describes how to determine the arc. Set this parameter to `IMAQ_METER_ARC_ROI` to use the **roi** parameter and ignore the **base**, **start**, and **end** parameters. Set this parameter to `IMAQ_METER_ARC_POINTS` to use the **base**, **start**, and **end** parameters and ignore the **roi** parameter. |
| **roi** | const ROI* | Consist of two lines, each drawn from the tip of the needle to its base. The first line represents the minimum position of the needle, and the second line represents the maximum position of the needle. If **mode** is `IMAQ_METER_ARC_ROI`, **roi** is required and cannot be NULL. If **mode** is `IMAQ_METER_ARC_POINTS`, the function ignores **roi**, and the parameter can be NULL. |
| **base** | PointFloat | The location of the base of the needle. If **mode** is `IMAQ_METER_ARC_POINTS`, **base** is required and cannot be NULL. If **mode** is `IMAQ_METER_ARC_ROI`, the function ignores **base**. |
| **start** | PointFloat | The location of the tip of the needle when the needle is at the minimum sweep position. If **mode** is `IMAQ_METER_ARC_POINTS`, **start** is required and cannot be NULL. If **mode** is `IMAQ_METER_ARC_ROI`, the function ignores **start**. |

| Name | Type | Description |
|---|---|---|
| **end** | PointFloat | The location of the tip of the needle when the needle is at the maximum sweep position. If **mode** is `IMAQ_METER_ARC_POINTS`, **end** is required. If **mode** is `IMAQ_METER_ARC_ROI`, the function ignores **end**. |

## Return Value

**MeterArc***—On success, this function returns a structure describing the arc across which a meter sweeps. On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. When you are finished with this information, dispose of it by calling `imaqDispose()`.

# imaqReadMeter

## Usage

```
int = imaqReadMeter(const Image* image, const MeterArc* arcInfo, double*
                          percentage, PointFloat* endOfNeedle)
```

## Purpose

Reads a meter. You must have already determined the arc information with `imaqGetMeterArc()`.

## Image Types Supported

`IMAQ_IMAGE_U8`, `IMAQ_IMAGE_I16`, `IMAQ_IMAGE_SGL`

## Parameters

| Name | Type | Description |
|---|---|---|
| **image** | const Image* | The image of the meter to read. |
| **arcInfo** | const MeterArc* | Information about the meter's arc. This information is returned by `imaqGetMeterArc()`. |
| **percentage** | double* | Returns the current sweep position of the needle in comparison to the maximum sweep position, expressed as a percentage. For example, a value of 100 indicates the needle is at the maximum sweep position. Set this parameter to NULL if you do not need this information. |
| **endOfNeedle** | PointFloat* | Returns the location of the endpoint of the needle. Set this parameter to NULL if you do not need this information. |

## Return Value

**int**—On success, this function returns a non-zero value. On failure, this function returns 0. To get extended error information, call `imaqGetLastError()`.

# 21

# Utilities

This chapter describes the Utilities functions in IMAQ Vision for LabWindows/CVI. Utilities functions allow you to set up structures that you can embed in other functions to eliminate the need to declare certain types of variables such as `Point`, `PointFloat`, and `Rect`.

## Utilities Function Panels

Table 21-1 lists the Utilities functions in a tree structure. The functions in the Utilities class are grouped according to the types of operations they perform. The first-level heading in the tree is the name of the class. The second-level headings are names of individual function panels. Each Utilities function panel represents one function.

**Table 21-1.** Utilities Function Tree

| **Class/Panel Name** | **Function Name** |
|---|---|
| Utilities | |
| Get Kernel | `imaqGetKernel` |
| Make Point | `imaqMakePoint` |
| Make Point Float | `imaqMakePointFloat` |
| Make Rect | `imaqMakeRect` |

# imaqGetKernel

## Usage

```
const float* = imaqGetKernel(KernelFamily family, int size, int number)
```

## Purpose

Returns a pointer to a predefined convolution matrix. You can use the returned pointer in conjunction with `imaqConvolve()`. You cannot dispose of or alter the returned pointer because it is a reference to static memory. If you need to alter the kernel, copy the data from the supplied kernel to the memory space you have allocated yourself. For information about the predefined kernels, see Appendix B, *Kernels*.

## Parameters

| Name | Type | Description |
|---|---|---|
| **family** | KernelFamily | The family of the kernel matrix. The following options are valid:<br>`IMAQ_GRADIENT_FAMILY`<br>`IMAQ_LAPLACIAN_FAMILY`<br>`IMAQ_SMOOTHING_FAMILY`<br>`IMAQ_GAUSSIAN_FAMILY` |
| **size** | int | The horizontal and vertical matrix size. Valid values are 3, 5, and 7, corresponding to the convolution matrix sizes of 3 × 3, 5 × 5, and 7 × 7. |
| **number** | int | References the particular desired matrix among the predefined matrices that are available for each family and size. |

## Return Value

**const float***—On success, this function returns a pointer to the requested matrix. This pointer points to constant data in memory that you should not alter. On failure, this function returns NULL. To get extended error information, call `imaqGetLastError()`. You do not need to call `imaqDispose()` on the pointer.

# imaqMakePoint

## Usage

```
Point = imaqMakePoint(int xCoordinate, int yCoordinate)
```

## Purpose

Returns a `Point` structure with the values you specify. The `Point` structure defines the location of a point. You can embed a call to `imaqMakePoint()` in calls to other IMAQ functions that require `Point` structures as input parameters, thereby eliminating the need to declare a `Point` variable.

**Note** LabWindows/CVI users: This function duplicates the functionality of the LabWindows/CVI function `MakePoint()`.

## Parameters

| Name | Type | Description |
|---|---|---|
| **xCoordinate** | int | Horizontal location of the point. |
| **yCoordinate** | int | Vertical location of the point. |

## Return Value

**Point**—This function returns a `Point` structure containing the coordinate values you specify.

# imaqMakePointFloat

## Usage

```
PointFloat = imaqMakePointFloat(float xCoordinate, float yCoordinate)
```

## Purpose

Returns a `PointFloat` structure with the values you specify. The `PointFloat` structure defines the location of a point. You can embed a call to `imaqMakePointFloat()` in calls to other IMAQ functions that require `PointFloat` structures as input parameters, thereby eliminating the need to declare a `PointFloat` variable.

## Parameters

| Name | Type | Description |
|---|---|---|
| **xCoordinate** | float | Horizontal location of the point. |
| **yCoordinate** | float | Vertical location of the point. |

## Return Value

**PointFloat**—This function returns a `PointFloat` structure containing the coordinate values you specify.

# imaqMakeRect

## Usage

```
Rect = imaqMakeRect(int top, int left, int height, int width)
```

## Purpose

Returns a `Rect` structure with the values you specify. The `Rect` structure defines the location and size of a rectangle. You can embed a call to `imaqMakeRect()` in calls to other IMAQ functions that require `Rect` structures as input parameters, thereby eliminating the need to declare a `Rect` variable.

**Note** LabWindows/CVI users: This function duplicates the functionality of the LabWindows/CVI function `MakeRect()`.

## Parameters

| Name | Type | Description |
|---|---|---|
| **top** | int | Location of the top edge of the rectangle. |
| **left** | int | Location of the left edge of the rectangle. |
| **height** | int | Height of the rectangle. |
| **width** | int | Width of the rectangle. |

## Return Value

**Rect**—This function returns a `Rect` structure containing the coordinate values you specify.

# A

# Error Codes

Table A-1 lists the IMAQ Vision for LabWindows/CVI function error codes.

**Table A-1.** IMAQ Vision for LabWindows/CVI Error Codes

| Error Code | Error Name | Description |
|---|---|---|
| 0 | ERR_SUCCESS | No error. |
| -1074396160 | ERR_SYSTEM_ERROR | System error. |
| -1074396159 | ERR_OUT_OF_MEMORY | Not enough memory for requested operation. |
| -1074396158 | ERR_MEMORY_ERROR | Memory error. |
| -1074396157 | ERR_UNREGISTERED | Unlicensed IMAQ Vision. |
| -1074396156 | ERR_NEED_FULL_VERSION | The function requires an IMAQ Vision 5.0 Advanced license. |
| -1074396155 | ERR_UNINIT | IMAQ Vision did not initialize properly. |
| -1074396154 | ERR_IMAGE_TOO_SMALL | The image is not large enough for the operation. |
| -1074396153 | ERR_BARCODE_CODABAR | The barcode is not a valid Codabar barcode. |
| -1074396152 | ERR_BARCODE_CODE39 | The barcode is not a valid Code 3 of 9 barcode. |
| -1074396151 | ERR_BARCODE_CODE93 | The barcode is not a valid Code93 barcode. |
| -1074396150 | ERR_BARCODE_CODE128 | The barcode is not a valid Code128 barcode. |
| -1074396149 | ERR_BARCODE_EAN8 | The barcode is not a valid EAN8 barcode. |
| -1074396148 | ERR_BARCODE_EAN13 | The barcode is not a valid EAN13 barcode. |
| -1074396147 | ERR_BARCODE_I25 | The barcode is not a valid Interleaved 2 of 5 barcode. |
| -1074396146 | ERR_BARCODE_MSI | The barcode is not a valid MSI barcode. |
| -1074396145 | ERR_BARCODE_UPCA | The barcode is not a valid UPCA barcode. |
| -1074396144 | ERR_BARCODE_CODE93_SHIFT | The Code93 barcode contains an invalid shift encoding. |
| -1074396143 | ERR_BARCODE_TYPE | The barcode type is invalid. |
| -1074396142 | ERR_BARCODE_INVALID | The image does not represent a valid linear barcode. |
| -1074396141 | ERR_BARCODE_CODE128_FNC | The FNC value in the Code128 barcode is not before the first data value. |

**Table A-1.** IMAQ Vision for LabWindows/CVI Error Codes (Continued)

| Error Code | Error Name | Description |
|---|---|---|
| -1074396140 | ERR_BARCODE_CODE128_SET | The starting code set in the Code128 barcode is not valid. |
| -1074396120 | ERR_NOT_IMAGE | Not an image. |
| -1074396117 | ERR_MATRIX_SIZE | Invalid matrix size in the structuring element. |
| -1074396080 | ERR_INVALID_IMAGE_TYPE | Invalid image type. |
| -1074396079 | ERR_INVALID_METAFILE_HANDLE | Invalid metafile handle. |
| -1074396077 | ERR_INCOMP_TYPE | Incompatible image type. |
| -1074396074 | ERR_INCOMP_SIZE | Incompatible image size. |
| -1074396072 | ERR_INVALID_BORDER | Invalid image border. |
| -1074396070 | ERR_INVALID_FUNCTION | Unsupported function. |
| -1074396069 | ERR_INVALID_COLOR_MODE | IMAQ Vision does not support the color mode you specified. |
| -1074396068 | ERR_INVALID_ACTION | The function does not support the requested action. |
| -1074396067 | ERR_IMAGES_NOT_DIFF | Source and destination images must be different. |
| -1074396040 | ERR_DRIVER | Cannot access NI-IMAQ driver. |
| -1074396039 | ERR_IO_ERROR | I/O error. |
| -1074396037 | ERR_TIMEOUT | Trigger timeout. |
| -1074396000 | ERR_FILE_FILE_HEADER | Invalid file header. |
| -1074395999 | ERR_FILE_FILE_TYPE | Invalid file type. |
| -1074395998 | ERR_FILE_COLOR_TABLE | Invalid color table. |
| -1074395997 | ERR_FILE_ARGERR | Invalid parameter. |
| -1074395996 | ERR_FILE_OPEN | File is already open for writing. |
| -1074395995 | ERR_FILE_NOT_FOUND | File not found. |
| -1074395994 | ERR_FILE_TOO_MANY_OPEN | Too many files open. |
| -1074395993 | ERR_FILE_IO_ERR | File I/O error. |
| -1074395992 | ERR_FILE_PERMISSION | File access denied. |
| -1074395991 | ERR_FILE_INVALID_TYPE | IMAQ Vision does not support the file type you specified. |
| -1074395990 | ERR_FILE_GET_INFO | Unable to get file information. |
| -1074395989 | ERR_FILE_READ | Unable to read data. |
| -1074395988 | ERR_FILE_WRITE | Unable to write data. |

**Table A-1.** IMAQ Vision for LabWindows/CVI Error Codes (Continued)

| Error Code | Error Name | Description |
|---|---|---|
| -1074395987 | ERR_FILE_EOF | Premature end of file. |
| -1074395986 | ERR_FILE_FORMAT | Invalid file format. |
| -1074395985 | ERR_FILE_OPERATION | Invalid file operation. |
| -1074395984 | ERR_FILE_INVALID_DATA_TYPE | IMAQ Vision does not support the file data type you specified. |
| -1074395983 | ERR_FILE_NO_SPACE | Disk full. |
| -1074395960 | ERR_INIT | Initialization error. |
| -1074395959 | ERR_CREATE_WINDOW | Unable to create window. |
| -1074395958 | ERR_WINDOW_ID | Invalid window ID. |
| -1074395957 | ERR_ARRAY_SIZE_MISMATCH | The array sizes are not compatible. |
| -1074395920 | ERR_NUMBER_CLASS | Invalid number of classes. |
| -1074395880 | ERR_PARTICLE | Invalid particle. |
| -1074395879 | ERR_BAD_MEASURE | Invalid measure number. |
| -1074395840 | ERR_BAD_INDEX | Invalid handle table index. |
| -1074395800 | ERR_PROTECTION | Protection error. |
| -1074395799 | ERR_INTERNAL | Internal error. |
| -1074395760 | ERR_BOARD_NOT_FOUND | Board not found. |
| -1074395758 | ERR_BOARD_NOT_OPEN | Board not opened. |
| -1074395757 | ERR_DLL_NOT_FOUND | DLL not found. |
| -1074395756 | ERR_DLL_FUNCTION_NOT_FOUND | DLL function not found. |
| -1074395754 | ERR_TRIG_TIMEOUT | Trigger timeout. |
| -1074395720 | ERR_BAD_ROI | Invalid ROI. |
| -1074395719 | ERR_BAD_ROI_BOX | Invalid ROI global rectangle. |
| -1074395718 | ERR_LAB_VERSION | The version of LabVIEW or BridgeVIEW you are running does not support this operation. |
| -1074395600 | ERR_INFO_NOT_FOUND | You did not provide information about the subimage within the browser. |
| -1074395401 | ERR_INVALID_MATCHFACTOR | The function does not support the matchFactor that you specified. |
| -1074395397 | ERR_COMPLEX_IMAGE_REQUIRED | Complex image required. |

**Table A-1.** IMAQ Vision for LabWindows/CVI Error Codes (Continued)

| Error Code | Error Name | Description |
|---|---|---|
| -1074395360 | ERR_COMPLEX_PLANE | Invalid complex plane. |
| -1074395349 | ERR_INVALID_SKELETONMETHOD | IMAQ Vision does not support the SkeletonMethod value you supplied. |
| -1074395348 | ERR_INVALID_3DPLANE | IMAQ Vision does not support the 3DPlane value you supplied. |
| -1074395347 | ERR_INVALID_3DDIRECTION | IMAQ Vision does not support the 3DDirection value you supplied. |
| -1074395346 | ERR_INVALID_ INTERPOLATIONMETHOD_ FOR_ROTATE | imaqRotate does not support the InterpolationMethod value you supplied. |
| -1074395345 | ERR_INVALID_FLIPAXIS | IMAQ Vision does not support the FlipAxis value you supplied. |
| -1074395343 | ERR_FILE_FILENAME_NULL | You must pass a valid file name. Do not pass in NULL. |
| -1074395340 | ERR_INVALID_SIZETYPE | IMAQ Vision does not support the SizeType value you supplied. |
| -1074395332 | ERR_INVALID_COMPAREFUNCTION | IMAQ Vision does not support the ComparisonFunction value you supplied. |
| -1074395331 | ERR_INVALID_BORDERMETHOD | IMAQ Vision does not support the BorderMethod value you supplied. |
| -1074395330 | ERR_INVALID_BORDER_SIZE | The border size is not within the valid range (0-50). |
| -1074395329 | ERR_INVALID_OUTLINEMETHOD | IMAQ Vision does not support the OutlineMethod value you supplied. |
| -1074395328 | ERR_INVALID_ INTERPOLATIONMETHOD | IMAQ Vision does not support the InterpolationMethod value you supplied. |
| -1074395327 | ERR_INVALID_SCALINGMODE | IMAQ Vision does not support the ScalingMode value you supplied. |
| -1074395326 | ERR_INVALID_DRAWMODE_ FOR_LINE | imaqDrawLineOnImage does not support the DrawMode value you supplied. |
| -1074395325 | ERR_INVALID_DRAWMODE | IMAQ Vision does not support the DrawMode value you supplied. |
| -1074395324 | ERR_INVALID_SHAPEMODE | IMAQ Vision does not support the ShapeMode value you supplied. |
| -1074395323 | ERR_INVALID_FONTCOLOR | IMAQ Vision does not support the FontColor value you supplied. |
| -1074395322 | ERR_INVALID_TEXTALIGNMENT | IMAQ Vision does not support the TextAlignment value you supplied. |

**Table A-1.** IMAQ Vision for LabWindows/CVI Error Codes (Continued)

| Error Code | Error Name | Description |
|---|---|---|
| -1074395321 | ERR_INVALID_ MORPHOLOGYMETHOD | IMAQ Vision does not support the MorphologyMethod value you supplied. |
| -1074395320 | ERR_TEMPLATE_EMPTY | The template image is empty. |
| -1074395319 | ERR_INVALID_SUBPIX_TYPE | IMAQ Vision does not support the interpolation type you supplied. |
| -1074395318 | ERR_INSF_POINTS | Insufficient points in array. |
| -1074395317 | ERR_UNDEF_POINT | Invalid point defined. |
| -1074395316 | ERR_INVALID_KERNEL_CODE | Invalid kernel code. |
| -1074395310 | ERR_INVALID_PALETTE_TYPE | IMAQ Vision does not support the PaletteType value you supplied. |
| -1074395309 | ERR_INVALID_WINDOW_THREAD_ POLICY | IMAQ Vision does not support the WindowThreadPolicy value you supplied. |
| -1074395308 | ERR_INVALID_COLORCOMPLEXITY | IMAQ Vision does not support the ColorComplexity value you supplied. |
| -1074395307 | ERR_PRECISION_NOT_GTR_THAN_0 | The percision parameter must be greater than 0. |
| -1074395306 | ERR_INVALID_TOOL | IMAQ Vision does not support the Tool value you supplied. |
| -1074395305 | ERR_INVALID_REFERENCEMODE | IMAQ Vision does not support the ReferenceMode value you supplied. |
| -1074395304 | ERR_INVALID_ MATHTRANSFORMMETHOD | IMAQ Vision does not support the MathTransformMethod value you supplied. |
| -1074395303 | ERR_INVALID_NUM_OF_CLASSES | The number of classes for imaqAutoThreshold is not within the valid range (2-256). |
| -1074395302 | ERR_INVALID_THRESHOLDMETHOD | IMAQ Vision does not support the ThresholdMethod value you supplied. |
| -1074395301 | ERR_ROI_NOT_2_LINES | The ROI you passed into imaqGetMeterArc must consist of two lines. |
| -1074395300 | ERR_INVALID_METERARCMODE | IMAQ Vision does not support the MeterArcMode value you supplied. |
| -1074395299 | ERR_INVALID_COMPLEXPLANE | IMAQ Vision does not support the ComplexPlane value you supplied. |
| -1074395298 | ERR_COMPLEXPLANE_ NOT_REAL_OR_IMAGINARY | You can perform this operation on a real or an imaginary ComplexPlane only. |
| -1074395297 | ERR_INVALID_PARTICLEINFOMODE | IMAQ Vision does not support the ParticleInfoMode value you supplied. |

**Table A-1.** IMAQ Vision for LabWindows/CVI Error Codes (Continued)

| Error Code | Error Name | Description |
|---|---|---|
| -1074395296 | ERR_INVALID_BARCODETYPE | IMAQ Vision does not support the BarcodeType value you supplied. |
| -1074395295 | ERR_INVALID_ INTERPOLATIONMETHOD_ INTERPOLATEPOINTS | imaqInterpolatePoints does not support the InterpolationMethod value you supplied. |
| -1074395294 | ERR_CONTOUR_INDEX_ OUT_OF_RANGE | The contour index you supplied is larger than the number of contours in the ROI. |
| -1074395293 | ERR_CONTOURID_NOT_FOUND | The supplied ContourID did not correlate to a contour inside the ROI. |
| -1074395280 | ERR_ROI_NOT_RECT | ROI is not a rectangle. |
| -1074395279 | ERR_ROI_NOT_POLYGON | ROI is not a polygon. |
| -1074395278 | ERR_LCD_NOT_NUMERIC | LCD image is not a number. |
| -1074395277 | ERR_BARCODE_CHECKSUM | The decoded barcode information did not pass the checksum test. |
| -1074395276 | ERR_LINES_PARALLEL | You specified parallel lines for the meter ROI. |
| -1074395275 | ERR_INVALID_BROWSER_IMAGE | Invalid browser image. |
| -1074395270 | ERR_DIV_BY_ZERO | Cannot divide by zero. |
| -1074395269 | ERR_NULL_POINTER | Null pointer. |
| -1074395268 | ERR_LINEAR_COEFF | The linear equations are not independent. |
| -1074395267 | ERR_COMPLEX_ROOT | The roots of the equation are complex. |
| -1074395265 | ERR_BARCODE | The barcode does not match the type you specified. |
| -1074395263 | ERR_LCD_NO_SEGMENTS | No lit segment. |
| -1074395262 | ERR_LCD_BAD_MATCH | The LCD does not form a known digit. |
| -1074395261 | ERR_GIP_RANGE | An internal error occurred while attempting to access an invalid coordinate on an image. |
| -1074395260 | ERR_HEAP_TRASHED | An internal memory error occurred. |
| -1074395258 | ERR_BAD_FILTER_WIDTH | The filter width must be odd for the Canny operator. |
| -1074395257 | ERR_INVALID_EDGE_DIR | You supplied an invalid edge direction in the Canny operator. |
| -1074395256 | ERR_EVEN_WINDOW_SIZE | The window size must be odd for the Canny operator. |
| -1074395253 | ERR_LEARN_MODE | Invalid learn mode. |
| -1074395252 | ERR_LEARN_SETUP_DATA | Invalid learn setup data. |

**Table A-1.** IMAQ Vision for LabWindows/CVI Error Codes (Continued)

| Error Code | Error Name | Description |
|---|---|---|
| -1074395251 | ERR_MATCH_MODE | Invalid match mode. |
| -1074395250 | ERR_MATCH_SETUP_DATA | Invalid match setup data. |
| -1074395249 | ERR_ROTATION_ANGLE_ RANGE_TOO_LARGE | At least one range in the array of rotation angle ranges exceeds 360 degrees. |
| -1074395248 | ERR_TOO_MANY_ ROTATION_ANGLE_RANGES | The array of rotation angle ranges contains too many ranges. |
| -1074395247 | ERR_TEMPLATE_DESCRIPTOR | Invalid template descriptor. |
| -1074395246 | ERR_TEMPLATE_DESCRIPTOR_ IMAGE_INFO | Invalid template descriptor. |
| -1074395245 | ERR_TEMPLATE_DESCRIPTOR_ LEARN_MODE | Invalid template descriptor. |
| -1074395244 | ERR_TEMPLATE_DESCRIPTOR_ VERSION | Invalid template descriptor. |
| -1074395243 | ERR_TEMPLATE_DESCRIPTOR_ MATCH_VERSION | The template descriptor was created in a newer version of IMAQ Vision. Upgrade to a higher version of IMAQ Vision to use this template. |
| -1074395242 | ERR_TEMPLATE_DESCRIPTOR_ ROTATION | Invalid template descriptor. |
| -1074395241 | ERR_TEMPLATE_DESCRIPTOR_ ROTATION_EMPTY | The template descriptor does not contain data that imaqMatchPattern requires for rotation-invariant matching. |
| -1074395240 | ERR_TEMPLATE_DESCRIPTOR_ ROTATION_FINALSTAGE | Invalid template descriptor. |
| -1074395239 | ERR_TEMPLATE_DESCRIPTOR_SHIFT | Invalid template descriptor. |
| -1074395238 | ERR_TEMPLATE_DESCRIPTOR_ SHIFT_EMPTY | The template descriptor does not contain data that imaqMatchPattern requires for shift-invariant matching. |
| -1074395237 | ERR_TEMPLATE_DESCRIPTOR_ SHIFT_FINALSTAGE | Invalid template descriptor. |
| -1074395235 | ERR_TEMPLATE_IMAGE_CONTRAST_ TOO_LOW | The template image does not contain enough contrast. |
| -1074395234 | ERR_TEMPLATE_IMAGE_TOO_SMALL | The template image is too small. |
| -1074395233 | ERR_TEMPLATE_IMAGE_TOO_LARGE | The template image is too large. |

# B

# Kernels

Appendix B shows the available predefined convolution matrices. You can access these matrices by calling `imaqGetKernel()`.

**Table B-1.** Gradient 3 x 3

| matrix # & Content | matrix # & Content | matrix # & Content | matrix # & Content |
|---|---|---|---|
| #0 | #1 | #2 | #3 |
| -1 0 1 | -1 0 1 | 0 1 1 | 0 1 1 |
| -1 0 1 | -1 1 1 | -1 0 1 | -1 1 1 |
| -1 0 1 | -1 0 1 | -1 -1 0 | -1 -1 0 |
| #4 | #5 | #6 | #7 |
| 1 1 1 | 1 1 1 | 1 1 0 | 1 1 0 |
| 0 0 0 | 0 1 0 | 1 0 -1 | 1 1 -1 |
| -1 -1 -1 | -1 -1 -1 | 0 -1 -1 | -0 -1 -1 |
| #8 | #9 | #10 | #11 |
| 1 0 -1 | 1 0 -1 | 0 -1 -1 | 0 -1 -1 |
| 1 0 -1 | 1 1 -1 | 1 0 -1 | 1 1 -1 |
| 1 0 -1 | 1 0 -1 | 1 1 0 | 1 1 0 |
| #12 | #13 | #14 | #15 |
| -1 -1 -1 | -1 -1 -1 | -1 -1 0 | -1 -1 0 |
| 0 0 0 | 0 1 0 | -1 0 1 | -1 1 1 |
| 1 1 1 | 1 1 1 | 0 1 1 | 0 1 1 |
| #16 | #17 | #18 | #19 |
| -1 0 1 | -1 0 1 | 0 1 2 | 0 1 2 |
| -2 0 2 | -2 1 2 | -1 0 1 | -1 1 1 |
| -1 0 1 | -1 0 1 | -2 -1 0 | -2 -1 0 |

**Table B-1.** Gradient 3 x 3 (Continued)

| matrix # & Content | matrix # & Content | matrix # & Content | matrix # & Content |
|---|---|---|---|
| #20 | #21 | #22 | #23 |
| 1 2 1 | 1 2 1 | 2 1 0 | 2 1 0 |
| 0 0 0 | 0 1 0 | 1 0 -1 | 1 1 -1 |
| -1 -2 -1 | -1 -2 -1 | 0 -1 -2 | 0 -1 -2 |
| #24 | #25 | #26 | #27 |
| 1 0 -1 | 1 0 -1 | 0 -1 -2 | 0 -1 -2 |
| 2 0 -2 | 2 1 -2 | 1 0 -1 | 1 1 -1 |
| 1 0 -1 | 1 0 -1 | 2 1 0 | 2 1 0 |
| #28 | #29 | #30 | #31 |
| -1 -2 -1 | -1 -2 -1 | -2 -1 0 | -2 -1 0 |
| 0 0 0 | 0 1 0 | -1 0 1 | -1 1 1 |
| 1 2 1 | 1 2 1 | 0 1 2 | 0 1 2 |

**Table B-2.** Gradient 5 x 5

| matrix # & Content | matrix # & Content | matrix # & Content | matrix # & Content |
|---|---|---|---|
| #0 | #1 | #2 | #3 |
| 0 -1 0 1 0 | 0 -1 0 1 0 | 0 0 1 1 1 | 0 0 1 1 1 |
| -1 -2 0 2 1 | -1 -2 0 2 1 | 0 0 2 2 1 | 0 0 2 2 1 |
| -1 -2 0 2 1 | -1 -2 1 2 1 | -1 -2 0 2 1 | -1 -2 1 2 1 |
| -1 -2 0 2 1 | -1 -2 0 2 1 | -1 -2 -2 0 0 | -1 -2 -2 0 0 |
| 0 -1 0 1 0 | 0 -1 0 1 0 | -1 -1 -1 0 0 | -1 -1 -1 0 0 |
| #4 | #5 | #6 | #7 |
| 0 1 1 1 0 | 0 1 1 1 0 | 1 1 1 0 0 | 1 1 1 0 0 |
| 1 2 2 2 1 | 1 2 2 2 1 | 1 2 2 0 0 | 1 2 2 0 0 |
| 0 0 0 0 0 | 0 0 1 0 0 | 1 2 0 -2 -1 | 1 2 1 -2 -1 |
| -1 -2 -2 -2 -1 | -1 -2 -2 -2 -1 | 0 0 -2 -2 -1 | 0 0 -2 -2 -1 |
| 0 -1 -1 -1 0 | 0 -1 -1 -1 0 | 0 0 -1 -1 -1 | 0 0 -1 -1 -1 |

**Table B-2.** Gradient 5 x 5 (Continued)

| matrix # & Content | matrix # & Content | matrix # & Content | matrix # & Content |
|---|---|---|---|
| #8 | #9 | #10 | #11 |
| 0 1 0 -1 0 | 0 1 0 -1 0 | 0 0 -1 -1 -1 | 0 0 -1 -1 -1 |
| 1 2 0 -2 -1 | 1 2 0 -2 -1 | 0 0 -2 -2 -1 | 0 0 -2 -2 -1 |
| 1 2 0 -2 -1 | 1 2 1 -2 -1 | 1 2 0 -2 -1 | 1 2 1 -2 -1 |
| 1 2 0 -2 -1 | 1 2 0 -2 -1 | 1 2 2 0 0 | 1 2 2 0 0 |
| 0 1 0 -1 0 | 0 1 0 -1 0 | 1 1 1 0 0 | 1 1 1 0 0 |
| #12 | #13 | #14 | #15 |
| 0 -1 -1 -1 0 | 0 -1 -1 -1 0 | -1 -1 -1 0 0 | -1 -1 -1 0 0 |
| -1 -2 -2 -2 -1 | -1 -2 -2 -2 -1 | -1 -2 -2 0 0 | -1 -2 -2 0 0 |
| 0 0 0 0 0 | 0 0 1 0 0 | -1 -2 0 2 1 | -1 -2 1 2 1 |
| 1 2 2 2 1 | 1 2 2 2 1 | 0 0 2 2 1 | 0 0 2 2 1 |
| 0 1 1 1 0 | 0 1 1 1 0 | 0 0 1 1 1 | 0 0 1 1 1 |

**Table B-3.** Gradient 7 x 7

| matrix # & Content | matrix # & Content | matrix # & Content | matrix # & Content |
|---|---|---|---|
| #0 | #1 | #2 | #3 |
| 0 -1 -1 0 1 1 0 | 0 -1 -1 0 1 1 0 | 0 1 1 1 1 1 0 | 0 1 1 1 1 1 0 |
| -1 -2 -2 0 2 2 1 | -1 -2 -2 0 2 2 1 | 1 2 2 2 2 2 1 | 1 2 2 2 2 2 1 |
| -1 -2 -3 0 3 2 1 | -1 -2 -3 0 3 2 1 | 1 2 3 3 3 2 1 | 1 2 3 3 3 2 1 |
| -1 -2 -3 0 3 2 1 | -1 -2 -3 1 3 2 1 | 0 0 0 0 0 0 0 | 0 0 0 1 0 0 0 |
| -1 -2 -3 0 3 2 1 | -1 -2 -3 0 3 2 1 | -1 -2 -3 -3 -3 -2 -1 | -1 -2 -3 -3 -3 -2 -1 |
| -1 -2 -2 0 2 2 1 | -1 -2 -2 0 2 2 1 | -1 -2 -2 -2 -2 -2 -1 | -1 -2 -2 -2 -2 -2 -1 |
| 0 -1 -1 0 1 1 0 | 0 -1 -1 0 1 1 0 | 0 -1 -1 -1 -1 -1 0 | 0 -1 -1 -1 -1 -1 0 |
| #4 | #5 | #6 | #7 |
| 0 1 1 0 -1 -1 0 | 0 1 1 0 -1 -1 0 | 0 -1 -1 -1 -1 -1 0 | 0 -1 -1 -1 -1 -1 0 |
| 1 2 2 0 -2 -2 -1 | 1 2 2 0 -2 -2 -1 | -1 -2 -2 -2 -2 -2 -1 | -1 -2 -2 -2 -2 -2 -1 |
| 1 2 3 0 -3 -2 -1 | 1 2 3 0 -3 -2 -1 | -1 -2 -3 -3 -3 -2 -1 | -1 -2 -3 -3 -3 -2 -1 |
| 1 2 3 0 -3 -2 -1 | 1 2 3 1 -3 -2 -1 | 0 0 0 0 0 0 0 | 0 0 0 1 0 0 0 |
| 1 2 3 0 -3 -2 -1 | 1 2 3 0 -3 -2 -1 | 1 2 3 3 3 2 1 | 1 2 3 3 3 2 1 |
| 1 2 2 0 -2 -2 -1 | 1 2 2 0 -2 -2 -1 | 1 2 2 2 2 2 1 | 1 2 2 2 2 2 1 |
| 0 1 1 0 -1 -1 0 | 0 1 1 0 -1 -1 0 | 0 1 1 1 1 1 0 | 0 1 1 1 1 1 0 |

**Table B-4.** Laplacian 3 x 3

| matrix # & Content | matrix # & Content | matrix # & Content | matrix # & Content |
|---|---|---|---|
| #0 | #1 | #2 | #3 |
| 0 -1 0 | 0 -1 0 | 0 -1 0 | -1 -1 -1 |
| -1 4 -1 | -1 5 -1 | -1 6 -1 | -1 8 -1 |
| 0 -1 0 | 0 -1 0 | 0 -1 0 | -1 -1 -1 |
| #4 | #5 | #6 | #7 |
| -1 -1 -1 | -1 -1 -1 | -1 -2 -1 | -1 -2 -1 |
| -1 9 -1 | -1 10 -1 | -2 12 -2 | -2 13 -2 |
| -1 -1 -1 | -1 -1 -1 | -1 -2 -1 | -1 -2 -1 |

**Table B-5.** Laplacian 5 x 5

| matrix # & Content | matrix # & Content |
|---|---|
| #0 | #1 |
| -1 -1 -1 -1 -1 | -1 -1 -1 -1 -1 |
| -1 -1 -1 -1 -1 | -1 -1 -1 -1 -1 |
| -1 -1 24 -1 -1 | -1 -1 25 -1 -1 |
| -1 -1 -1 -1 -1 | -1 -1 -1 -1 -1 |
| -1 -1 -1 -1 -1 | -1 -1 -1 -1 -1 |

**Table B-6.** Laplacian 7 x 7

| matrix # & Content | matrix # & Content |
|---|---|
| #0 | #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 48 -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 | -1 -1 -1 -1 -1 -1 -1 |
| -1 -1 -1 -1 -1 -1 -1 | -1 -1 -1 -1 -1 -1 -1 |

**Table B-7.** Smoothing 3 x 3

| matrix # & Content | matrix # & Content | matrix # & Content | matrix # & Content |
|---|---|---|---|
| #0 | #1 | #2 | #3 |
| 0 1 0 | 0 1 0 | 0 2 0 | 0 4 0 |
| 1 0 1 | 1 1 1 | 2 1 2 | 4 1 4 |
| 0 1 0 | 0 1 0 | 0 2 0 | 0 4 0 |
| #4 | #5 | #6 | #7 |
| 1 1 1 | 1 1 1 | 2 2 2 | 4 4 4 |
| 1 0 1 | 1 1 1 | 2 1 2 | 4 1 4 |
| 1 1 1 | 1 1 1 | 2 2 2 | 4 4 4 |

**Table B-8.** Smoothing 5 x 5

| matrix # & Content | matrix # & Content |
|---|---|
| #0 | #1 |
| 1 1 1 1 1 | 1 1 1 1 1 |
| 1 1 1 1 1 | 1 1 1 1 1 |
| 1 1 0 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 |

**Table B-9.** Smoothing 7 x 7

| matrix # & Content | matrix # & Content |
|---|---|
| #0 | #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 0 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 1 1 1 |

**Table B-10.** Gaussian 3 x 3

| matrix # & Content | matrix # & Content | matrix # & Content |
|---|---|---|
| #0 | #1 | #2 |
| 0 1 0 | 0 1 0 | 1 1 1 |
| 1 2 1 | 1 4 1 | 1 2 1 |
| 0 1 0 | 0 1 0 | 1 1 1 |
| #3 | #4 | #5 |
| 1 1 1 | 1 2 1 | 1 4 1 |
| 1 4 1 | 2 4 2 | 4 16 4 |
| 1 1 1 | 1 2 1 | 1 4 1 |

**Table B-11.** Gaussian 5 x 5

| matrix # & Content |
|---|
| #0 |
| 1 2 4 2 1 |
| 2 4 8 4 2 |
| 4 8 16 8 4 |
| 2 4 8 4 2 |
| 1 2 4 2 1 |

**Table B-12.** Gaussian 7 x 7

| matrix # & Content |
|---|
| #0 |
| 1 1 2 2 2 1 1 |
| 1 2 2 4 2 2 1 |
| 2 2 4 8 4 2 2 |
| 2 4 8 16 8 4 2 |
| 2 2 4 8 4 2 2 |
| 1 2 2 4 2 2 1 |
| 1 1 2 2 2 1 1 |

# C

# Technical Support Resources

This appendix describes the comprehensive resources available to you in the Technical Support section of the National Instruments Web site and provides technical support telephone numbers for you to use if you have trouble connecting to our Web site or if you do not have internet access.

## NI Web Support

To provide you with immediate answers and solutions 24 hours a day, 365 days a year, National Instruments maintains extensive online technical support resources. They are available to you at no cost, are updated daily, and can be found in the Technical Support section of our Web site at `www.ni.com/support`

### Online Problem-Solving and Diagnostic Resources

- **KnowledgeBase**—A searchable database containing thousands of frequently asked questions (FAQs) and their corresponding answers or solutions, including special sections devoted to our newest products. The database is updated daily in response to new customer experiences and feedback.
- **Troubleshooting Wizards**—Step-by-step guides lead you through common problems and answer questions about our entire product line. Wizards include screen shots that illustrate the steps being described and provide detailed information ranging from simple getting started instructions to advanced topics.
- **Product Manuals**—A comprehensive, searchable library of the latest editions of National Instruments hardware and software product manuals.
- **Hardware Reference Database**—A searchable database containing brief hardware descriptions, mechanical drawings, and helpful images of jumper settings and connector pinouts.
- **Application Notes**—A library with more than 100 short papers addressing specific topics such as creating and calling DLLs, developing your own instrument driver software, and porting applications between platforms and operating systems.

## Software-Related Resources

- **Instrument Driver Network**—A library with hundreds of instrument drivers for control of standalone instruments via GPIB, VXI, or serial interfaces. You also can submit a request for a particular instrument driver if it does not already appear in the library.
- **Example Programs Database**—A database with numerous, non-shipping example programs for National Instruments programming environments. You can use them to complement the example programs that are already included with National Instruments products.
- **Software Library**—A library with updates and patches to application software, links to the latest versions of driver software for National Instruments hardware products, and utility routines.

# Worldwide Support

National Instruments has offices located around the globe. Many branch offices maintain a Web site to provide information on local services. You can access these Web sites from `www.ni.com/worldwide`

If you have trouble connecting to our Web site, please contact your local National Instruments office or the source from which you purchased your National Instruments product(s) to obtain support.

For telephone support in the United States, dial 512 795 8248. For telephone support outside the United States, contact your local branch office:

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Brazil 011 284 5011, Canada (Calgary) 403 274 9391,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521,
China 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11,
France 01 48 14 24 24, Germany 089 741 31 30, Greece 30 1 42 96 427,
Hong Kong 2645 3186, India 91805275406, Israel 03 6120092,
Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456,
Mexico (D.F.) 5 280 7625, Mexico (Monterrey) 8 357 7695,
Netherlands 0348 433466, New Zealand 09 914 0488,
Norway 32 27 73 00, Poland 0 22 528 94 06, Portugal 351 1 726 9011,
Singapore 2265886, Spain 91 640 0085, Sweden 08 587 895 00,
Switzerland 056 200 51 51, Taiwan 02 2528 7227,
United Kingdom 01635 523545

# Glossary

| Prefix | Meaning | Value |
|---|---|---|
| p- | pico- | $10^{-12}$ |
| n- | nano- | $10^{-9}$ |
| µ- | micro- | $10^{-6}$ |
| m- | milli- | $10^{-3}$ |
| k- | kilo- | $10^{3}$ |
| M- | mega- | $10^{6}$ |
| G- | giga- | $10^{9}$ |
| t- | tera- | $10^{12}$ |

## Numbers/Symbols

1D — One-dimensional.

2D — Two-dimensional.

3D — Three-dimensional.

3D view — Displays the light intensity of an image in a three-dimensional coordinate system, where the spatial coordinates of the image form two dimensions and the light intensity forms the third dimension.

## A

address — Value that identifies a specific location (or series of locations) in memory.

AIPD — National Instruments internal image file format used for saving complex images and calibration information associated with an image (extension APD).

| | |
|---|---|
| aliasing | The effect created by sampling a signal that contains frequency components higher than one-half the sampling frequency. Aliasing causes high frequency information to appear as low frequency information, which can make small repetitive features look large and, in horizontally sampled images, can produce jagged lines. |
| alignment | The process by which a machine vision application determines the location, orientation, and scale of a part being inspected. |
| alpha channel | Channel used to code extra information, such as gamma correction, about a color image. The alpha channel is stored as the first byte in the four-byte representation of an RGB pixel. |
| ANSI | American National Standards Institute. |
| API | Application programming interface. |
| area | A rectangular portion of an acquisition window or frame that is controlled and defined by software. |
| area threshold | Detects objects based on their size, which can fall within a user-specified range. |
| arithmetic operators | The image operations multiply, divide, add, subtract, and remainder. |
| array | Ordered, indexed set of data elements of the same type. |
| aspect ratio | The ratio of a picture or image's width to its height. |
| asynchronous | Property of a function or operation that begins an operation and returns control to the program before the completion or termination of the operation. |
| auto-median function | A function that uses dual combinations of opening and closing operations to smooth the boundaries of objects. |

## B

| | |
|---|---|
| b | Bit. One binary digit, either 0 or 1. |
| B | Byte. Eight related bits of data, an eight-bit binary number. Also denotes the amount of memory required to store one byte of data. |

| | |
|---|---|
| barycenter | The grayscale value representing the centroid of the range of an image's grayscale values in the image histogram. |
| binary image | An image in which the objects usually have a pixel intensity of 1 (or 255) and the background has a pixel intensity of 0. |
| binary morphology | Functions that perform morphological operations on a binary image. |
| binary threshold | Separation of an image into objects of interest (assigned a pixel value of 1) and background (assigned pixel values of 0) based on the intensities of the image pixels. |
| bit depth | The number of bits ($n$) used to encode the value of a pixel. For a given $n$, a pixel can take $2^n$ different values. For example, if $n$ equals 8-bits, a pixel can take 256 different values ranging from 0 to 255. If $n$ equals 16 bits, a pixel can take 65,536 different values ranging from 0 to 65,535 or –32,768 to 32,767. |
| blob | Binary large object. A connected region or grouping of pixels in an image in which all pixels have the same intensity level. |
| blob analysis | A series of processing operations and analysis functions that produce some information about the blobs in an image. |
| blurring | Reduces the amount of detail in an image. Blurring commonly occurs because the camera is out of focus. You can blur an image intentionally by applying a lowpass frequency filter. |
| BMP | Bitmap. Image file format commonly used for 8-bit and color images (extension BMP). |
| border function | Removes objects (or particles) in a binary image that touch the image border. |
| brightness | (1) A constant added to the red, green, and blue components of a color pixel during the color decoding process. (2) The perception by which white objects are distinguished from gray and light objects from dark objects. |
| buffer | Temporary storage for acquired data. |

# C

| | |
|---|---|
| C | Celsius. |
| caliper | (1) A function in IMAQ Vision Builder that calculates distances, angles, circular fits, and the center of mass based on positions given by edge detection, particle analysis, centroid, and search functions. (2) A measurement function that finds edge pairs along a specified path in the image. This function performs an edge extraction and then finds edge pairs based on specified criteria such as the distance between the leading and trailing edges, edge contrasts, and so forth. |
| callback function | A user-defined function that receives a message. |
| center of mass | The point on an object where all the mass of the object could be concentrated without changing the first moment of the object about any axis |
| character recognition | The ability of a machine to read human-readable text. |
| chroma | The color information in a video signal. |
| chrominance | *See* chroma. |
| circle function | Detects circular objects in a binary image. |
| closing | A dilation followed by an erosion. A closing fills small holes in objects and smooths the boundaries of objects. |
| clustering | Technique where the image is sorted within a discrete number of classes corresponding to the number of phases perceived in an image. The gray values and a barycenter are determined for each class. This process is repeated until a value is obtained that represents the center of mass for each phase or class. |
| color images | Images containing color information, usually encoded in the RGB form. |
| color space | The mathematical representation for a color. For example, color can be described in terms of red, green, and blue; hue, saturation, and luma; or hue, saturation, and intensity. |
| compiler | A software utility that converts a source program in a high-level programming language, such as Basic, C, or Pascal, into an object or compiled program in machine language. Compiled programs run 10 to 1,000 times faster than interpreted programs. *See also* interpreter. |

| | |
|---|---|
| complex image | Stores information obtained from the FFT of an image. The complex numbers that compose the FFT plane are encoded in 64-bit floating-point values: 32 bits for the real part and 32 bits for the imaginary part. |
| connectivity | Defines which of the surrounding pixels of a given pixel constitute its neighborhood. |
| connectivity-4 | Only pixels adjacent in the horizontal and vertical directions are considered neighbors. |
| connectivity-8 | All adjacent pixels are considered neighbors. |
| contrast | A constant multiplication factor applied to the luma and chroma components of a color pixel in the color decoding process. |
| convex function | Computes the convex regions of objects in a binary image. |
| convolution | *See* linear filter. |
| convolution kernel | Simple 3 × 3, 5 × 5, or 7 × 7 matrices (or templates) used to represent the filter in the filtering process. The contents of these kernels are a discrete two-dimensional representation of the impulse response of the filter that they represent. |
| cross correlation | The most common way to perform pattern matching. |

## D

| | |
|---|---|
| Danielsson function | Similar to the distance functions, but with more accurate results. |
| definition | The number of values a pixel can take on, which is the number of colors or shades that you can see in the image. |
| dendrite | Branches of the skeleton of an object. |
| densitometry | Determination of optical or photographic density. |
| density function | For each gray level in a linear histogram, the function gives the number of pixels in the image that have the same gray level. |
| differentiation filter | Extracts the contours (edge detection) in gray level. |
| digital image | An image $f(x, y)$ that has been converted into a discrete number of pixels. Both spatial coordinates and brightness are specified. |

| | |
|---|---|
| dilation | Increases the size of an object along its boundary and removes tiny holes in the object. |
| distance calibration | Determination of the physical dimensions of a pixel by defining the physical dimensions of a line in the image. |
| distance function | Assigns to each pixel in an object a gray-level value equal to its shortest Euclidean distance from the border of the object. |
| DLL | Dynamic link library. A software module in Microsoft Windows containing executable code and data that can be called or used by Windows applications or other DLLs; functions and data in a DLL are loaded and linked at run time when they are referenced by a Windows application or other DLLs. |

## E

| | |
|---|---|
| edge | Defined by a sharp change (transition) in the pixel intensities in an image or along an array of pixels. |
| edge contrast | The difference between the average pixel intensity before and the average pixel intensity after the edge. |
| edge detection | Any of several techniques to identify the edges of objects in an image. |
| edge hysteresis | The difference in threshold level between a rising and a falling edge. |
| edge steepness | The number of pixels that corresponds to the slope or transition area of an edge. |
| energy center | The center of mass of a grayscale image. *See* center of mass. |
| entropy | A measure of the randomness in an image. An image with high entropy contains more pixel value variation than an image with low entropy. |
| equalize function | *See* histogram equalization. |
| erosion | Reduces the size of an object along its boundary and eliminates isolated points in the image. |
| Euclidean distance | The shortest distance between two points in a Cartesian system. |

| | |
|---|---|
| event | Object-generated response to some action or change in state, such as a mouse click or *x* number of points being acquired. The event calls an event handler (callback function), which processes the event. Events are defined as part of an ActiveX control object. |
| event handler | *See* callback function *and* event. |
| exception | Error message generated by a control and sent directly to the application or programming environment containing the control. |
| exponential and gamma corrections | Expand the high gray-level information in an image while suppressing low gray-level information. |
| exponential function | Decreases brightness and increases contrast in bright regions of an image, and decreases contrast in dark regions of an image. |

## F

| | |
|---|---|
| falling edge | The digital signal transition from the low state to the high state. |
| FFT | Fast Fourier Transform. A method used to compute the Fourier transform of an image. |
| fiducial | A reference pattern on a part that helps a machine vision application find the part's location and orientation in an image. |
| field | For an interlaced video signal, a field is half the number of horizontal lines needed to represent a frame of video. The first field of a frame contains all the odd-numbered lines, the second field contains all of the even-numbered lines. |
| Fourier spectrum | The magnitude information of the Fourier transform of an image. |
| Fourier transform | Transforms an image from the spatial domain to the frequency domain. |
| frame | A complete image. In interlaced formats, a frame is composed of two fields. |
| frequency filters | Counterparts of spatial filters in the frequency domain. For images, frequency information is in the form of spatial frequency. |
| ft | Feet. |
| function | A set of software instructions executed by a single line of code that may have input and/or output parameters and returns a value when executed. |

# G

| | |
|---|---|
| gamma | The nonlinear change in the difference between the video signal's brightness level and the voltage level needed to produce that brightness. |
| gauging | Measurement of an object or distances between objects. |
| Gaussian filter | A filter similar to the smoothing filter, but using a Gaussian kernel in the filter operation. The blurring in a Gaussian filter is more gentle than a smoothing filter. |
| grab | Acquisition technique that acquires and displays a continuous sequence of images using an IMAQ device. |
| gradient convolution filter | *See* gradient filter. |
| gradient filter | Extracts the contours (edge detection) in gray-level values. Gradient filters include the Prewitt and Sobel filters. |
| gray level | The brightness of a point (pixel) in an image. |
| gray-level dilation | Increases the brightness of pixels in an image that are surrounded by other pixels with a higher intensity. |
| gray-level erosion | Reduces the brightness of pixels in an image that are surrounded by other pixels with a lower intensity. |
| gray-level images | Images with monochrome information. |
| gray-level morphology | Functions that perform morphological operations on a gray-level image. |
| grayscale | The range of shades of black in an image. |
| GUI | Graphical user interface. An intuitive, easy-to-use means of communicating information to and from a computer program by means of graphical screen displays. GUIs can resemble the front panels of instruments or other objects associated with a computer program. |

# H

| | |
|---|---|
| h | Hour. |
| hardware | The physical components of a computer system, such as the circuit boards, plug-in boards, chassis, enclosures, peripherals, and cables. |
| highpass attenuation | Inverse of lowpass attenuation. |
| highpass FFT filter | Removes or attenuates low frequencies present in the FFT domain of an image. |
| highpass filter | Emphasizes the intensity variations in an image, detects edges (or object boundaries), and enhances fine details in an image. |
| highpass frequency filter | Attenuates or removes (truncates) low frequencies present in the frequency domain of the image. A highpass frequency filter suppresses information related to slow variations of light intensities in the spatial image. |
| highpass truncation | Inverse of lowpass truncations. |
| histogram | Indicates the quantitative distribution of the pixels of an image per gray-level value. |
| histogram equalization | Transforms the gray-level values of the pixels of an image to occupy the entire range (0 to 255 in an 8-bit image) of the histogram, increasing the contrast of the image. |
| histogram inversion | Finds the photometric negative of an image. The histogram of a reversed image is equal to the original histogram flipped horizontally around the center of the histogram. |
| histograph | In LabVIEW, a histogram that can be wired directly into a graph. |
| hit-miss function | Locates objects in the image similar to the pattern defined in the structuring element. |
| hole filling function | Fills all holes in objects that are present in a binary image. |
| HSI | Color encoding scheme in Hue, Saturation, and Intensity. |
| HSL | Color encoding scheme using Hue, Saturation, and Luma information where each image in the pixel is encoded using 32 bits: 8 bits for hue, 8 bits for saturation, 8 bits for luma, and 8 unused bits. |

| | |
|---|---|
| HSV | Color encoding scheme in Hue, Saturation, and Value. |
| hue | Represents the dominant color of a pixel. The hue function is a continuous function that covers all the possible colors generated using the R, G, and B primaries. *See also* RGB. |
| hue offset angle | The value added to all hue values so that the discontinuity occurs outside the values of interest during analysis. |
| Hz | Hertz. Frequency in units of 1/second. |

## I

| | |
|---|---|
| I/O | Input/output. The transfer of data to/from a computer system involving communications channels, operator interface devices, and/or data acquisition and control interfaces. |
| image | A two-dimensional light intensity function $f(x, y)$ where $x$ and $y$ denote spatial coordinates and the value $f$ at any point $(x, y)$ is proportional to the brightness at that point. |
| image buffer | Memory location used to store images. |
| image definition | *See* pixel depth. |
| image enhancement | The process of improving the quality of an image that you acquire from a sensor in terms of signal-to-noise ratio, image contrast, edge definition, and so on. |
| image file | A file containing image information and data. |
| image format | Defines how an image is stored in a file. Usually composed of a header followed by the pixel data. |
| image palette | The gradation of colors used to display an image on screen, usually defined by a color look-up table. |
| image processing | Encompasses various processes and analysis functions that you can apply to an image. |
| image source | Original input image. |
| image understanding | A technique that interprets the content of the image at a symbolic level rather than a pixel level. |

| | |
|---|---|
| image visualization | The presentation (display) of an image (image data) to the user. |
| imaging | Any process of acquiring and displaying images and analyzing image data. |
| in. | Inches. |
| inner gradient | Finds the inner boundary of objects. |
| inspection | The process by which parts are tested for simple defects such as missing parts or cracks on part surfaces. |
| inspection function | Analyzes groups of pixels within an image and returns information about the size, shape, position, and pixel connectivity. Typical applications include quality of parts, analyzing defects, locating objects, and sorting objects. |
| intensity | The sum of the Red, Green, and Blue primary colors divided by three. (Red+Green+Blue)/3 |
| intensity calibration | Assigning user-defined quantities such as optical densities or concentrations to the gray-level values in an image. |
| intensity profile | The gray-level distribution of the pixels along an ROI in an image. |
| intensity range | Defines the range of gray-level values in an object of an image. |
| intensity threshold | Characterizes an object based on the range of gray-level values in the object. If the intensity range of the object falls within the user-specified range, it is considered an object. Otherwise it is considered part of the background. |
| interpolation | The technique used to find values in between known values when resampling an image or array of pixels. |
| interpreter | A software utility that executes source code from a high-level language such as Basic, C or Pascal, by reading one line at a time and executing the specified operation. *See also* compiler. |

## J

| | |
|---|---|
| JPEG | Joint Photographic Experts Group. Image file format for storing 8-bit and color images with lossy compression (extension JPG). |

## K

| | |
|---|---|
| k | Kilo. The standard metric prefix for 1,000, or $10^3$, used with units of measure such as volts, hertz, and meters. |
| K | Kilo. The prefix for 1,024, or $2^{10}$, used with B in quantifying data or computer memory. |
| kbytes/s | A unit for data transfer that means 1,000 or $10^3$ bytes/s. |
| kernel | Structure that represents a pixel and its relationship to its neighbors. The relationship is specified by weighted coefficients of each neighbor. |
| Kword | 1,024 words of memory. |

## L

| | |
|---|---|
| L-skeleton function | Uses an L-shaped structuring element in the skeleton function. |
| labeling | The process by which each object in a binary image is assigned a unique value. This process is useful for identifying the number of objects in the image and giving each object a unique identity. |
| LabVIEW | Laboratory Virtual Instrument Engineering Workbench. Program development environment application based on the programming language G used commonly for test and measurement applications. |
| Laplacian filter | Extracts the contours of objects in the image by highlighting the variation of light intensity surrounding a pixel. |
| library | A file containing compiled object modules, each comprised of one of more functions, that can be linked to other object modules that make use of these functions. |
| line count | The total number of horizontal lines in the picture. |
| line gauge | Measures the distance between selected edges with high-precision subpixel accuracy along a line in an image. For example, this function can be used to measure distances between points and edges. This function also can step and repeat its measurements across the image. |
| line profile | Represents the gray-level distribution along a line of pixels in an image. |

| | |
|---|---|
| linear filter | A special algorithm that calculates the value of a pixel based on its own pixel value as well as the pixel values of its neighbors. The sum of this calculation is divided by the sum of the elements in the matrix to obtain a new pixel value. |
| logarithmic and inverse gamma corrections | Expand low gray-level information in an image while compressing information from the high gray-level ranges. |
| logarithmic function | Increases the brightness and contrast in dark regions of an image and decreases the contrast in bright regions of the image. |
| logic operators | The image operations AND, NAND, OR, XOR, NOR, difference, mask, mean, max, and min. |
| lossless compression | Compression in which the decompressed image is identical to the original image. |
| lossy compression | Compression in which the decompressed image is visually similar but not identical to the original image. |
| lowpass attenuation | Applies a linear attenuation to the frequencies in an image, with no attenuation at the lowest frequency and full attenuation at the highest frequency. |
| lowpass FFT filter | Removes or attenuates high frequencies present in the FFT domain of an image. |
| lowpass filter | Attenuates intensity variations in an image. You can use these filters to smooth an image by eliminating fine details and blurring edges. |
| lowpass frequency filter | Attenuates high frequencies present in the frequency domain of the image. A lowpass frequency filter suppresses information related to fast variations of light intensities in the spatial image. |
| lowpass truncation | Removes all frequency information above a certain frequency. |
| LSB | Least significant bit. |
| luma | The brightness information in the video picture. The luma signal amplitude varies in proportion to the brightness of the video signal and corresponds exactly to the monochrome picture. |
| luminance | *See* luma. |

| | |
|---|---|
| LUT | Look-up table. Table containing values used to transform the gray-level values of an image. For each gray-level value in the image, the corresponding new value is obtained from the look-up table. |

## M

| | |
|---|---|
| m | Meters. |
| M | (1) Mega, the standard metric prefix for 1 million or $10^6$, when used with units of measure such as volts and hertz (2) Mega, the prefix for 1,048,576, or $2^{20}$, when used with B to quantify data or computer memory. |
| M-skeleton function | Uses an M-shaped structuring element in the skeleton function. |
| machine vision | An automated application that performs a set of visual inspection tasks. |
| mask | Isolates parts of an image for further processing. |
| mask FFT filter | Removes frequencies contained in a mask (range) specified by the user. |
| mask image | An image containing a value of 1 and values of 0. Pixels in the source image with a corresponding mask image value of 1 are processed, while the others are left unchanged. |
| match score | A number ranging from 0 to 1000 that indicates how closely an acquired image matches the template image. A match score of 1000 indicates a perfect match. A match score of 0 indicates no match. |
| MB | Megabyte of memory. |
| Mbytes/s | A unit for data transfer that means 1 million or $10^6$ bytes/s. |
| median filter | A lowpass filter that assigns to each pixel the median value of its neighbors. This filter effectively removes isolated pixels without blurring the contours of objects. |
| memory buffer | *See* buffer. |
| memory window | Continuous blocks of memory that can be accessed quickly by changing addresses on the local processor. |
| method | Function that performs a specific action on or with an object. The operation of the method often depends on the values of the object properties. |

| | |
|---|---|
| mile | An Imperial unit of length equal to 5,280 feet or 1,609.344 meters. Also known as a *statute mile* to discern from a nautical mile. *See also* nautical mile. |
| MMX | Multimedia Extensions. Intel chip-based technology that allows parallel operations on integers, which results in accelerated processing of 8-bit images. |
| morphological transformations | Extract and alter the structure of objects in an image. You can use these transformations for expanding (dilating) or reducing (eroding) objects, filling holes, closing inclusions, or smoothing borders. They are used primarily to delineate objects and prepare them for quantitative inspection analysis. |
| MSB | Most significant bit. |

## N

| | |
|---|---|
| nautical mile | International unit of length used for sea and air navigation equal to 6,076.115 feet or 1,852 meters. *See also* mile. |
| neighbor | A pixel whose value affects the values of nearby pixels when an image is processed. The neighbors of a pixel are usually defined by a kernel. |
| neighborhood operations | Operations on a point in an image that take into consideration the values of the pixels neighboring that point. |
| NI-IMAQ | Driver software for National Instruments IMAQ hardware. |
| nonlinear filter | Replaces each pixel value with a nonlinear function of its surrounding pixels. |
| nonlinear gradient filter | A highpass edge-extraction filter that favors vertical edges. |
| nonlinear Prewitt filter | A highpass, edge-extraction filter based on two-dimensional gradient information. |
| nonlinear Sobel filter | A highpass, edge-extraction filter based on two-dimensional gradient information. The filter has a smoothing effect that reduces noise enhancements caused by gradient operators. |
| *N*th order filter | Filters an image using a nonlinear filter. This filter orders (or classifies) the pixel values surrounding the pixel being processed. The pixel being processed is set to the *N*th pixel value, where *N* is the order of the filter. |

| | |
|---|---|
| number of planes (in an image) | The number of arrays of pixels that compose the image. A gray-level or pseudo-color image is composed of one plane, while an RGB image is composed of three planes (one for the red component, one for the blue, and one for the green). |

## O

| | |
|---|---|
| opening | An erosion followed by a dilation. An opening removes small objects and smoothes boundaries of objects in the image. |
| operating system | Base-level software that controls a computer, runs programs, interacts with users, and communicates with installed hardware or peripheral devices. |
| operators | Allow masking, combination, and comparison of images. You can use arithmetic and logic operators in IMAQ Vision. |
| optical character verification | A machine vision application that inspects the quality of printed characters. |
| optical representation | Contains the low-frequency information at the center and the high-frequency information at the corners of an FFT-transformed image. |
| outer gradient | Finds the outer boundary of objects. |

## P

| | |
|---|---|
| palette | The gradation of colors used to display an image on screen, usually defined by a color look-up table. |
| pattern matching | The technique used to locate quickly a grayscale template within a grayscale image |
| picture element | An element of a digital image. Also called *pixel*. |
| pixel | Picture element. The smallest division that makes up the video scan line. For display on a computer monitor, a pixel's optimum dimension is square (aspect ratio of 1:1, or the width equal to the height). |
| pixel depth | The number of bits used to represent the gray level of a pixel. |
| PNG | Portable Network Graphic. Image file format for storing 8-bit, 16-bit, and color images with lossless compression (extension PNG). |

| | |
|---|---|
| power 1/Y function | Similar to a logarithmic function but with a weaker effect. |
| power Y function | *See* exponential function. |
| Prewitt filter | Extracts the contours (edge detection) in gray-level values using a 3 × 3 filter kernel. |
| probability function | Defines the probability that a pixel in an image has a certain gray-level value. |
| proper-closing | A finite combination of successive closing and opening operations that you can use to fill small holes and smooth the boundaries of objects. |
| proper-opening | A finite combination of successive opening and closing operations that you can use to remove small particles and smooth the boundaries of objects. |
| property | Attribute that controls the appearance or behavior of an object. The property can be a specific value or another object with its own properties and methods. For example, a value property is the color (property) of a plot (object), while an object property is a specific Y axis (property) on a graph (object). The Y axis itself is another object with properties, such as minimum and maximum values. |
| pts | Points. |
| pyramidal matching | A technique used to increase the speed of a pattern matching algorithm by matching subsampled versions of the image and the reference pattern. |

## Q

| | |
|---|---|
| quantitative analysis | Obtaining various measurements of objects in an image. |

## R

| | |
|---|---|
| RAM | Random-access memory. |
| real time | A property of an event or system in which data is processed as it is acquired instead of being accumulated and processed at a later time. |

| | |
|---|---|
| resolution | (1) The number of rows and columns of pixels. An image composed of *m* rows and *n* columns has a resolution of $m \times n$. This image has *n* pixels along its horizontal axis and *m* pixels along its vertical axis. (2) The smallest signal increment that can be detected by a measurement system. Resolution can be expressed in bits, proportions, or a percentage of full scale. For example, a system has 12-bit resolution, one part in 4,096 resolution, and 0.0244 percent of full scale. |
| reverse function | Inverts the pixel values in an image, producing a photometric negative of the image. |
| RGB | Color encoding scheme using red, green, and blue (RGB) color information where each pixel in the color image is encoded using 32 bits: 8 bits for red, 8 bits for green, 8 bits for blue, and 8 bits for the alpha value (unused). |
| rising edge | The digital signal transition from the high state to the low state. |
| Roberts filter | Extracts the contours (edge detection) in gray level, favoring diagonal edges. |
| ROI | Region of interest. (1) An area of the image that is graphically selected from a window displaying the image. This area can be used focus further processing. (2) A hardware-programmable rectangular portion of the acquisition window. |
| ROI tools | Collection of tools from the Lab VIEW Tools palette that enable you to select a region of interest from an image. These tools let you select a point or line; polygon, rectangle, and oval regions; and freehand lines and areas. |
| rotational shift | The amount by which one image is rotated with respect to a reference image. This rotation is computed with respect to the center of the image. |
| rotation-invariant matching | A pattern matching technique in which the reference pattern can be located at any orientation in the test image as well as rotated at any degree. |

## S

| | |
|---|---|
| s | Seconds. |
| saturation | The amount of white added to a pure color. Saturation relates to the richness of a color. A saturation of zero corresponds to a pure color with no white added. Pink is a red with low saturation. |

| | |
|---|---|
| scale-invariant matching | A pattern matching technique in which the reference pattern can be any size in the test image. |
| segmentation function | Fully partitions a labeled binary image into non-overlapping segments, with each segment containing a unique object. |
| separation function | Separates objects that touch each other by narrow isthmuses. |
| shape matching | Finds objects in an image whose shape matches the shape of the object specified by a shape template. The matching process is invariant to rotation and can be set to be invariant to the scale of the objects. |
| shift-invariant matching | A pattern matching technique in which the reference pattern can be located anywhere in the test image but cannot be rotated or scaled. |
| Sigma filter | A highpass filter that outlines edges. |
| skeleton function | Applies a succession of thinning operations to an object until its width becomes one pixel. |
| skiz function | Obtains lines in an image that separate each object from the others and are equidistant from the objects that they separate. |
| smoothing filter | Blurs an image by attenuating variations of light intensity in the neighborhood of a pixel. |
| snap | Acquisition technique that acquires and displays a single image. |
| Sobel filter | Extracts the contours (edge detection) in gray-level values using a $3 \times 3$ filter kernel. |
| spatial calibration | Assigning physical dimensions to the area of a pixel in an image. |
| spatial filters | Alter the intensity of a pixel with respect to variations in intensities of its neighboring pixels. You can use these filters for edge detection, image enhancement, noise reduction, smoothing, and so forth. |
| spatial resolution | The number of pixels in an image, in terms of the number of rows and columns in the image. |
| square function | *See* exponential function. |
| square root function | *See* logarithmic function. |

| | |
|---|---|
| standard representation | Contains the low-frequency information at the corners and high-frequency information at the center of an FFT-transformed image. |
| statute mile | *See mile.* |
| structuring element | A binary mask used in most morphological operations. A structuring element is used to determine which neighboring pixels contribute in the operation. |
| sub-pixel analysis | Finds the location of the edge coordinates in terms of fractions of a pixel. |
| synchronous | Property or operation that begins an operation and returns control to the program only when the operation is complete. |
| syntax | Set of rules to which statements must conform in a particular programming language. |

## T

| | |
|---|---|
| template | Color, shape, or pattern that you are trying to match in an image using the color matching, shape matching, or pattern matching functions. A template can be a region selected from an image or it can be an entire image. |
| thickening | Alters the shape of objects by adding parts to the object that match the pattern specified in the structuring element. |
| thinning | Alters the shape of objects by eliminating parts of the object that match the pattern specified in the structuring element. |
| threshold | Separates objects from the background by assigning all pixels with intensities within a specified range to the object and the rest of the pixels to the background. In the resulting binary image, objects are represented with a pixel intensity of 255 and the background is set to 0. |
| threshold interval | Two parameters, the lower threshold gray-level value and the upper threshold gray-level value. |
| TIFF | Tagged Image File Format. Image format commonly used for encoding 8-bit, 16-bit, and color images (extension TIF). |

| | |
|---|---|
| Tools palette | Collection of tools that enable you to select regions of interest, zoom in and out, and change the image palette. |
| truth table | A table associated with a logic operator that describes the rules used for that operation. |

## U

| | |
|---|---|
| UV plane | *See* YUV. |

## V

| | |
|---|---|
| V | Volts. |
| value | The grayscale intensity of a color pixel computed as the average of the maximum and minimum red, green, and blue values of that pixel. |
| VI | Virtual Instrument. (1) A combination of hardware and/or software elements, typically used with a PC, that has the functionality of a classic stand-alone instrument (2) A LabVIEW software module (VI), which consists of a front panel user interface and a block diagram program. |

## W

| | |
|---|---|
| watershed | A technique used to segment an image into multiple regions. |

## Y

| | |
|---|---|
| YUV | A representation of a color image used for the coding of NTSC or PAL video signals. The luma information is called Y, while the chroma information is represented by two components, U and V representing the coordinates in a color plane. |

# Index

## A



## B

## C



## D

## E



## F

## G



## H



## I

## K

## L



## M



## N



## O

## P



## R

## S



## T



## U



## W