# IMAQ™

# NI-IMAQ Function Reference Manual

NATIONAL INSTRUMENTS™

November 2000 Edition
Part Number 370161A-01

.


### Worldwide Technical Support and Product Information

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:

| | |
|---|---|
| <> | Angle brackets that contain numbers separated by an ellipsis represent a range of values associated with a bit or signal name—for example, DBIO<3..0>. |
| » | The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box. |
|  | 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. |
| *`monospace italic`* | Italic text in this font denotes text that is a placeholder for a word or value that you must supply. |

# Contents

## Chapter 1
## Introduction


Status Codes ..... 1-1
Variable Data Types ..... 1-1
Primary Types ..... 1-1
Programming Language Considerations ..... 1-2
LabVIEW ..... 1-2
LabWindows/CVI ..... 1-2
Other Programming Environments ..... 1-6
Code Examples ..... 1-6


## Chapter 2
## Generic Functions


imgInterfaceOpen ..... 2-2
imgSessionOpen ..... 2-3
imgClose ..... 2-4


## Chapter 3
## High-Level Functions


Snap Functions ..... 3-1
imgSnap ..... 3-2
imgSnapArea ..... 3-3
Grab Functions ..... 3-5
imgGrabSetup ..... 3-6
imgGrab ..... 3-7
imgGrabArea ..... 3-8
Ring and Sequence Functions ..... 3-10
imgRingSetup ..... 3-11
imgSequenceSetup ..... 3-13
imgSessionStartAcquisition ..... 3-15
imgSessionStopAcquisition ..... 3-16
Signal I/O Functions ..... 3-17
imgSessionTriggerConfigure ..... 3-18
imgSessionLineTrigSource ..... 3-20
imgSessionTriggerClear ..... 3-22
imgSessionTriggerDrive ..... 3-23
imgSessionTriggerRead ..... 3-25
imgSessionWaitSignal ..... 3-27

imgSessionWaitSignalAsync ... 3-29
imgPulseCreate ... 3-31
imgPulseDispose ... 3-34
imgPulseRate ... 3-35
imgPulseStart ... 3-36
imgPulseStop ... 3-37
Miscellaneous Functions ... 3-38
imgSessionStatus ... 3-39
imgSessionSetROI ... 3-40
imgSessionGetROI ... 3-42
imgSessionGetBufferSize ... 3-43

## Chapter 4
## Low-Level Functions

Acquisition Functions ... 4-1
imgMemLock ... 4-2
imgMemUnlock ... 4-3
imgSessionAbort ... 4-4
imgSessionAcquire ... 4-5
imgSessionConfigure ... 4-6
imgSessionCopyArea ... 4-7
imgSessionCopyBuffer ... 4-9
imgSessionExamineBuffer ... 4-10
imgSessionReleaseBuffer ... 4-12
Attribute Functions ... 4-13
imgGetAttribute ... 4-14
imgGetCameraAttributeNumeric ... 4-15
imgGetCameraAttributeString ... 4-16
imgSessionGetLostFramesList ... 4-17
imgSessionSetUserLUT8bits ... 4-18
imgSessionSetUserLUT16bits ... 4-19
imgSetAttribute ... 4-20
imgSetCameraAttributeNumeric ... 4-21
imgSetCameraAttributeString ... 4-22
Buffer Management Functions ... 4-23
imgCreateBuffer ... 4-24
imgCreateBufList ... 4-26
imgDisposeBuffer ... 4-27
imgDisposeBufList ... 4-28
imgGetBufferElement ... 4-29
imgSessionClearBuffer ... 4-30
imgSetBufferElement ... 4-31
Interface Functions ... 4-33

imgInterfaceQueryNames....4-34
imgInterfaceReset....4-35
Utility Functions....4-36
imgPlot....4-37
imgPlotDC....4-39
imgSessionSaveBufferEx....4-41
imgShowError....4-42

**Appendix A**
**Attributes and Constants**

NI-IMAQ Attributes....A-1
Constants....A-8

**Appendix B**
**Status Codes**

**Appendix C**
**Calibration Information for the IMAQ PCI/PXI-1411**

**Appendix D**
**Technical Support Resources**

**Glossary**

**Index**

# 1 Introduction

The *NI-IMAQ Function Reference Manual* is for users of the NI-IMAQ software. NI-IMAQ is a powerful application programming interface (API) between your image acquisition (IMAQ) application and the National Instruments IMAQ board.

This chapter contains important information about how to apply the function descriptions in this manual to your programming language and environment.

## Status Codes

Every NI-IMAQ function is of the following form:

```
rval = Function_Name(parameter 1, parameter 2, ... parameter n)
```

Each function returns a status code (**rval**) that indicates the success or failure of the function, as discussed in Appendix B, *Status Codes*.

## Variable Data Types

The following sections describe the notation used in parameter tables and throughout the manual for variable data types.

### Primary Types

Table 1-1 shows the primary type names and their ranges.

**Table 1-1.** Primary Type Names

| Primary Type Name | Description | Range | Intrinsic Type |
|---|---|---|---|
| Int8 | 8-bit ASCII character | 0 to 127, –128 to 0 | `char` |
| uInt8 | 8-bit ASCII character | 0 to 255 | `char` |
| Int16 | 16-bit signed integer | –32,768 to 32,767 | `short` |
| uInt16 | 16-bit unsigned integer | 0 to 65,535 | `unsigned short` |

**Table 1-1.** Primary Type Names (Continued)

| Primary Type Name | Description | Range | Intrinsic Type |
|---|---|---|---|
| Int32 | 32-bit signed integer | -2,147,483,648 to 2,147,483,647 | `long` |
| uInt32 | 32-bit unsigned integer | 0 to 4,294,967,295 | `unsigned long` |

# Programming Language Considerations

Apart from the data type differences, there are a few language-dependent considerations you need to be aware of when you use the NI-IMAQ API.

**Note** Be sure to include the NI-IMAQ function prototypes by including the appropriate NI-IMAQ header file in your source code.

## LabVIEW

For information on how to use LabVIEW VIs with your IMAQ system, refer to the *NI-IMAQ VI Reference Manual*.

## LabWindows/CVI

Inside the LabWindows/CVI environment, the NI-IMAQ functions appear in **Libraries**»**NI-IMAQ**. Each function panel represents an NI-IMAQ function, which is displayed at the bottom of the panel.

Table 1-2 shows how the LabWindows/CVI function panel tree is organized, and the NI-IMAQ function name that corresponds to each function panel.

**Table 1-2.** The LabWindows/CVI Function Tree for Image Acquisition

| LabWindows/CVI Function Panel | NI-IMAQ Function |
|---|---|
| **Generic Functions** | |
| Close Object | `imgClose` |
| Interface Open | `imgInterfaceOpen` |
| Session Open | `imgSessionOpen` |

**Table 1-2.** The LabWindows/CVI Function Tree for Image Acquisition (Continued)

| LabWindows/CVI Function Panel | NI-IMAQ Function |
|---|---|
| **High-Level Snap Functions** | |
| Snap | `imgSnap` |
| Snap Area | `imgSnapArea` |
| **High-Level Grab Functions** | |
| Grab | `imgGrab` |
| Grab Area | `imgGrabArea` |
| Grab Setup | `imgGrabSetup` |
| **High-Level Ring and Sequence Functions** | |
| Ring Setup | `imgRingSetup` |
| Sequence Setup | `imgSequenceSetup` |
| Session Start Acquisition | `imgSessionStartAcquisition` |
| Session Stop Acquisition | `imgSessionStopAcquisition` |
| **High-Level Signal I/O Functions** | |
| Line Trigger Source | `imgSessionLineTrigSource` |
| Pulse Create | `imgPulseCreate` |
| Pulse Dispose | `imgPulseDispose` |
| Pulse Rate | `imgPulseRate` |
| Pulse Start | `imgPulseStart` |
| Pulse Stop | `imgPulseStop` |
| Trigger Configure | `imgSessionTriggerConfigure` |
| Trigger Clear | `imgSessionTriggerClear` |
| Trigger Drive | `imgSessionTriggerDrive` |
| Trigger Read | `imgSessionTriggerRead` |

**Table 1-2.** The LabWindows/CVI Function Tree for Image Acquisition (Continued)

| LabWindows/CVI Function Panel | NI-IMAQ Function |
|---|---|
| Wait Signal | `imgSessionWaitSignal` |
| Wait Signal Asynchronous | `imgSessionWaitSignalAsync` |
| **High-Level Miscellaneous Functions** | |
| Session Get Buffer Size | `imgSessionGetBufferSize` |
| Session Get ROI | `imgSessionGetROI` |
| Session Set ROI | `imgSessionSetROI` |
| Session Status | `imgSessionStatus` |
| **Low-Level Acquisition Functions** | |
| Lock Buffer List Memory | `imgMemLock` |
| Session Abort | `imgSessionAbort` |
| Session Acquire | `imgSessionAcquire` |
| Session Configure | `imgSessionConfigure` |
| Session Copy Area | `imgSessionCopyArea` |
| Session Copy Buffer | `imgSessionCopyBuffer` |
| Session Examine Buffer | `imgSessionExamineBuffer` |
| Session Release Buffer | `imgSessionReleaseBuffer` |
| Unlock Buffer List Memory | `imgMemUnlock` |
| **Low-Level Attribute Functions** | |
| Get Attribute | `imgGetAttribute` |
| Get Camera Attribute Numeric | `imgGetCameraAttributeNumeric` |
| Get Camera Attribute String | `imgGetCameraAttributeString` |
| Get Lost Frames List | `imgSessionGetLostFramesList` |
| Set Attribute | `imgSetAttribute` |
| Set Camera Attribute Numeric | `imgSetCameraAttributeNumeric` |

**Table 1-2.** The LabWindows/CVI Function Tree for Image Acquisition (Continued)

| LabWindows/CVI Function Panel | NI-IMAQ Function |
|---|---|
| Set Camera Attribute String | `imgSetCameraAttributeString` |
| Set User LUT 8 bits | `imgSessionSetUserLUT8bits` |
| Set User LUT 16 bits | `ImgSessionSetUserLUT16bits` |
| **Low-Level Buffer Management Functions** | |
| Create Buffer | `imgCreateBuffer` |
| Create Buffer List | `imgCreateBufList` |
| Dispose Buffer | `imgDisposeBuffer` |
| Dispose Buffer List | `imgDisposeBufList` |
| Get Buffer Element | `imgGetBufferElement` |
| Session Clear Buffer | `imgSessionClearBuffer` |
| Set Array Pointer Value | `imgSetArrayPointerValue` |
| Set Buffer Element | `imgSetBufferElement` |
| **Low-Level Interface Functions** | |
| Interface Lock | `imgInterfaceLock` |
| Interface Query Names | `imgInterfaceQueryNames` |
| Interface Reset | `imgInterfaceReset` |
| Interface Unlock | `imgInterfaceUnlock` |
| **Low-Level Utility Functions** | |
| Plot Buffer to DC | `imgPlotDC` |
| Plot Buffer to Window | `imgPlot` |
| Session Save Buffer | `imgSessionSaveBufferEx` |
| Show Error | `imgShowError` |

## Other Programming Environments

If you are using other programming languages, such as Microsoft Visual C++, with your IMAQ system, use the functions described in this manual. For additional information, refer to the *NI-IMAQ User Manual*.

## Code Examples

You can find code examples in the same directory in which you installed the NI-IMAQ driver software. You can find source code common to all environments in the `Samples` default subfolder.

# 2

# Generic Functions

This chapter contains a detailed explanation of each generic NI-IMAQ function. The functions are arranged according to the order in which you will use them.

Generic functions include `imgInterfaceOpen`, `imgSessionOpen`, and `imgClose`. These functions set up your interface and session, and close both when you are finished with your application. Use these functions in combination with both high- and low-level functions.

# imgInterfaceOpen

## Format

```
rval = imgInterfaceOpen(char* interface_name, INTERFACE_ID* pifid)
```

## Purpose

Opens by name an interface as specified in Measurement & Automation Explorer. If it is successful, this function returns an INTERFACE_ID.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **interface_name** | char* | input | null-terminated string containing the name of interface to open |
| **pifid** | INTERFACE_ID* | output | pointer to INTERFACE_ID variable |
| **rval** | Int32 | output | status |

## Parameter Discussion

**interface_name** needs a null-terminated string that is the name of the interface to open, such as `img0`, `img1`, and so on.

**pifid** passes a pointer to an INTERFACE_ID variable. If the function succeeds, the variable contains a valid INTERFACE_ID that you can use in subsequent functions.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more information, call `imgShowError`.

**Note** You can use `imgInterfaceQueryNames` to retrieve a valid list of interface names.

# imgSessionOpen

## Format

```
rval = imgSessionOpen(INTERFACE_ID ifid, SESSION_ID* psid)
```

## Purpose

Opens a session of an unknown type and returns a session ID. This function inherits all data associated with the given interface.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **ifid** | INTERFACE_ID | input | interface ID to open session |
| **psid** | SESSION_ID* | output | pointer to a session ID |
| **rval** | Int32 | output | status |

## Parameter Discussion

**ifid** is a valid INTERFACE_ID variable.

**psid** passes a pointer to an area of memory reserved for a SESSION_ID variable.
If the function succeeds, the variable contains a valid SESSION_ID that you can use in subsequent functions.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more information, call `imgShowError`.

# imgClose

## Format

```
rval = imgClose(uInt32 void_id, uInt32 freeResources)
```

## Purpose

Closes a session or interface and unlocks and releases all buffers associated with the data type.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **void_id** | uInt32 | input | session or interface ID |
| **freeResources** | uInt32 | input | cleanup flag |
| **rval** | Int32 | output | status |

## Parameter Discussion

**void_id** is a valid SESSION_ID or INTERFACE_ID variable.

**freeResources** is the cleanup flag. If **freeResources** is TRUE, the function releases all buffers and buffer lists associated with the session. If **freeResources** is FALSE, the function performs no buffer cleanup.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more information, call `imgShowError`.

**Note** Closing an interface closes all sessions attached to that interface.

# 3

# High-Level Functions

This chapter contains a detailed explanation of each high-level NI-IMAQ function. The functions are arranged according to the category of image acquisition procedure and then the order in which you will use them.

Using high-level functions, you can easily perform such functions as acquiring images in single-shot (snap) or continuous (grab) mode without advanced knowledge of the NI-IMAQ low-level function calls and image acquisition details.

## Snap Functions

Snap functions include `imgSnap` and `imgSnapArea`. Use these functions to acquire a single image after using `imgInterfaceOpen` and `imgSessionOpen` to obtain a valid SESSION_ID.

# imgSnap

## Format

```
rval = imgSnap(SESSION_ID sid, void** bufAddr)
```

## Purpose

Performs a single frame or field acquisition. This function uses the following attributes to perform an image acquisition:

| | |
|---|---|
| IMG_ATTR_ACQWINDOW_LEFT | IMG_ATTR_ROI_TOP |
| IMG_ATTR_ACQWINDOW_WIDTH | IMG_ATTR_ROI_HEIGHT |
| IMG_ATTR_ACQWINDOW_TOP | IMG_ATTR_ROI_WIDTH |
| IMG_ATTR_ACQWINDOW_HEIGHT | IMG_ATTR_ROWBYTES |
| IMG_ATTR_ROI_LEFT | IMG_ATTR_YOFF_BUFFER |

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **bufAddr** | void** | input | pointer to a pointer to the buffer address |
| **rval** | Int32 | output | no error |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**bufAddr** is a pointer to a pointer to an area of memory in which to store the image. If **bufAddr** points to a NULL pointer, this call allocates an appropriate size buffer and returns the buffer address in the location specified by **bufAddr**.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSnapArea

## Format

```
rval = imgSnapArea(SESSION_ID sid, void** bufAddr, uInt32 top, uInt32 left,
                           uInt32 height,uInt32 width, uInt32 rowPixels)
```

## Purpose

Performs an area-specific frame or field acquisition. This function does not modify any attributes. This operates the same as `imgSnap`, but can be used to acquire only a portion of the image.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **bufAddr** | void** | input | pointer to a pointer to the buffer address |
| **top** | uInt32 | input | top ordinate of the first pixel transferred |
| **left** | uInt32 | input | left ordinate of the first pixel transferred |
| **height** | uInt32 | input | height of the rectangle to transfer |
| **width** | uInt32 | input | width of the rectangle to transfer |
| **rowPixels** | uInt32 | input | used in calculating the address of the next line |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**bufAddr** is a pointer to a pointer to an area of memory in which to store the image. If **bufAddr** points to a NULL pointer, this call allocates an appropriate size buffer and returns the buffer address in the location specified by **bufAddr**.

**top** indicates the top vertical offset of the first pixel transferred.

**left** indicates the left horizontal offset of the first pixel transferred.

**height** indicates the height of the area to transfer.

**width** indicates the width of the area to transfer.

**rowPixels** indicates the exact pixel-width of the horizontal line to acquire. This parameter specifies the number of pixels to add to the line pointer for the next scan line. This value must be greater than or equal to the width parameter and a multiple of four. Passing a zero for this value causes it to be ignored.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# Grab Functions

Grab functions include `imgGrabSetup`, `imgGrab`, and `imgGrabArea`. Use the grab functions to perform a continuous acquisition into a single buffer.

To use the grab functions, first call `imgGrabSetup` to configure the session for grabbing and optionally start the acquisition process. If you do not start the acquisition using `imgGrabSetup`, you must start it by calling `imgSessionStartAcquisition` prior to calling the `imgGrab` and `imgGrabArea` functions. After the acquisition has started, you obtain an image copy by calling the `imgGrab` and `imgGrabArea`. To stop the acquisition, call `imgSessionStopAcquisition`.

# imgGrabSetup

## Format

```
rval = imgGrabSetup(SESSION_ID sid, uInt32 startNow)
```

## Purpose

Configures and optionally starts a continuous acquisition. This function uses the following attributes to perform an image acquisition:

IMG_ATTR_ACQWINDOW_LEFT
IMG_ATTR_ACQWINDOW_WIDTH
IMG_ATTR_ACQWINDOW_TOP
IMG_ATTR_ACQWINDOW_HEIGHT
IMG_ATTR_ROI_LEFT
IMG_ATTR_ROI_TOP
IMG_ATTR_ROI_HEIGHT
IMG_ATTR_ROI_WIDTH
IMG_ATTR_ROWBYTES
IMG_ATTR_YOFF_BUFFER

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **startNow** | uInt32 | input | start acquisition after setup completes |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**startNow** starts the acquisition after setup has been completed. A non-zero value specifies that the continuous acquisition should start immediately. If the value is zero, start the acquisition with `imgSessionStartAcquisition`.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgGrab

## Format

```
rval = imgGrab(SESSION_ID sid, void** bufAddr, uInt32 syncOnVB)
```

## Purpose

Acquires the most current frame into the specified buffer. Call this function only after calling `imgGrabSetup`. This function uses the following attributes:

| | |
|---|---|
| IMG_ATTR_ROI_LEFT | IMG_ATTR_ROI_WIDTH |
| IMG_ATTR_ROI_TOP | IMG_ATTR_ROWBYTES |
| IMG_ATTR_ROI_HEIGHT | IMG_ATTR_YOFF_BUFFER |

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **bufAddr** | void** | input | pointer to a pointer to the buffer address |
| **syncOnVB** | uInt32 | input | vertical blank flag |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**bufAddr** is a pointer to a pointer to an area of memory in which to store the image. If **bufAddr** points to a NULL pointer, this call allocates an appropriate size buffer and returns the buffer address in the location specified by **bufAddr**.

**syncOnVB** indicates a wait for a vertical blank. If this parameter is TRUE, the function performs the transfer according to and using the video synchronization. Use this option to avoid mixing two different time bases within the same video field. If **syncOnVB** is FALSE, the function performs the transfer without considering the video synchronization.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgGrabArea

## Format

```
rval = imgGrabArea(SESSION_ID sid, void** bufAddr, uInt32 syncOnVB,
                           uInt32 top,uInt32 left, uInt32 height,
                           uInt32 width, uInt32 rowPixels)
```

## Purpose

Performs a transfer from a continuous acquisition using the given parameters. This function does not modify any attributes. Call this function only after calling `imgGrabSetup`.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **bufAddr** | void** | input | pointer to a pointer to the buffer address |
| **syncOnVB** | uInt32 | input | vertical blank flag |
| **top** | uInt32 | input | top ordinate of the first pixel transferred |
| **left** | uInt32 | input | left ordinate of the first pixel transferred |
| **height** | uInt32 | input | height of the rectangle to transfer |
| **width** | uInt32 | input | width of the rectangle to transfer |
| **rowPixels** | uInt32 | input | used in calculating the address of the next line |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**bufAddr** is a pointer to a pointer to an area of memory in which to store the image. If **bufAddr** points to a NULL pointer, this call allocates an appropriate size buffer and returns the buffer address in the location specified by **bufAddr**.

**syncOnVB** indicates a wait for a vertical blank. If **syncOnVB** is TRUE, the function performs the transfer using the video synchronization. Use this option to avoid mixing two different time bases within the same video field. If **syncOnVB** is FALSE, the function performs the transfer without considering the video synchronization.

**top** indicates the top vertical offset of the first pixel transferred.

**left** indicates the left horizontal offset of the first pixel transferred.

**height** indicates the height of the area to transfer.

**width** indicates the width of the area to transfer.

**rowPixels** specifies the number of pixels to add to the line pointer for the next scan line. This value must be greater than or equal to the width parameter and a multiple of four. Passing a zero for this value causes the function to ignore the parameter. Use this parameter for byte alignment or if the image buffer contains a border for image processing.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# Ring and Sequence Functions

Ring and sequence functions include `imgRingSetup`, `imgSequenceSetup`, `imgSessionStartAcquisition`, and `imgSessionStopAcquisition`. Use these functions to perform a multibuffered acquisition that stops after all buffers are filled (sequence) or continually loops through the buffers (ring).

To use the ring and sequence functions, you must first call `imgRingSetup` or `imgSequenceSetup` to configure the session and optionally start the acquisition process. If you do not start the acquisition using `imgRingSetup` or `imgSequenceSetup`, you must call `imgSessionStartAcquisition` to start the acquisition.

# imgRingSetup

## Format

```
rval = imgRingSetup(SESSION_ID sid, uInt32 numberBuffer, void* bufferList[],
                    uInt32 skipCount, uInt32 startnow)
```

## Purpose

Prepares a session for acquiring continuously and looping into a buffer list.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **numberBuffer** | uInt32 | input | number of buffers used in the ring session |
| **bufferList[]** | void* | input | array of buffer pointers |
| **skipCount** | uInt32 | input | number of frames or field to skip before each acquisition |
| **startnow** | uInt32 | input | start acquisition after setup completes |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**numberBuffer** indicates the number of buffers in the buffer list.

**bufferList[]** is an array of buffer pointers. For each element in the buffer list that is initialized to NULL, **bufferList[]** allocates a buffer and returns this buffer address in the array element. If buffer[0] contains a NULL pointer, this function allocates the number of buffers required and returns the buffer addresses in **bufferList[]**.

**skipCount** indicates the number of frames or field to skip before each acquisition. This number is the same for all acquisitions.

**startNow** starts the acquisition after setup has been completed. A non-zero value specifies that the continuous acquisition should start immediately. If the value is zero, start the acquisition with `imgSessionStartAcquisition`.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSequenceSetup

## Format

```
rval = imgSequenceSetup(SESSION_ID sid, uInt32 numberBuffer,
                        void* bufferList[],uInt32 skipCount[],
                        uInt32 startnow, uInt32 async)
```

## Purpose

Prepares a session for acquiring a full sequence into the buffer list.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **numberBuffer** | uInt32 | input | number of buffers used in the ring session |
| **bufferList[]** | void* | input | array of buffer pointers |
| **skipCount[]** | uInt32 | input | array containing the number of frames or field to skip before each acquisition |
| **startNow** | uInt32 | input | start acquisition after setup completes |
| **async** | uInt32 | input | indicates either a synchronous or an asynchronous acquisition |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**numberBuffer** indicates the number of buffers in the buflist.

**bufferList[]** is an array of buffer pointers. For each element in the buffer list that is initialized to NULL, **bufferList[]** allocates a buffer and returns this buffer address in the array element.

**skipCount[]** is an array containing the number of frames or fields to skip before each acquisition.

**startNow** starts the acquisition after setup has been completed. A non-zero value specifies that the continuous acquisition should start immediately. If the value is zero, start the acquisition with `imgSessionStartAcquisition`.

**async** indicates either an asynchronous or a synchronous acquisition. If **async** is non-zero, it indicates an asynchronous acquisition. If **async** is zero, it indicates a synchronous acquisition. This parameter is valid only if the **startNow** parameter is non-zero.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionStartAcquisition

## Format

```
rval = imgSessionStartAcquisition(SESSION_ID sid)
```

## Purpose

Starts an acquisition in the session identified by **sid**. Use this function with grab, ring, and sequence functions only if startNow was set to FALSE in the setup functions, or if `imgSessionStopAcquisition` has been called.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionStopAcquisition

## Format

```
rval = imgSessionStopAcquisition(SESSION_ID sid)
```

## Purpose

Stops the acquisition in the session identified by **sid**. Use this function with grab, ring, and sequence functions.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# Signal I/O Functions

Signal I/O functions include `imgSessionTriggerConfigure`, `imgSessionLineTrigSource`, `imgSessionTriggerClear`, `imgSessionTriggerDrive`, `imgSessionTriggerRead`, `imgSessionWaitSignal`, `imgSessionWaitSignalAsync`, `imgPulseCreate`, `imgPulseDispose`, `imgPulseRate`, `imgPulseStart`, and `imgPulseStop`.

You can use signal I/O functions to control the trigger lines on IMAQ devices. Use these functions to start an acquisition based on a trigger, output status signals on a trigger line, wait for a specified signal to occur, or output pulses on the trigger lines.

# imgSessionTriggerConfigure

## Format

```
rval = imgSessionTriggerConfigure(SESSION_ID sid, uInt32 trig_num,
                          uInt32 trig_polarity, uInt32 time_out,
                          uInt32 trig_action)
```

## Purpose

Configures an acquisition to start based on an external trigger.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **trig_num** | uInt32 | input | trigger line on which pulse is generated |
| **trig_polarity** | uInt32 | input | polarity |
| **time_out** | uInt32 | input | time to wait for image |
| **trig_action** | uInt32 | input | start acquisition based on trigger, start each buffer list based on trigger, trigger each buffer |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**trig_num** is the source of the trigger signal as specified by the constants:

| | |
|---|---|
| IMG_EXT_TRIG0 | IMG_EXT_RTSI2 |
| IMG_EXT_TRIG1 | IMG_EXT_RTSI3 |
| IMG_EXT_TRIG2 | IMG_EXT_RTSI4 |
| IMG_EXT_TRIG3 | IMG_EXT_RTSI5 |
| IMG_EXT_RTSI0 | IMG_EXT_RTSI6 |
| IMG_EXT_RTSI1 | |

**Note** IMG_EXT_TRIG <0..3> refers to the external trigger lines of your IMAQ board; IMG_EXT_RTSI <0..6> refers to the internal pins on the RTSI controller of your IMAQ board.

**trig_polarity** is the polarity of the trigger line as defined by the constants:

IMG_TRIG_POLAR_ACTIVEL
IMG_TRIG_POLAR_ACTIVEH

**time_out** is the amount of time in milliseconds to wait for the trigger to occur and the image to be captured.

**trig_action** specifies if an assertion edge of **trig_num** should start an acquisition. Values are:

IMG_TRIG_ACTION_NONE
IMG_TRIG_ACTION_CAPTURE
IMG_TRIG_ACTION_BUFLIST
IMG_TRIG_ACTION_BUFFER

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionLineTrigSource

## Format

```
rval = imgSessionLineTrigSource(SESSION_ID sid, uInt32 trig_source,
                                uInt32 trig_polarity, uInt32 skip_number)
```

## Purpose

Configures triggering per line for acquisition from a line scan camera. Use this function to require a trigger to start the acquisition of each line from a line scan camera.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **trig_source** | uInt32 | input | source of the trigger signal |
| **trig_polarity** | uInt32 | input | polarity of the trigger line |
| **skip_number** | uInt32 | input | number of triggers to wait before acquiring a new line |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**trig_source** is the source of the trigger signal as specified by the following constants:

IMG_EXT_TRIG0
IMG_EXT_TRIG1
IMG_EXT_TRIG2
IMG_EXT_TRIG3
IMG_EXT_RTSI0
IMG_EXT_RTSI1
IMG_EXT_RTSI2
IMG_EXT_RTSI3
IMG_EXT_RTSI4
IMG_EXT_RTSI5
IMG_EXT_RTSI6

**Note** IMG_EXT_TRIG <0..3> refers to the external trigger lines of your IMAQ board; IMG_EXT_RTSI <0..6> refers to the internal pins on the RTSI controller of your IMAQ board.

**trig_polarity** is the polarity of the trigger line as defined by the constants:

IMG_TRIG_POLAR_ACTIVEL
IMG_TRIG_POLAR_ACTIVEH

**skip_number** is the number of triggers to wait before acquiring a new line. For example, if you are using an encoder to trigger lines and it outputs 1,000 ticks per revolution but you want to acquire only 10 lines per revolution, set **skip_number** to 99. Set **skip_number** to 0 to acquire a line on every trigger from the encoder.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionTriggerClear

## Format

```
rval = imgSessionTriggerClear(SESSION_ID sid)
```

## Purpose

Disables all triggers on the session.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionTriggerDrive

## Format

```
rval = imgSessionTriggerDrive(SESSION_ID sid, uInt32 trig_num,
                              uInt32 trig_polarity,uInt32 trig_drive)
```

## Purpose

Configures the specified trigger line to drive a signal out.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **trig_num** | uInt32 | input | trigger line on which pulse is generated |
| **trig_polarity** | uInt32 | input | polarity |
| **trig_drive** | uInt32 | input | HIGH, LOW, internal status signals |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**trig_num** is the trigger line to drive as specified by the constants:

IMG_EXT_TRIG0
IMG_EXT_TRIG1
IMG_EXT_TRIG2
IMG_EXT_TRIG3
IMG_EXT_RTSI0
IMG_EXT_RTSI1
IMG_EXT_RTSI2
IMG_EXT_RTSI3
IMG_EXT_RTSI4
IMG_EXT_RTSI5
IMG_EXT_RTSI6

**Note** IMG_EXT_TRIG <0..3> refers to the external trigger lines of your IMAQ board; IMG_EXT_RTSI <0..6> refers to the internal pins on the RTSI controller of your IMAQ board.

**trig_polarity** is the polarity of the trigger line as defined by the constants:

IMG_TRIG_POLAR_ACTIVEL
IMG_TRIG_POLAR_ACTIVEH

**trig_drive** specifies the signal that drives the trigger line as specified by the constants:

IMG_TRIG_DRIVE_DISABLED
IMG_TRIG_DRIVE_AQ_IN_PROGRESS
IMG_TRIG_DRIVE_AQ_DONE
IMG_TRIG_DRIVE_ASSERTED
IMG_TRIG_DRIVE_UNASSERTED
IMG_TRIG_DRIVE_HSYNC
IMG_TRIG_DRIVE_VSYNC
IMG_TRIG_DRIVE_PIXEL_CLK (not valid on digital IMAQ boards)
IMG_TRIG_DRIVE_FRAME_START
IMG_TRIG_DRIVE_FRAME_DONE

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionTriggerRead

## Format

```
rval = imgSessionTriggerRead(SESSION_ID sid, uInt32 trig_num,
                             uInt32 trig_polarity,uInt32* status)
```

## Purpose

Reads the current value of the specified trigger line.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **trig_num** | uInt32 | input | trigger line on which pulse is generated |
| **trig_polarity** | uInt32 | input | polarity |
| **status** | uInt32* | output | value of trigger line |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**trig_num** is the trigger line to read as specified by the constants:

IMG_EXT_TRIG0
IMG_EXT_TRIG1
IMG_EXT_TRIG2
IMG_EXT_TRIG3
IMG_EXT_RTSI0
IMG_EXT_RTSI1
IMG_EXT_RTSI2
IMG_EXT_RTSI3
IMG_EXT_RTSI4
IMG_EXT_RTSI5
IMG_EXT_RTSI6

**Note** IMG_EXT_TRIG <0..3> refers to the external trigger lines of your IMAQ board; IMG_EXT_RTSI <0..6> refers to the internal pins on the RTSI controller of your IMAQ board.

**trig_polarity** is the polarity of the trigger line as defined by the constants:

IMG_TRIG_POLAR_ACTIVEL
IMG_TRIG_POLAR_ACTIVEH

**status** is a pointer to an area of memory reserved as a trigger status variable. Returns TRUE if the trigger is currently asserted, FALSE if it is unasserted.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionWaitSignal

## Format

```
rval = imgSessionWaitSignal(SESSION_ID sid, uInt32 signal,
                            uInt32 signal_pol,uInt32 timeout)
```

## Purpose

Waits for a signal to be asserted. This function returns when the specified signal is asserted.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **signal** | uInt32 | input | signal which will cause the pulse to be generated |
| **signal_pol** | uInt32 | input | polarity of signal |
| **timeout** | uInt32 | input | time to wait for image |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**signal** is the assertion edge of the signal that causes the function to return as specified by the constants:

IMG_AQ_IN_PROGRESS
IMG_AQ_DONE
IMG_FRAME_START
IMG_FRAME_DONE
IMG_BUF_COMPLETE
IMG_EXT_TRIG0
IMG_EXT_TRIG1
IMG_EXT_TRIG2
IMG_EXT_TRIG3
IMG_EXT_RTSI0
IMG_EXT_RTSI1
IMG_EXT_RTSI2
IMG_EXT_RTSI3
IMG_EXT_RTSI4
IMG_EXT_RTSI5
IMG_EXT_RTSI6

**Note** IMG_EXT_TRIG <0..3> refers to the external trigger lines of your IMAQ board; IMG_EXT_RTSI <0..6> refers to the internal pins on the RTSI controller of your IMAQ board.

**signal_pol** is the polarity of the signal input as defined by the constants:

IMG_TRIG_POLAR_ACTIVEL
IMG_TRIG_POLAR_ACTIVEH

**Note** This input is valid only for external triggers and RTSI lines. It is ignored for all other signals.

**timeout** is the amount of time in milliseconds to wait for the assertion edge of signal.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionWaitSignalAsync

## Format

```
rval = imgSessionWaitSignalAsync(SESSION_ID sid, uInt32 signal,
                          uInt32 signal_pol,CALL_BACK_PTR funcptr,
                          void* callback_data)
```

## Purpose

Monitors for a signal to be asserted and invokes a user-defined callback when the signal is asserted.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **signal** | uInt32 | input | signal which will cause the pulse to be generated |
| **signal_pol** | uInt32 | input | polarity of signal |
| **funcptr** | CALL_BACK_PTR | input | callback function |
| **callback_data** | void* | input | value passed to the callback function |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**signal** is the assertion edge of the signal that causes the callback function to be invoked as defined by the constants:

IMG_AQ_IN_PROGRESSIMG_EXT_TRIG3
IMG_AQ_DONEIMG_EXT_RTSI0
IMG_FRAME_STARTIMG_EXT_RTSI1
IMG_FRAME_DONEIMG_EXT_RTSI2
IMG_BUF_COMPLETEIMG_EXT_RTSI3
IMG_EXT_TRIG0IMG_EXT_RTSI4
IMG_EXT_TRIG1IMG_EXT_RTSI5
IMG_EXT_TRIG2IMG_EXT_RTSI6

**Note** IMG_EXT_TRIG <0..3> refers to the external trigger lines of your IMAQ board; IMG_EXT_RTSI <0..6> refers to the internal pins on the RTSI controller of your IMAQ board.

**signal_pol** is the polarity of the signal input as defined by the constants:

IMG_TRIG_POLAR_ACTIVEL
IMG_TRIG_POLAR_ACTIVEH

**Note** This input is valid only for external triggers and RTSI lines. It is ignored for all other signals.

**funcptr** is a pointer to the callback function, which is specified by the following function prototype:

```
uInt32(*CALLBACK_PTR)(SESSION_ID sid, IMG_ERR err, uInt32 signal,
void* userdata)
```

**callback_data** is a four-byte value that is passed to the callback function. The value can be a pointer to user data.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgPulseCreate

## Format

```
rval = imgPulseCreate(uInt32 timebase, uInt32 delay, uInt32 width,
                      uInt32 signal_source,uInt32 signal_polarity,
                      uInt32 output, uInt32 output_polarity,
                      uInt32 pulse_mode, PULSE_ID* plsID)
```

## Purpose

Configures the attributes of a pulse. A single pulse consists of a delay phase (phase 1), followed by a pulse phase (phase 2), and then a return to the phase 1 level.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **timebase** | uInt32 | input | timebase value |
| **delay** | uInt32 | input | interval before the pulse |
| **width** | uInt32 | input | interval of the pulse |
| **signal_source** | uInt32 | input | signal that triggers the pulse generation |
| **signal_polarity** | uInt32 | input | polarity of the signal |
| **output** | uInt32 | input | trigger line on which the pulse is generated |
| **output_polarity** | uInt32 | input | the polarity of the pulse output |
| **pulse_mode** | uInt32 | input | indicates if the pulse is repeated |
| **plsID** | PULSE_ID* | input | pulse ID created and configured with pulse functions |
| **rval** | Int32 | output | status |

## Parameter Discussion

**timebase** selects the timebase, or resolution, to be used by the counter. **timebase** has the following possible values:

PULSE_TIMEBASE_50MHZ
PULSE_TIMEBASE_100KHZ
PULSE_TIMEBASE_PIXELCLK

**delay** is the desired duration of the first phase of the pulse, phase 1. The unit is cycles of the **timebase**. Use the following formula to determine the actual time period that delay represents:

**delay** × (**timebase** resolution)

**width** is the desired duration of the second phase of the pulse, phase 2. The unit is cycles of the **timebase**. Use the following formula to determine the actual time period that width represents:

**width** × (**timebase** resolution)

**signal_source** specifies the signal that will cause the pulse to be generated. The assertion edge of the following signals can initiate pulse generation, as specified by the following constants:

| | |
|---|---|
| IMG_IMMEDIATE | IMG_EXT_TRIG3 |
| IMG_AQ_IN_PROGRESS | IMG_EXT_RTSI0 |
| IMG_AQ_DONE | IMG_EXT_RTSI1 |
| IMG_FRAME_START | IMG_EXT_RTSI2 |
| IMG_FRAME_DONE | IMG_EXT_RTSI3 |
| IMG_EXT_TRIG0 | IMG_EXT_RTSI4 |
| IMG_EXT_TRIG1 | IMG_EXT_RTSI5 |
| IMG_EXT_TRIG2 | IMG_EXT_RTSI6 |

**Note** IMG_EXT_TRIG <0..3> refers to the external trigger lines of your IMAQ board; IMG_EXT_RTSI <0..6> refers to the internal pins on the RTSI controller of your IMAQ board.

**signal_polarity** is the polarity of the signal input as defined by the constants:

IMG_TRIG_POLAR_ACTIVEL
IMG_TRIG_POLAR_ACTIVEH

**Note** This input is valid only for external triggers and RTSI lines. It is ignored for all other signals.

**output** is the trigger line on which the pulse is generated, as specified by the following constants:

| | |
|---|---|
| IMG_EXT_TRIG0 | IMG_EXT_RTSI2 |
| IMG_EXT_TRIG1 | IMG_EXT_RTSI3 |
| IMG_EXT_TRIG2 | IMG_EXT_RTSI4 |
| IMG_EXT_TRIG2 | IMG_EXT_RTSI5 |
| IMG_EXT_RTSI0 | IMG_EXT_RTSI6 |
| IMG_EXT_RTSI1 | |

**Note** IMG_EXT_TRIG <0..3> refers to the external trigger lines of your IMAQ board; IMG_EXT_RTSI <0..6> refers to the internal pins on the RTSI controller of your IMAQ board.

**output_polarity** is the polarity of the pulse output as defined by the following constants:

IMG_PULSE_POLAR_ACTIVEL
IMG_PULSE_POLAR_ACTIVEH

**pulse_mode** indicates if the pulse is generated once or continuously, as specified by the following constants:

PULSE_MODE_TRAIN
PULSE_MODE_SINGLE
PULSE_MODE_SINGLE_REARM

**plsID** passes a pointer to an area of memory reserved as a PULSE_ID variable. If the function succeeds, the variable contains a valid PULSE_ID that can be used in subsequent functions.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgPulseDispose

## Format

```
rval = imgPulseDispose(PULSE_ID plsID)
```

## Purpose

Disposes of a pulse ID.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **plsID** | PULSE_ID | input | pulse ID |
| **rval** | Int32 | output | status |

## Parameter Discussion

**plsID** is a valid PULSE_ID variable.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgPulseRate

## Format

```
rval = imgPulseRate(uInt32 delaytime, uInt32 widthtime, uInt32* delay,
                    uInt32* width,uInt32* timebase)
```

## Purpose

Converts delay and width into delay, width, and timebase values needed by `imgPulseCreate`.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **delaytime** | uInt32 | input | interval before the pulse in microseconds |
| **widthtime** | uInt32 | input | interval of the pulse in microseconds |
| **delay** | uInt32* | output | interval before the pulse in cycles |
| **width** | uInt32* | output | interval of the pulse in cycles |
| **timebase** | uInt32* | output | timebase source |
| **rval** | Int32 | output | status |

## Parameter Discussion

**delaytime** is the desired duration of the first phase of the pulse in microseconds.

**widthtime** is the desired duration of the second phase of the pulse in microseconds.

**delay** represents the number of cycles of timebase of the first phase of the pulse.

**width** represents the number of cycles of timebase of the second phase of the pulse.

**timebase** is a code that represents the timebase on the board that the counter uses to produce the pulse.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgPulseStart

```
rval = imgPulseStart(PULSE_ID pid, SESSION_ID sid)
```

## Purpose

Starts the generation of a pulse. You must call `imgPulseCreate` first to configure the pulse.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **pid** | PULSE_ID | input | pulse ID |
| **sid** | SESSION_ID | input | session ID |
| **rval** | Int32 | output | status |

## Parameter Discussion

**pid** is a valid PULSE_ID variable.

**sid** is a valid SESSION_ID variable.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgPulseStop

```
rval = imgPulseStop(PULSE_ID pid)
```

## Purpose

Stops the generation of a pulse.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **pid** | PULSE_ID | input | pulse ID |
| **rval** | Int32 | output | status |

## Parameter Discussion

**pid** is a valid PULSE_ID variable.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# Miscellaneous Functions

Miscellaneous functions include `imgSessionStatus`, `imgSessionSetROI`, `imgSessionGetROI`, and `imgSessionGetBufferSize`.

Use these functions to obtain status information on a session, get and set a region of interest, and get the buffer size required for a session based on current attributes.

# imgSessionStatus

## Format

```
rval = imgSessionStatus(SESSION_ID sid, uInt32* status,
                        uInt32* bufferNumber)
```

## Purpose

Gets the current session status.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **status** | uInt32* | output | current session status |
| **bufferNumber** | uInt32* | output | pointer to an area of memory reserved for the current buffer number |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**status** indicates the current session status. If **status** is non-zero, the session currently is acquiring. If **status** is zero, the session is idle. **status** is the value returned by IMG_ATTR_ACQ_IN_PROGRESS in `imgGetAttribute`.

**bufferNumber** is a pointer to an area of memory reserved for the current buffer number.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionSetROI

## Format

```
rval = imgSessionSetROI(SESSION_ID sid, uInt32 top, uInt32 left,
                        uInt32 height, uInt32 width)
```

## Purpose

Sets the acquisition region of interest. You typically make this call after creating a session and before calling the `imgSessionStartAcquisition`. This function modifies the following attributes:

IMG_ATTR_ROI_TOP
IMG_ATTR_ROI_LEFT
IMG_ATTR_ROI_HEIGHT
IMG_ATTR_ROI_WIDTH

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **top** | uInt32 | output | top ordinate of the first pixel transferred |
| **left** | uInt32 | output | left ordinate of the first pixel transferred |
| **height** | uInt32 | output | height of the rectangle to transfer |
| **width** | uInt32 | output | width of the rectangle to transfer |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**top** indicates the top vertical offset of the first pixel transferred.

**left** indicates the left horizontal offset of the first pixel transferred.

**height** indicates the height of the area to transfer.

**width** indicates the width of the area to transfer.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionGetROI

## Format

```
rval = imgSessionGetROI(SESSION_ID sid, uInt32* top, uInt32* left,
                        uInt32* height,uInt32* width)
```

## Purpose

Gets the acquisition region of interest.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **top** | uInt32* | output | current value of IMG_ATTR_ROI_TOP |
| **left** | uInt32* | output | current value of IMG_ATTR_ROI_LEFT |
| **height** | uInt32* | output | current value of IMG_ATTR_ROI_HEIGHT |
| **width** | uInt32* | output | current value of IMG_ATTR_ROI_WIDTH |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**top** indicates the top vertical offset of the first pixel transferred.

**left** indicates the left horizontal offset of the first pixel transferred.

**height** indicates the height of the area to transfer.

**width** indicates the width of the area to transfer.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionGetBufferSize

## Format

```
rval = imgSessionGetBufferSize(SESSION_ID sid, uInt32* sizeNeeded)
```

## Purpose

Gets the minimum buffer size needed for frame buffer allocation. This function calculates the buffer size by using the following attributes:

IMG_ATTR_ROI_HEIGHT
IMG_ATTR_ROWBYTES
IMG_ATTR_YOFF_BUFFER

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **sizeNeeded** | uInt32* | output | buffer size |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**sizeNeeded** returns the buffer size needed for an image based on the attributes listed in the Purpose.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# 4

# Low-Level Functions

This chapter contains a detailed explanation of each low-level NI-IMAQ function. The functions are arranged alphabetically under the type of image acquisition procedure—acquisition, attribute, buffer management, interface, and utility.

Low-level functions let you perform in-depth tasks that require a more advanced understanding of the IMAQ hardware and image acquisition, such as directly controlling video parameters like gain and offset level or locking down a buffer during a continuous acquisition.

## Acquisition Functions

Acquisition functions include `imgMemLock`, `imgMemUnlock`, `imgSessionAbort`, `imgSessionAcquire`, `imgSessionConfigure`, `imgSessionCopyArea`, `imgSessionCopyBuffer`, `imgSessionExamineBuffer`, and `imgSessionReleaseBuffer`.

Use these functions to configure, start, and abort an image acquisition. These functions also let you examine a buffer during acquisition.

# imgMemLock

## Format

```
rval = imgMemLock(BUFLIST_ID bid)
```

## Purpose

Locks in memory all image buffers associated with the given buffer list in preparation for an acquisition. The buffers must be successfully locked down before an acquisition can begin. The function does not do anything if the buffers exist in onboard memory.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **bid** | BUFLIST_ID | input | buffer list ID |
| **rval** | Int32 | output | status |

## Parameter Discussion

**bid** is a valid BUFLIST_ID variable.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgMemUnlock

## Format

```
rval = imgMemUnlock(BUFLIST_ID bid)
```

## Purpose

Unlocks all buffers associated with the given buffer list.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **bid** | BUFLIST_ID | input | buffer list ID |
| **rval** | Int32 | output | status |

## Parameter Discussion

**bid** is a valid BUFLIST_ID variable.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionAbort

## Format

```
rval = imgSessionAbort(SESSION_ID sid, uInt32* buf_num)
```

## Purpose

Stops an asynchronous acquisition or synchronous continuous acquisition immediately.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **buf_num** | uInt32* | output | pointer to the last valid buffer number |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**buf_num** returns the last valid buffer number.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionAcquire

## Format

```
rval = imgSessionAcquire(SESSION_ID sid, uInt32 async,
                         CALL_BACK_PTR callback)
```

## Purpose

Starts an acquisition synchronously or asynchronously to the frame buffers in the associated session buffer list. If the acquisition is triggered, the acquisition into the frame buffers will wait for the trigger.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **async** | uInt32 | input | asynchronous flag |
| **callback** | CALL_BACK_PTR | input | user completion routine |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**async** is the asynchronous flag. If **async** is non-zero, it indicates an asynchronous acquisition. If **async** is zero, it indicates a synchronous acquisition.

**callback** is a pointer to a user completion routine. NI-IMAQ calls this routine at the completion of the acquisition. Pass NULL if you do not want to call a completion routine.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionConfigure

## Format

```
rval = imgSessionConfigure(SESSION_ID sid, BUFLIST_ID buflist)
```

## Purpose

Specifies the buffer list to use with this session. You must pass a valid BUFLIST_ID. Upon successful completion of this call, you can then call `imgSessionAcquire`.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **buflist** | BUFLIST_ID | input | a valid BUFLIST_ID |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**buflist** is a valid BUFLIST_ID variable.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionCopyArea

## Format

```
rval = imgSessionCopyArea(SESSION_ID sid, uInt32 buf_num, uInt32 top,
                          uInt32 left,uInt32 height, uInt32 width,
                          Ptr buffer, uInt32 rowPixels, uInt32 vsync)
```

## Purpose

Copies an area of a session's buffer to a user-specified buffer.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **buf_num** | uInt32 | input | element number of buffer to copy |
| **top** | uInt32 | input | top vertical coordinate of area |
| **left** | uInt32 | input | left horizontal coordinate of area |
| **width** | uInt32 | input | width of area |
| **height** | uInt32 | input | height of area |
| **buffer** | Ptr | input | pointer to user image memory |
| **rowPixels** | uInt32 | input | used in calculating the address of the next line |
| **vsync** | uInt32 | input | wait until the next vertical blank to perform the copy |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**buf_num** indicates a valid buffer list element number of the buffer to copy from.

**top** indicates the top vertical coordinate of the area to copy.

**left** indicates the left horizontal coordinate.

**width** indicates the width of the area to copy.

**height** indicates the height of the area to copy.

**buffer** indicates an address to a buffer that is large enough to hold the data.

**rowPixels** indicates the exact pixel-width of the horizontal line to acquire. This parameter specifies the number of pixels to add to the line pointer for the next scan line. This value must be greater than or equal to the width parameter. Passing a zero for this value causes it to be ignored.

**vsync** controls when copying occurs. If **vsync** is TRUE, `imgSessionCopyArea` waits until the next vertical blank to perform the copy. If **vsync** is FALSE, `imgSessionCopyArea` does not wait.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionCopyBuffer

## Format

```
rval = imgSessionCopyBuffer(SESSION_ID sid, uInt32 buf_num, uInt8* buffer,
                            uInt32 vsync)
```

## Purpose

Copies a session's image data to a user buffer.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **buf_num** | uInt32 | input | element number of buffer to copy |
| **buffer** | uInt8* | input | pointer to user buffer |
| **vsync** | uInt32 | input | wait until the next vertical blank to perform |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**buf_num** indicates a buffer list element number that corresponds to the buffer you want to copy.

**buffer** points to an area of memory to receive the copy.

**vsync** controls a session's image data transfer. If **vsync** is TRUE, `imgSessionCopyBuffer` waits until a host vertical blank occurs before transferring. If **vsync** is FALSE, `imgSessionCopyBuffer` does not wait for a host vertical blank.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionExamineBuffer

## Format

```
rval = imgSessionExamineBuffer(SESSION_ID sid, uInt32 whichBuffer,
                               uInt32* bufferNumber, uInt32* bufferAddr)
```

## Purpose

Extracts a buffer from a live acquisition. This function lets you lock a buffer out of a continuous loop sequence for processing when you are using a ring (continuous) sequence. If the requested buffer has been acquired and exists in memory, the function will return that buffer immediately. If the requested buffer has not yet been acquired, the function will not return until the buffer has been acquired or the timeout period has expired. If the requested buffer has already been overwritten, the function will return the most current buffer.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **whichBuffer** | uInt32 | input | identifies which buffer to get |
| **bufferNumber** | uInt32* | input | element number of the returned buffer |
| **bufferAddr** | uInt32* | input | a pointer to an address to store the address of the locked buffer |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**whichBuffer** identifies which cumulative frame buffer to get.

**bufferNumber** returns the buffer number of the returned buffer.

**bufferAddr** is a pointer to an address to store the address of the locked buffer.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

**Note** Use `imgSessionReleaseBuffer` to release the buffer being held with `imgSessionExamineBuffer`.

# imgSessionReleaseBuffer

## Format

```
rval = imgSessionReleaseBuffer(SESSION_ID sid)
```

## Purpose

Releases a buffer that was previously held with `imgSessionExamineBuffer`. This function has the effect of re-entering a buffer into a continuous ring buffer pool after analysis.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# Attribute Functions

Attribute functions include `imgGetAttribute`, `imgGetCameraAttributeNumeric`, `imgGetCameraAttributeString`, `imgSessionGetLostFramesList`, `imgSessionSetUserLUT8bits`, `ImgSessionSetUserLUT16bits`, `imgSetAttribute`, `imgSetCameraAttributeNumeric`, and `imgSetCameraAttributeString`.

You can use these functions to examine and change NI-IMAQ and camera attributes.

When changing NI-IMAQ attributes, remember that attributes are either interface- or session-(channel) specific.

Some attribute changes such as gain or white reference that are session-specific can take effect while a live acquisition is in progress. In this case, the driver will wait for a vertical blank before making the change. Most session attributes, however, require that you call `imgSessionConfigure` to reconfigure the driver, especially when changing the ROI width or height.

NI-IMAQ does not let you change any attribute that would have a detrimental effect on any acquisition in progress. If NI-IMAQ lets you change an attribute during a live acquisition, you should see the effect of the change immediately. If NI-IMAQ does not let you change an attribute during a live acquisition, stop the acquisition, change the attribute, call `imgSessionConfigure`, and restart the acquisition.

Calling `imgSessionConfigure` reprograms the video hardware.

# imgGetAttribute

## Format

```
rval = imgGetAttribute(uInt32 void_id, uInt32 type, void* value)
```

## Purpose

Returns an attribute for an interface or session.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **void_id** | uInt32 | input | session or interface ID |
| **type** | uInt32 | input | attribute type |
| **value** | void* | input/ output | pointer to a place attribute value |
| **rval** | Int32 | output | status |

## Parameter Discussion

**void_id** indicates an area of memory reserved for a valid SESSION_ID or INTERFACE_ID variable.

**type** passes a valid SESSION_ID or INTERFACE_ID attribute type. See Appendix A, *Attributes and Constants*, for valid types.

**value** passes a pointer to place attribute value.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

**Note** When you use an IMAQ PCI/PXI-1411 device, the following attributes of the `imgGetAttribute` function behave differently than they did in previous versions of NI-IMAQ:

IMG_ATTR_COLOR_CONTRAST
IMG_ATTR_COLOR_SATURATION
IMG_ATTR_COLOR_BRIGHTNES

For more information, see the *IMAQ PCI/PXI-1411 User Manual.*

# imgGetCameraAttributeNumeric

## Format

```
rval = imgGetCameraAttributeNumeric (SESSION_ID sid, Int8* attributeString,
                                     double* currentValueNumeric)
```

## Purpose

Gets the value of numeric camera attributes. Consult the `<my camera>.txt` file in the `ni-imaq\camera info` directory for information on valid attributes for your camera.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **attributeString** | int8* | input | string containing attribute name |
| **currentValueNumeric** | double* | input | pointer to place attribute value |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**attributeString** is a string containing the attribute name.

**currentValueNumeric** is a pointer to place the attribute value. After the function is called, this parameter contains the current value of the attribute.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgGetCameraAttributeString

## Format

**rval =** `imgGetCameraAttributeString (SESSION_ID sid, Int8* attributeString, Int8* currentValueString, uInt32 sizeofCurrentValueString)`

## Purpose

Gets the value of camera attributes. Consult the `<my camera>.txt` file in the `ni-imaq\camera info` directory for information on valid attributes for your camera. Use this function to get the value of string, integer, or float attribute types.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **attributeString** | int8* | input | string containing attribute name |
| **currentValueString** | int8* | input | string containing attribute value |
| **sizeofCurrentValueString** | uInt32 | input | size of **currentValueString** |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**attributeString** is a string containing the attribute name.

**currentValueString** is a pointer to an array in memory large enough to hold the attribute value returned. After the function is called, this parameter contains the current value of the attribute.

**sizeofCurrentValueString** is the size of the array in memory pointed to by **currentValueString**.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionGetLostFramesList

## Format

```
rval = imgSessionGetLostFramesList(SESSION_ID sid, uInt32* framelist,
                                   uInt32 numEntries)
```

## Purpose

Gets information about frames that were overwritten during a continuous acquisition. Use this function during a ring acquisition to determine if any frames were overwritten before being examined.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **framelist** | uInt32* | input | pointer to cumulative frame number array |
| **numEntries** | uInt32 | input | size of **framelist** array |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**framelist** is a pointer to an array of user-allocated memory that contains the cumulative frame number of any image buffer that was overwritten during an acquisition.

**numEntries** is the size of the framelist array.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionSetUserLUT8bits

## Format

```
rval = imgSessionSetUserLUT8bits(SESSION_ID sid, uInt32 lutType, uInt8* lut)
```

## Purpose

Downloads a custom 8-bit LUT to your IMAQ device. If your camera has more than one LUT, you must program all of them. Call this function at least once with the default constant (IMG_LUT_TYPE_DEFAULT) to initialize all LUTs, or make successive calls using different constants for each LUT. You can also override the default LUT on a per-tap basis. This function works with analog IMAQ devices or digital IMAQ devices acquiring from an 8-bit camera.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **lutType** | uInt32 | input | LUT to be downloaded |
| **lut** | uInt8* | input | pointer to user buffer containing the LUT |
| **rval** | Int32 | output | no error |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**lutType** indicates the type of LUT to be written. Valid values are:

| | |
|---|---|
| IMG_LUT_TYPE_DEFAULT | IMG_LUT_TYPE_TAP0 |
| IMG_LUT_TYPE_RED | IMG_LUT_TYPE_TAP1 |
| IMG_LUT_TYPE_GREEN | IMG_LUT_TYPE_TAP2 |
| IMG_LUT_TYPE_BLUE | IMG_LUT_TYPE_TAP3 |

**lut** points to an area of memory that contains the LUT to be downloaded to the image acquisition board. This array must contain 256 elements.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionSetUserLUT16bits

## Format

```
rval = imgSessionSetUserLUT16bits(SESSION_ID sid, uInt32 lutType,
                                  uInt16* lut)
```

## Purpose

Downloads a custom 16-bit LUT to your IMAQ device. If your camera has more than one LUT, you must program all of them. Call this function at least once with the default constant (IMG_LUT_TYPE_DEFAULT) to initialize all LUTs, or make successive calls using different constants for each LUT. You can also override the default LUT on a per-tap basis. This function works with digital cameras of 10-, 12-, 14-, or 16-bit pixel depths.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **lutType** | uInt32 | input | LUT to be downloaded |
| **lut** | uInt16* | input | pointer to user buffer containing the LUT |
| **rval** | Int32 | output | no error |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**lutType** indicates the type of LUT to be written. Valid values are:

IMG_LUT_TYPE_DEFAULT
IMG_LUT_TYPE_TAP0
IMG_LUT_TYPE_TAP1

**lut** points to an area of memory that contains the LUT to be downloaded to the IMAQ device. This array must contain $2^n$ elements where $n$ is the bit depth of the camera (1,024 for 10-bit cameras, 2,048 for 12-bit cameras, and so on).

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSetAttribute

## Format

```
rval = imgSetAttribute(uInt32 void_id, uInt32 type, uInt32 value)
```

## Purpose

Sets an attribute for an interface or session.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **void_id** | uInt32 | input | session or interface ID |
| **type** | uInt32 | input | attribute type |
| **value** | uInt32 or double* | input | new attribute value |
| **rval** | Int32 | output | status |

## Parameter Discussion

**void_id** indicates an area of memory reserved for a valid SESSION_ID or INTERFACE_ID variable.

**type** passes a valid SESSION_ID or INTERFACE_ID attribute type. See Appendix A, *Attributes and Constants*, for valid types.

**value** is the new attribute value. The type of this input is either a uInt32 or a pointer to a double. If the attribute is of type uInt32, use this function as follows:

```
uInt32 value;
imgSetAttribute (sid, IMG_ATTR_FRAME_COUNT, value)
```

If the attribute is of type double*, use this function as follows:

```
double value;
imgSetAttribute (sid, IMG_ATTR_BLACK_REF_VOLT,(uInt32) &value)
```

See Appendix A, *Attributes and Constants*, for the attribute type.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSetCameraAttributeNumeric

## Format

```
rval = imgSetCameraAttributeNumeric (SESSION_ID sid, Int8* attributeString,
                                     double newValueNumeric)
```

## Purpose

Sets the value of numeric camera attributes. Consult the `<my camera>.txt` file in the `ni-imaq\camera info` directory for information on valid attributes for your camera.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **attributeString** | int8* | input | string containing attribute name |
| **newValueNumeric** | double | input | pointer to place attribute value |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**attributeString** is a string containing the attribute name.

**newValueNumeric** is a numeric containing the new value of the attribute. You can find valid attribute values in the `<my camera>.txt` file.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSetCameraAttributeString

## Format

```
rval = imgSetCameraAttributeString(SESSION_ID sid, Int8* attributeString,
                                   Int8* newValueString)
```

## Purpose

Sets the value of camera attributes. Consult the `<my camera>.txt` file in the `ni-imaq\camera info` directory for information on valid attributes for your camera. Use this function to set the value of string, integer, or float attribute types. If the attribute is a numeric type (integer or float), this function converts the string input into a numeric value.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **attributeString** | int8* | input | string containing attribute name |
| **newValueString** | int8* | input | string containing attribute value |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**attributeString** is a string containing the attribute name.

**newValueString** is a string containing the new value of the attribute. You can find valid attribute values in the `<my camera>.txt` file.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# Buffer Management Functions

Buffer management functions include `imgCreateBuffer`, `imgCreateBufList`, `imgDisposeBuffer`, `imgDisposeBufList`, `imgGetBufferElement`, `imgSessionClearBuffer`, and `imgSetBufferElement`.

Use these functions to set up objects such as buffer lists and buffers. When changing buffer list elements, make sure no other sessions depend on that buffer list to be in a known state.

# imgCreateBuffer

## Format

```
rval = imgCreateBuffer(SESSION_ID sid, uInt32 where, uInt32 bufferSize,
                       void* bufPtrAddr)
```

## Purpose

Creates a user frame buffer based on the geometric definitions of the associated session. Passing a NULL or zero for the SESSION_ID is valid. In this case, you must pass a buffer size. If buffer size is zero, the buffer size is computed based on the ROI height × rowPixels for the associated session multiplied times the number of bytes per pixel. The function returns an error if the buffer size is smaller than the minimum buffer size required for the session.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **where** | uInt32 | input | indicates where the buffer should be stored |
| **bufferSize** | uInt32 | input | size of buffer to create |
| **bufPtrAddr** | void* | output | pointer to return the buffer address |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**where** indicates if the buffer should be stored in system memory or in onboard memory on the IMAQ device, as specified by the constants:

IMG_HOST_FRAME
IMG_DEVICE_FRAME

**bufferSize** indicates the size of the buffer you want to create.

**bufPtrAddr** is a pointer to an area of memory that stores the new buffer address.

**Note** If you use IMG_DEVICE_FRAME, do not access the returned **bufPtrAddr**.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgCreateBufList

## Format

```
rval = imgCreateBufList(uInt32 numElements, BUFLIST_ID* bid)
```

## Purpose

Creates a buffer list. `imgCreateBufList` passes this buffer list to `imgSessionConfigure`. You must initialize the buffer list to an empty default state and then fill it before calling `imgSessionConfigure`. Use `imgSetBufferElement` to fill the buffer list.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **numElements** | uInt32 | input | number of entries in the buffer list |
| **bid** | BUFLIST_ID* | output | pointer to the new buffer list |
| **rval** | Int32 | output | status |

## Parameter Discussion

**numElements** indicates the maximum number of elements the buffer list should contain.

**bid** is a pointer to an area of memory that contains a BUFLIST_ID variable.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgDisposeBuffer

## Format

```
rval = imgDisposeBuffer(void* buffPtrAddr)
```

## Purpose

Disposes of a user frame buffer.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **buffPtrAddr** | void* | input | pointer to a buffer |
| **rval** | Int32 | output | status |

## Parameter Discussion

**buffPtrAddr** is a pointer to buffer created by `imgCreateBuffer`.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

**Note** Make sure no active buffer lists contain this buffer before disposing of the user frame buffer.

# imgDisposeBufList

## Format

```
rval = imgDisposeBufList(BUFLIST_ID bid, uInt32 freeResources)
```

## Purpose

Purges all image buffers associated with this buffer list. You must call `imgSessionConfigure` to reconfigure any session that was attached to the purged buffer list.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **bid** | BUFLIST_ID | input | valid BUFLIST_ID |
| **freeResources** | uInt32 | input | determines whether buffers and buffer list will be disposed or just the buffer list |
| **rval** | Int32 | output | status |

## Parameter Discussion

**bid** is a valid BUFLIST_ID variable.

**freeResources** determines whether both the buffers and the buffer list or just the buffer list will be disposed. If **freeResources** is non-zero, it indicates that the function should dispose all of the driver-allocated buffers assigned to this list in addition to the buffer list. If **freeResources** is zero, the function disposes only the buffer list.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgGetBufferElement

## Format

```
rval = imgGetBufferElement(BUFLIST_ID bid, uInt32 element, uInt32 itemType,
                           void* itemValue)
```

## Purpose

Gets an element of a specific type from a buffer list.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **bid** | BUFLIST_ID | input | valid BUFLIST_ID |
| **element** | uInt32 | input | element number the buffer list |
| **itemType** | uInt32 | input | element item to get |
| **itemValue** | void* | input | new value for item |
| **rval** | Int32 | output | status |

## Parameter Discussion

**bid** is a valid BUFLIST_ID variable.

**element** is the element number of the buffer list item to examine.

**itemType** passes a valid buffer list element type as specified by the constants:

IMG_BUFF_ADDRESS
IMG_BUFF_CHANNEL
IMG_BUFF_COMMAND
IMG_BUFF_SIZE
IMG_BUFF_SKIPCOUNT

**itemValue** passes a pointer to an area of memory reserved for the return type (32 bits).

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

**Note** See *Constants* in Appendix A, *Attributes and Constants*, for valid element and command types.

# imgSessionClearBuffer

## Format

```
rval = imgSessionClearBuffer(SESSION_ID sid, uInt32 buf_num,
                             uInt8 pixel_value)
```

## Purpose

Clears a session's image data to the specified pixel value.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **buf_num** | uInt32 | input | element number of the buffer to clear |
| **pixel_value** | uInt8 | input | pixel value to set data to |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**buf_num** indicates a valid buffer list element number.

**pixel_value** indicates a pixel value to set all the buffer data with.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSetBufferElement

## Format

```
rval = imgSetBufferElement(BUFLIST_ID bid, uInt32 element, uInt32 itemType,
                           uInt32 itemValue)
```

## Purpose

Sets a buffer list element of a given type to a specific value.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **bid** | BUFLIST_ID | input | a valid BUFLIST_ID |
| **element** | uInt32 | input | element number the buffer list |
| **itemType** | uInt32 | input | element item to set |
| **itemValue** | uInt32 | input | new value for item |
| **rval** | Int32 | output | status |

## Parameter Discussion

**bid** is a valid BUFLIST_ID variable.

**element** is the element number of the buffer list item to modify.

**itemType** describes the parameter of the element to set as specified by the constants:

IMG_BUFF_ADDRESS
IMG_BUFF_CHANNEL
IMG_BUFF_COMMAND
IMG_BUFF_NUMBUFS
IMG_BUFF_SIZE
IMG_BUFF_SKIPCOUNT
IMG_BUFF_TRIGGER

**itemValue** indicates the value of the element type to set. Use the following constants to specify the IMG_BUFF_COMMAND itemType:

IMG_CMD_LOOP
IMG_CMD_NEXT
IMG_CMD_PASS
IMG_CMD_STOP

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

**Note** See *Constants* in Appendix A, *Attributes and Constants*, for valid element and command types.

# Interface Functions

Interface functions include `imgInterfaceQueryNames` and `imgInterfaceReset`.

Use these functions to perform operations specific to an interface. All interface functions require a valid INTERFACE_ID.

Interface functions operate on a board-wide basis. When you make a call to an interface function, it affects all sessions connected to that interface in all processes. Interface function changes are global and must be done with care.

# imgInterfaceQueryNames

## Format

```
rval = imgInterfaceQueryNames(uInt32 index, Int8* queryName)
```

## Purpose

Returns the interface name identified by the **index** parameter. To obtain a list of all the available interface names, call this function repeatedly until the function returns an error. Make the first call with the **index** parameter initialized to zero. Each successive call increments the **index** parameter by one.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **index** | uInt32 | input | interface number to obtain |
| **queryName** | Int8* | input | pointer to a character array of no less than 255 bytes |
| **rval** | Int32 | output | status |

## Parameter Discussion

**index** is the interface number to obtain.

**queryNames** is a pointer to a character array to receive the interface name.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgInterfaceReset

## Format

```
rval = imgInterfaceReset(INTERFACE_ID ifid)
```

## Purpose

Performs a hardware reset on the interface type and returns a status, either good or bad. This function sets the hardware associated with the interface to its default state.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **ifid** | INTERFACE_ID | output | interface ID |
| **rval** | Int32 | output | status |

## Parameter Discussion

**ifid** is a valid INTERFACE_ID variable.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# Utility Functions

Utility functions include `imgPlot`, `imgPlotDC`, `imgSessionSaveBufferEx`, and `imgShowError`.

Use these functions to display an image in a window, save an image to a file, or to get detailed error information.

# imgPlot

## Format

```
rval = imgPlot(GUIHNDL window, void* buffer, uInt32 leftBufOffset,
                          uInt32 topBufOffset,uInt32 xsize, uInt32 ysize,
                          uInt32 xpos, uInt32 ypos, uInt32 flags)
```

## Purpose

Plots a buffer to a window given a native Windows handle. This function is an easy way to display a buffer after it is acquired.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **window** | GUIHNDL | input | window handle |
| **buffer** | void* | input | pointer to video data |
| **leftBufOffset** | uInt32 | input | x-offset into the buffer to start plotting |
| **topBufOffset** | uInt32 | input | y-offset into the buffer to start plotting |
| **xsize** | uInt32 | input | width of the image |
| **ysize** | uInt32 | input | height of the image |
| **xpos** | uInt32 | input | x-position to start plotting in the window |
| **ypos** | uInt32 | input | y-position to start plotting in the window |
| **flags** | uInt32 | input | set display property |
| **rval** | Int32 | output | status |

## Parameter Discussion

**window** is a native Windows handle designating the window in which to plot.

**buffer** is a pointer to an area of memory containing a video frame buffer.

**leftBufOffset** is the left offset into the buffer to start plotting.

**topBufOffset** is the top offset into the buffer to start plotting.

**xsize** is the pixel width of the image.

**ysize** is the number of scanlines (pixel height) in the image.

**xpos** is the left position to start plotting in the window.

**ypos** is the top position to start plotting in the window.

**flags** sets the display property. **flags** is used with the following constants:

| | |
|---|---|
| IMGPLOT_INVERT | IMGPLOT_MONO_10 |
| IMGPLOT_COLOR_HSL32 | IMGPLOT_MONO_12 |
| IMGPLOT_COLOR_RGB32 | IMGPLOT_MONO_14 |
| IMGPLOT_MONO_8 | IMGPLOT_MONO_16 |

Use IMGPLOT_COLOR_HSL32 and IMGPLOT_COLOR_RGB32 constants to display a color image.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgPlotDC

## Format

```
rval = imgPlot(GUIHNDL dc, void* buffer, uInt32 leftBufOffset,
                           uInt32 topBufOffset,uInt32 xsize, uInt32 ysize,
                           uInt32 xpos, uInt32 ypos, uInt32 flags)
```

## Purpose

Plots a buffer to a device context given a device context handle.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **dc** | GUIHNDL | input | device context |
| **buffer** | void* | input | pointer to video data |
| **leftBufOffset** | uInt32 | input | x-offset into the buffer to start plotting |
| **topBufOffset** | uInt32 | input | y-offset into the buffer to start plotting |
| **xsize** | uInt32 | input | width of the image |
| **ysize** | uInt32 | input | height of the image |
| **xpos** | uInt32 | input | x-position to start plotting in the window |
| **ypos** | uInt32 | input | y-position to start plotting in the window |
| **flags** | uInt32 | input | set display property |
| **rval** | Int32 | output | status |

## Parameter Discussion

**dc** is a native Windows device context in which to draw.

**buffer** is a pointer to an area of memory containing a video frame buffer.

**leftBufOffset** is the left offset into the buffer to start plotting.

**topBufOffset** is the top offset into the buffer to start plotting.

**xsize** is the pixel width of the image.

**ysize** is the number of scanlines (pixel height) in the image.

**xpos** is the left position to start plotting in the window.

**ypos** is the top position to start plotting in the window.

**flags** sets the display property. **flags** is used with the following constants:

| | |
|---|---|
| IMGPLOT_INVERT | IMGPLOT_MONO_10 |
| IMGPLOT_COLOR_HSL32 | IMGPLOT_MONO_12 |
| IMGPLOT_COLOR_RGB32 | IMGPLOT_MONO_14 |
| IMGPLOT_MONO_8 | IMGPLOT_MONO_16 |

Use IMGPLOT_COLOR_HSL32 and IMGPLOT_COLOR_RGB32 constants to display a color image.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgSessionSaveBufferEx

## Format

```
rval = imgSessionSaveBufferEx(SESSION_ID sid, void* buffer, Int8* file_name)
```

## Purpose

Saves a buffer of a session to disk in bitmap, TIFF, or PNG format.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **sid** | SESSION_ID | input | session ID |
| **buffer** | void* | input | buffer to save |
| **file_name** | Int8* | input | NULL terminated string describing a file name (optional path) |
| **rval** | Int32 | output | status |

## Parameter Discussion

**sid** is a valid SESSION_ID variable.

**buffer** is a pointer to an image buffer to save.

**Note** The function takes information such as bits per pixel and buffer size from the current session settings. Therefore, this buffer must be associated with the current session.

**file_name** indicates a file name used to save the image. If the filename has the extension `.bmp`, the function saves the image as a bitmap file. Bitmap files support 8-bit monochrome and 32-bit RGB color images. If the filename extension is `.tif`, the function saves the image as a TIFF file. TIFF files support 8-bit monochrome and 32-bit RGB color images. If the filename extension is `.png`, the function saves the image as an uncompressed portable network graphics (PNG) file. PNG files support all image types.

**Note** If you are saving a 10-, 12-, 14-, or 16-bit monochrome image, you should use the PNG file format.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# imgShowError

## Format

```
rval = imgShowError(IMG_ERR error, char* text)
```

## Purpose

Returns a NULL terminated string describing the error code.

## Parameters

| Name | Type | Direction | Description |
|---|---|---|---|
| **error** | IMG_ERR | input | a valid error code |
| **text** | char* | input | pointer to a character array of no less than 255 bytes |
| **rval** | Int32 | output | status |

## Parameter Discussion

**error** is a valid NI-IMAQ error code.

**text** is a pointer to an area of memory reserved for an error string.

## Return Value

**rval** returns 0 on success. On failure, this function returns an error code. For more error information, call `imgShowError`.

# A

# Attributes and Constants

This appendix describes the attributes and constants used by NI-IMAQ.

## NI-IMAQ Attributes

Attributes describe a specific property of a session or interface. A summary of NI-IMAQ attributes is listed in Table A-1.

*Attribute* describes the constant name of the attribute. The *type* is the attribute's data type. *I/S* describes whether the get/set attribute function requires an INTERFACE_ID (I) or SESSION_ID (S) type parameter. *Imd.* describes whether the effect of setting the attribute is immediate (Yes), or whether it requires a subsequent `imgSessionConfigure` to take effect (No). *R/W* describes whether the attribute is read only (R), write only (W), or both (R/W). *Description* describes what values the attribute can take and the effect the setting of the attribute has or what values are returned.

**Table A-1.** Attribute Summary

| Attribute | Type | I/S | Imd. | R/W | Description |
|---|---|---|---|---|---|
| IMG_ATTR_ACQ_IN_PROGRESS | uInt32 | S | Yes | R | Returns TRUE if an acquisition is in progress on the camera associated with this session. |
| IMG_ATTR_ACQUIRE_FIELD | uInt32 | S | Yes | R/W | Sets the field acquired when IMG_ATTR_FRAME_FIELD is set to FIELD_MODE. Possible values are:<br>IMG_ACQUIRE_ODD<br>IMG_ACQUIRE_EVEN<br>IMG_ACQUIRE_ALL<br>The IMAQ PCI/PXI-1408 supports only IMG_ACQUIRE_ALL. |
| IMG_ATTR_ACQWINDOW_HEIGHT | uInt32 | S | No | R/W | Gets/sets the acquisition window height of the camera/channel associated with this session. |
| IMG_ATTR_ACQWINDOW_LEFT | uInt32 | S | No | R/W | Gets/sets the acquisition window left of the camera/channel associated with this session. |
| IMG_ATTR_ACQWINDOW_TOP | uInt32 | S | No | R/W | Gets/sets the acquisition window top of the camera/channel associated with this session. |
| IMG_ATTR_ACQWINDOW_WIDTH | uInt32 | S | No | R/W | Gets/sets the acquisition window width of the camera/channel associated with this session. |
| IMG_ATTR_BITSPERPIXEL | uInt32 | S | Yes | R | Returns the bits per pixel value of the camera/channel associated with this session. |

**Table A-1.** Attribute Summary (Continued)

| Attribute | Type | I/S | Imd. | R/W | Description |
|---|---|---|---|---|---|
| IMG_ATTR_BLACK_REF_VOLT | double* | S | Yes | R/W | Sets the black reference value, in volts, of the channel associated with this session. Values are 0 V to 1.26 V. |
| IMG_ATTR_BYTESPERPIXEL | uInt32 | S | Yes | R | Returns the bytes per pixel value of the camera/channel associated with this session. |
| IMG_ATTR_CHROMA_FILTER | uInt32 | S | Yes | R/W | Gets/sets the antichrominance filter to be used. Values are:<br>IMG_FILTER_NONE<br>IMG_FILTER_NTSC<br>IMG_FILTER_PAL |
| IMG_ATTR_CHANNEL | uInt32 | S | No | R/W | Programs the current channel selected on the interface (0-3); valid on the IMAQ PCI/PXI-1408 only. |
| IMG_ATTR_CLOCK_FREQ | uInt32 | S | Yes | R | Returns the maximum possible clock frequency of the board; not valid on the IMAQ PCI/PXI-1408. |
| IMG_ATTR_COLOR | uInt32 | I | Yes | R | Returns TRUE if the interface board is color-capable. |
| IMG_ATTR_COLOR_AVG_ COUNT | uInt32 | S | Yes | R/W | Gets/sets the number of color images to be acquired and averaged for one output image (1–128). Default value is 1. |
| IMG_ATTR_COLOR_ BRIGHTNESS | double* | S | No | R/W | Adjusts the brightness of an image—the amount of white light added to or subtracted from each image pixel. The range is –50 to +50 IRE (percentage of the white level) in steps of 1, with a default of 0 IRE. For information on using this function with the IMAQ PCI/PXI-1411, see Appendix C, *Calibration Information for the IMAQ PCI/PXI-1411*, in the *NI-IMAQ Function Reference Manual*. |
| IMG_ATTR_COLOR_CHROMA_ BANDWIDTH | uInt32 | S | No | R/W | Specifies the resulting bandwidth of the chroma information of the image. Values are:<br>IMG_COLOR_CHROMA_BANDWIDTH_HIGH<br>IMG_COLOR_CHROMA_BANDWIDTH_LOW |
| IMG_ATTR_COLOR_CHROMA_ COMB | uInt32 | S | No | R/W | Selects the type of comb filter used in the chroma path. Values are:<br>IMG_COLOR_COMB_OFF<br>IMG_COLOR_COMB_1LINE<br>IMG_COLOR_COMB_2LINES |
| IMG_ATTR_COLOR_CHROMA_ PHASE | double* | S | No | R/W | Specifies the value of a correction angle that can be applied to the chroma vector (adjustment of tint). This attribute is only active when an NTSC camera is used. The range is –180 to +180° to a default of 0°. |
| IMG_ATTR_COLOR_CHROMA_ PROCESS | uInt32 | S | No | R/W | Specifies the processing applied to the chroma signal. Values are:<br>IMG_COLOR_CHROMA_PROCESS_ALWAYS_ OFF<br>IMG_COLOR_CHROMA_PROCESS_ALWAYS_ ON<br>IMG_COLOR_CHROMA_PROCESS_ AUTODETECT |

**Table A-1.** Attribute Summary (Continued)

| Attribute | Type | I/S | Imd. | R/W | Description |
|---|---|---|---|---|---|
| IMG_ATTR_COLOR_CHROMA_TRAP | uInt32 | S | No | R/W | Enables the chroma trap filter in the luma signal path. Should always be disabled in S-Video / Y/C mode. Values are:<br>FALSE—Chroma trap filter disabled (default in S-Video/ Y/C mode)<br>TRUE—Chroma trap filter enabled, if the chroma trap filter is needed in composite mode |
| IMG_ATTR_COLOR_CONTRAST | double* | S | No | R/W | Adjusts the contrast of the image. The value is a scaling factor applied to every pixel. The contrast adjustment is centered around the median pixel value. The range is 0.4 to 1.5 with a default of 1.00<br>For information on using this function with the IMAQ PCI/PXI-1411, see Appendix C, *Calibration Information for the IMAQ PCI/PXI-1411*, in the *NI-IMAQ Function Reference Manual*. |
| IMG_ATTR_COLOR_GAIN_BLUE | double* | S | No | R/W | Gets/sets the gain applied to the blue color plane of the RGB image. This gain also affects the blue data used to calculate the hue, saturation, and luminance planes. The range is 0.8 to 1.2. Default value is 1.0. |
| IMG_ATTR_COLOR_GAIN_GREEN | double* | S | No | R/W | Gets/sets the gain applied to the green color plane of the RGB image. This gain also affects the green data used to calculate the hue, saturation, and luminance planes. The range is 0.8 to 1.2. Default value is 1.0. |
| IMG_ATTR_COLOR_GAIN_RED | double* | S | No | R/W | Gets/sets the gain applied to the red color plane of the RGB image. This gain also affects the red data used to calculate the hue, saturation, and luminance planes. The range is 0.8 to 1.2. Default value is 1.0. |
| IMG_ATTR_COLOR_HIGH_REF_VOLT | double* | S | Yes | R/W | Gets/sets the hardware black reference of your IMAQ device when StillColor is selected. Values are 0 V to 1.26 V. |
| IMG_ATTR_COLOR_HSL_CORING_LEVEL | uInt32 | S | No | R/W | Sets the HSL coring level when Image Representation is set to HSL. In HSL mode, on any image pixel, if the saturation value (S) of the pixel is lower than the specified value, then the Hue value (H) of the pixel is set to zero. The range is 0 to 255 lsb, with a default of 0 lsb. |
| IMG_ATTR_COLOR_HUE_OFFS_ANGLE | double* | S | No | R/W | Rotates the Hue plane with a specified offset angle. The hue value of a pixel is defined as an angle in the normal color plane. You can offset this angle to move the discontinuity point (at 0 modulo 360°) to another angle value. The range is –180 to +180° with a default of 0°. |
| IMG_ATTR_COLOR_HUE_REPLACE_VALUE | uInt32 | S | No | R/W | Gets/sets the value used to replace the hue when it is below IMG_ATTR_COLOR_HSL_CORING_LEVEL. |

**Table A-1.** Attribute Summary (Continued)

| Attribute | Type | I/S | Imd. | R/W | Description |
|---|---|---|---|---|---|
| IMG_ATTR_COLOR_IMAGE_REP | uInt32 | S | No | R/W | Specifies the type of image data returned when a color image is acquired. Values are:<br>IMG_COLOR_REP_RGB32<br>IMG_COLOR_REP_RED8<br>IMG_COLOR_REP_GREEN8<br>IMG_COLOR_REP_BLUE8<br>IMG_COLOR_REP_LUM8<br>IMG_COLOR_REP_HUE8<br>IMG_COLOR_REP_SAT8<br>IMG_COLOR_REP_INT8<br>IMG_COLOR_REP_LUM16<br>IMG_COLOR_REP_HUE16<br>IMG_COLOR_REP_SAT16<br>IMG_COLOR_REP_INT16<br>IMG_COLOR_REP_RGB24<br>IMG_COLOR_REP_RGB16<br>IMG_COLOR_REP_HSL32<br>IMG_COLOR_REP_HSI32 |
| IMG_ATTR_COLOR_LOW_REF_VOLT | double* | S | Yes | R/W | Gets/sets the hardware white reference of your IMAQ device when StillColor is selected. Values are 0 V to 1.26 V. |
| IMG_ATTR_COLOR_LUMA_BANDWIDTH | uInt32 | S | No | R/W | Selects different bandwidths for the luminance signal. The four valid options are:<br>IMG_COLOR_LUMA_BANDWIDTH_FULL<br>IMG_COLOR_LUMA_BANDWIDTH_HIGH<br>IMG_COLOR_LUMA_BANDWIDTH_MEDIUM<br>IMG_COLOR_LUMA_BANDWIDTH_LOW |
| IMG_ATTR_COLOR_LUMA_COMB | uInt32 | S | No | R/W | Selects the type of comb filter used in the luma path. Values are:<br>IMG_COLOR_COMB_OFF<br>IMG_COLOR_COMB_1LINE<br>IMG_COLOR_COMB_2LINES |
| IMG_ATTR_COLOR_MODE | uInt32 | S | Yes | R/W | Sets or gets the color acquisition mode. Values are:<br>IMG_COLOR_MODE_DISABLED<br>IMG_COLOR_MODE_RGB<br>IMG_COLOR_MODE_COMPOSITE_STLC |
| IMG_ATTR_COLOR_NTSC_SETUP_ ENABLE | uInt32 | S | No | R/W | Enables the setup correction of 7.5 IRE in NTSC mode. A standard NTSC signal has a setup level that moves up the black level 7.5% of the white level (or 7.5 IRE). You can perform correction for this setup during acquisition by enabling this attribute. FALSE disables the setup correction (default in PAL or CCIR mode) and TRUE enables the setup correction (default in NTSC or RS-170 mode). |
| IMG_ATTR_COLOR_NTSC_SETUP_VALUE | double* | S | No | R/W | Gets/sets the NTSC setup compensation value. The unit is IRE (percentage of white level). Default value is 7.5%. (StillColor NTSC only) |
| IMG_ATTR_COLOR_PEAKING_ENABLE | uInt32 | S | No | R/W | Enables the peaking filter in the luma path. Values are:<br>FALSE—Peaking filter disabled (default)<br>TRUE—Peaking filter enabled |

**Table A-1.** Attribute Summary (Continued)

| Attribute | Type | I/S | Imd. | R/W | Description |
|---|---|---|---|---|---|
| IMG_ATTR_COLOR_RGB_ CORING_LEVEL | uInt32 | S | No | R/W | Selects between four different coring levels. On any image pixel, if the color saturation of the pixel is lower than the specified value, then the saturation is set to zero, resulting in a monochrome pixel. Values are:<br>IMG_COLOR_RGB_CORING_LEVEL_ NOCORING<br>IMG_COLOR_RGB_CORING_LEVEL_C1<br>IMG_COLOR_RGB_CORING_LEVEL_C3<br>IMG_COLOR_RGB_CORING_LEVEL_C7 |
| IMG_ATTR_COLOR_ SATURATION | double* | S | No | R/W | Adjusts the saturation of the image—a factor multiplied to the chroma information of the image. The range is 0.5 to 1.5 with a default of 1.00.<br>For information on using this function with the IMAQ PCI/PXI-1411, see Appendix C, *Calibration Information for the IMAQ PCI/PXI-1411*, in the *NI-IMAQ Function Reference Manual*. |
| IMG_ATTR_COLOR_SW_ CHROMA_FILTER | uInt32 | S | Yes | R/W | Gets/sets the software filter to clean the chroma signal (StillColor NTSC only):<br>Disabled (0)<br>Enabled (1) |
| IMG_ATTR_COLOR_SW_POST_ GAIN | double* | S | No | R/W | Gets/sets a software gain value used in StillColor acquisition. |
| IMG_ATTR_COLOR_TINT | double* | S | Yes | R/W | Gets/sets the tint of your image. Tint is specified in degrees and corresponds to the rotation of the UV color plane. Default value is 0. |
| IMG_ATTR_FRAME_COUNT | uInt32 | S | Yes | R | Returns the number of frames acquired since the start of an acquisition. |
| IMG_ATTR_FRAME_FIELD | uInt32 | S | Yes | R/W | Sets/gets the current mode of the session.<br>IMG_FIELD_MODE sets field mode acquisition. In this mode, use the IMG_ATTR_ACQUIRE_FIELD attribute to set the field acquired.<br>IMG_FRAME_MODE sets frame mode acquisition. In this mode, use the IMG_ATTR_START_FIELD attribute to set the field to acquire first. |
| IMG_ATTR_FRAMEWAIT_MSEC | uInt32 | S | No | R/W | Gets/sets the timeout value for a frame. Values are:<br>IMG_FRAMETIME_STANDARD<br>IMG_FRAMETIME_1SECOND<br>IMG_FRAMETIME_2SECONDS<br>IMG_FRAMETIME_5SECONDS<br>IMG_FRAMETIME_10SECONDS<br>IMG_FRAMETIME_1MINUTE<br>IMG_FRAMETIME_2MINUTES<br>IMG_FRAMETIME_5MINUTES<br>IMG_FRAMETIME_10MINUTES |
| IMG_ATTR_FREE_BUFFERS | uInt32 | S | Yes | R | Returns the number of reserved driver buffers currently left. |
| IMG_ATTR_GETSERIAL | uInt32 | I | Yes | R | Returns the serial number of the board. |
| IMG_ATTR_HASRAM | uInt32 | I | Yes | R | Returns TRUE if the interface board has onboard memory. |

**Table A-1.** Attribute Summary (Continued)

| Attribute | Type | I/S | Imd. | R/W | Description |
|---|---|---|---|---|---|
| IMG_ATTR_HORZ_ RESOLUTION | uInt32 | I | Yes | R | Returns the maximum horizontal resolution of the interface. |
| IMG_ATTR_HSCALE | uInt32 | S | No | R/W | Gets/sets the horizontal hardware scaling factor for the channel associated with this session. Values are:<br>IMG_SCALE_NONE<br>IMG_SCALE_DIV2<br>IMG_SCALE_DIV4<br>IMG_SCALE_DIV8 |
| IMG_ATTR_INTERFACE_TYPE | uInt32 | I | Yes | R | Returns the type of the interface in hex: 0x1408 for the IMAQ PCI-1408, 0x1424 for the IMAQ PCI-1424. |
| IMG_ATTR_LAST_VALID_ BUFFER | uInt32 | S | Yes | R | Returns a buffer element number of the last received frame buffer. |
| IMG_ATTR_LAST_VALID_ FRAME | uInt32 | S | Yes | R | Returns the cumulative buffer index (frame#). |
| IMG_ATTR_LINE_COUNT | uInt32 | S | Yes | R | Returns the current line count of the frame being acquired. |
| IMG_ATTR_LINESCAN | uInt32 | I | Yes | R | Returns TRUE if the camera attached to the interface is a linescan camera. |
| IMG_ATTR_LOST_FRAMES | uInt32 | S | Yes | R | For the IMAQ 1407 and IMAQ 1408—Returns the total number of lost frames in a continuous acquisition.<br>For the IMAQ 1409, 1411, 1422, 1424, 1428—Value increments once for every occurence of losing frames. |
| IMG_ATTR_LUT | uInt32 | S | Yes | R/W | Programs the lookup table for the given interface. Pass a constant to indicate the LUT you want to use or you can pass a pointer to your own LUT. Constant values are:<br>IMG_LUT_NORMAL<br>IMG_LUT_INVERSE<br>IMG_LUT_LOG<br>IMG_LUT_INVERSE_LOG<br>IMG_LUT_BINARY<br>IMG_LUT_INVERSE_BINARY |
| IMG_ATTR_MEM_LOCKED | uInt32 | S | Yes | R | Returns TRUE if the session's buffer list is locked in memory. |
| IMG_ATTR_PCLK_DETECT | uInt32 | S | Yes | R/W | Determines if NI-IMAQ checks for the existence of a pixel clock before starting an acquisition. The default value is TRUE. If your camera has a pixel clock less than 5 MHz, this detection may fail and you should set this attribute to FALSE. This attribute is valid only on the PCI-1424. |
| IMG_ATTR_PIXDEPTH | uInt32 | I | Yes | R | Returns the maximum pixel depth of the interface board in bytes. |
| IMG_ATTR_RAMSIZE | uInt32 | I | Yes | R | Returns the size of the RAM on the interface board. |
| IMG_ATTR_ROI_HEIGHT | uInt32 | S | No | R/W | Gets/sets the region of interest height of the camera/channel associated with this session. |

**Table A-1.** Attribute Summary (Continued)

| Attribute | Type | I/S | Imd. | R/W | Description |
|---|---|---|---|---|---|
| IMG_ATTR_ROI_LEFT | uInt32 | S | Yes | R/W | Gets/sets the region of interest left of the camera/channel associated with this session. |
| IMG_ATTR_ROI_TOP | uInt32 | S | Yes | R/W | Gets/sets the region of interest top of the camera/channel associated with this session. |
| IMG_ATTR_ROI_WIDTH | uInt32 | S | No | R/W | Gets/sets the region of interest width of the camera/channel associated with this session. |
| IMG_ATTR_ROWPIXELS | uInt32 | S | No | R/W | Gets/sets the number of pixels in a row of an image. |
| IMG_ATTR_START_FIELD | uInt32 | S | No | R/W | Sets/gets the start field setting of the camera when IMG_ATTR_FRAME_FIELD is set to FRAME_MODE. Possible values are:<br>IMG_FIELD_ODD<br>IMG_FIELD_EVEN |
| IMG_ATTR_VERT_RESOLUTION | uInt32 | I | Yes | R | Returns the maximum vertical resolution of the interface. |
| IMG_ATTR_VHA_MODE | uInt32 | S | No | R/W | Enables VHA Mode. Values are:<br>FALSE–VHA Mode disabled<br>TRUE–VHA Mode enabled<br>See Appendix B in the *NI-IMAQ User Manual* for more information. |
| IMG_ATTR_VSCALE | uInt32 | S | No | R | Gets/sets the vertical hardware scaling factor for the channel associated with this session. Values are:<br>IMG_SCALE_NONE<br>IMG_SCALE_DIV2<br>IMG_SCALE_DIV4<br>IMG_SCALE_DIV8 |
| IMG_ATTR_WHITE_REF_VOLT | double* | S | Yes | R/W | Sets the white reference value, in volts, of the channel associated with this session. Values are 0 V to 1.26 V. |
| IMG_ATTR_XOFF_BUFFER | uInt32 | S | No | R/W | Gets/sets the buffer x-left offset for image displacement. Use this attribute to acquire an image into a private buffer at a different location other than the top-left corner. You must use a private buffer when using this attribute. |
| IMG_ATTR_YOFF_BUFFER | uInt32 | S | No | R/W | Gets/sets the buffer y-line offset for image displacement. Use this attribute to acquire an image into a private buffer at a different location other than the top-left corner. You must use a private buffer when using this attribute. |

# Constants

Constants help clearly define specific function parameter values. These constants are included in your `niimaq.h` header file. Use these constants when coding the `imgGetAttribute` and `imgSetAttribute` functions when required.

Table A-2 lists the constant name, the function to which the constant applies, and a general description.

**Table A-2.** Constants Summary

| Constant | Use With | Description |
|---|---|---|
| IMG_ACQUIRE_ALL | `imgGetAttribute`<br>`imgSetAttribute` | Acquire all fields. |
| IMG_ACQUIRE_EVEN | `imgGetAttribute`<br>`imgSetAttribute` | Acquire only even fields. |
| IMG_ACQUIRE_ODD | `imgGetAttribute`<br>`imgSetAttribute` | Acquire only odd fields. |
| IMG_AQ_DONE | `imgSessionWaitSignal`<br>`imgSessionWaitSignalAsync`<br>`imgPulseCreate` | Asserted at the end of an acquisition when the last piece of data has been transferred to memory. |
| IMG_AQ_IN_PROGRESS | `imgSessionWaitSignal`<br>`imgSessionWaitSignalAsync`<br>`imgPulseCreate` | Asserted when the board initiates an acquisition either through a software- or hardware-triggered start. |
| IMG_BOARD_INTERFACE | `imgGetAttribute` | Specifies the type of interface is a plug-in board. |
| IMG_BUF_COMPLETE | `imgSessionWaitSignal`<br>`imgSessionWaitSignalAsync`<br>`imgPulseCreate` | Asserted when an image buffer has been transferred to memory and is available for image processing. |
| IMG_BUFF_ACTUAL_HEIGHT | `imgGetBufferElement` | Returns the actual height in lines of a buffer acquired in VHA mode. |
| IMG_BUFF_ADDRESS | `imgGetBufferElement`<br>`imgSetBufferElement` | Specifies the buffer address portion of a buffer list element. |
| IMG_BUFF_CHANNEL | `imgGetBufferElement`<br>`imgSetBufferElement` | Specifies the channel from which to acquire an image. |

**Table A-2.** Constants Summary (Continued)

| Constant | Use With | Description |
|---|---|---|
| IMG_BUFF_COMMAND | `imgGetBufferElement`<br>`imgSetBufferElement` | Specifies the command portion of a buffer list element. |
| IMG_BUFF_NUMBUFS | `imgGetBufferElement`<br>`imgSetBufferElement` | Specifies the number of items in a buffer list. |
| IMG_BUFF_SIZE | `imgGetBufferElement`<br>`imgSetBufferElement` | Specifies the size portion of a buffer list element (the buffer's size). Required for user-allocated buffers. |
| IMG_BUFF_SKIPCOUNT | `imgGetBufferElement`<br>`imgSetBufferElement` | Specifies the skip count portion of a buffer list element. |
| IMG_CMD_LOOP | `imgGetBufferElement`<br>`imgSetBufferElement`<br>`with IMG_BUFF_COMMAND constant` | Specifies a buffer list command of LOOP. Used as the command for the last buffer element, this constant causes an acquisition to perform a continuous type of acquisition such as a ring. |
| IMG_CMD_NEXT | `imgGetBufferElement`<br>`imgSetBufferElement`<br>`(with IMG_BUFF_COMMAND constant)` | Specifies a buffer list command of NEXT. This constant causes an acquisition to take place on the buffer and to proceed to the next buffer list element. |
| IMG_CMD_PASS | `imgGetBufferElement`<br>`imgSetBufferElement`<br>`(with IMG_BUFF_COMMAND constant)` | Specifies a buffer list command of PASS. Any buffer list element with this command is ignored. Use to reserved space for fast configuration. |
| IMG_CMD_STOP | `imgGetBufferElement`<br>`imgSetBufferElement`<br>`(with IMG_BUFF_COMMAND constant)` | Specifies a buffer list command of STOP. Used as the command for the last buffer element, this constant causes an acquisition to perform a one-shot acquisition such as a sequence. |
| IMG_COLOR_CHROMA_BANDWIDTH_HIGH | `imgSetAttribute`<br>`imgGetAttribute` | Highest bandwidth (default). |

**Table A-2.** Constants Summary (Continued)

| Constant | Use With | Description |
|---|---|---|
| IMG_COLOR_CHROMA_BANDWIDTH_LOW | imgSetAttribute<br>imgGetAttribute | Lowest bandwidth. |
| IMG_COLOR_CHROMA_PROCESS_ALWAYS_OFF | imgSetAttribute<br>imgGetAttribute | Use when you use a monochrome camera (default for CCIR or RS-170). |
| IMG_COLOR_CHROMA_PROCESS_ALWAYS_ON | imgSetAttribute<br>imgGetAttribute | Use when you use a color camera (default for NTSC or PAL). |
| IMG_COLOR_CHROMA_PROCESS_AUTODETECT | imgSetAttribute<br>imgGetAttribute | Can be used if the camera type (monochrome or color) is unknown. |
| IMG_COLOR_COMB_1LINE | imgSetAttribute<br>imgGetAttribute | Comb filtering using one delayed line. |
| IMG_COLOR_COMB_2LINES | imgSetAttribute<br>imgGetAttribute | Comb filtering using two delayed lines. |
| IMG_COLOR_COMB_OFF | imgSetAttribute<br>imgGetAttribute | Comb filter disabled (default in S-Video (Y/C) mode). |
| IMG_COLOR_LUMA_BANDWIDTH_FULL | imgSetAttribute<br>imgGetAttribute | All filters including decimation filter disabled. Default value in CCIR or RS-170 mode. |
| IMG_COLOR_LUMA_BANDWIDTH_HIGH | imgSetAttribute<br>imgGetAttribute | Highest available bandwidth with decimation filter enabled. Default value for PAL or NTSC mode. |
| IMG_COLOR_LUMA_BANDWIDTH_LOW | imgSetAttribute<br>imgGetAttribute | Decimation filter enabled, lowest bandwidth. |
| IMG_COLOR_LUMA_BANDWIDTH_MEDIUM | imgSetAttribute<br>imgGetAttribute | Decimation filter enabled, medium bandwidth. |
| IMG_COLOR_MODE_COMPOSITE_STLC | imgGetAttribute<br>imgSetAttribute | Specifies that color mode is StillColor Composite. |
| IMG_COLOR_MODE_DISABLED | imgGetAttribute<br>imgSetAttribute | Specifies that color mode is disabled. |
| IMG_COLOR_MODE_RGB | imgGetAttribute<br>imgSetAttribute | Specifies that color mode is StillColor RGB. |
| IMG_COLOR_REP_BLUE8 | imgGetAttribute<br>imgSetAttribute | Specifies 8-bit Blue color output. |

**Table A-2.** Constants Summary (Continued)

| Constant | Use With | Description |
|---|---|---|
| IMG_COLOR_REP_GREEN8 | imgGetAttribute<br>imgSetAttribute | Specifies 8-bit Green color output. |
| IMG_COLOR_REP_HSI32 | imgGetAttribute<br>imgSetAttribute | A color image encoded in 32 bits—8 bits unused and 8 bits each for the Hue, Saturation, and Intensity planes. |
| IMG_COLOR_REP_HSL32 | imgGetAttribute<br>imgSetAttribute | A color image encoded in 32 bits—8 bits unused and 8 bits each for the Hue, Saturation, and Luminance planes. |
| IMG_COLOR_REP_HUE16 | imgGetAttribute<br>imgSetAttribute | Specifies 16-bit Hue color output. |
| IMG_COLOR_REP_HUE8 | imgGetAttribute<br>imgSetAttribute | Specifies 8-bit Hue color output. |
| IMG_COLOR_REP_INT16 | imgGetAttribute<br>imgSetAttribute | Specifies 16-bit Intensity color output. |
| IMG_COLOR_REP_INT8 | imgGetAttribute<br>imgSetAttribute | Specifies 8-bit Intensity color output. |
| IMG_COLOR_REP_LUM16 | imgGetAttribute<br>imgSetAttribute | Specifies 16-bit Luminance color output. |
| IMG_COLOR_REP_LUM8 | imgGetAttribute<br>imgSetAttribute | Specifies 8-bit Luminance color output. |
| IMG_COLOR_REP_RED8 | imgGetAttribute<br>imgSetAttribute | Specifies 8-bit Red color output. |
| IMG_COLOR_REP_RGB16 | imgGetAttribute<br>imgSetAttribute | Specifies 16-bit RGB color output. |
| IMG_COLOR_REP_RGB24 | imgGetAttribute<br>imgSetAttribute | Specifies 24-bit RGB color output (default). |
| IMG_COLOR_REP_RGB32 | imgGetAttribute<br>imgSetAttribute | Specifies 32-bit RGB color output. |
| IMG_COLOR_REP_RGB48 | imgGetAttribute<br>imgSetAttribute | Specifies 48-bit RGB color output. |
| IMG_COLOR_REP_SAT16 | imgGetAttribute<br>imgSetAttribute | Specifies 16-bit Saturation color output. |
| IMG_COLOR_REP_SAT8 | imgGetAttribute<br>imgSetAttribute | Specifies 8-bit Saturation color output |

**Table A-2.** Constants Summary (Continued)

| Constant | Use With | Description |
|---|---|---|
| IMG_COLOR_RGB_CORING_LEVEL_NOCORING | imgSetAttribute<br>imgGetAttribute | The coring function is disabled. |
| IMG_COLOR_RGB_CORING_LEVEL_C1 | imgSetAttribute<br>imgGetAttribute | Coring activated for saturation equal or below 1 LSB. |
| IMG_COLOR_RGB_CORING_LEVEL_C3 | imgSetAttribute<br>imgGetAttribute | Coring activated for saturation equal or below 3 LSB. |
| IMG_COLOR_RGB_CORING_LEVEL_C7 | imgSetAttribute<br>imgGetAttribute | Coring activated for saturation equal or below 7 LSB. |
| IMG_CURRENT_BUFFER | imgSessionExamineBuffer | Specifies to examine current buffer in a live acquisition. Waits until vertical blank to return the buffer to you. |
| IMG_DEVICE_FRAME | imgCreateBuffer | Specifies the new buffer is created in onboard memory (not supported on the IMAQ PCI-1408). |
| IMG_EXT_RTSI0 | imgPulseCreate<br>imgSessionWaitSignal<br>imgSessionWaitSignalAsync<br>imgSessionTriggerConfigure<br>imgSessionTriggerDrive<br>imgSessionTriggerRead | Specifies RTSI line 0. |
| IMG_EXT_RTSI1 | imgSessionTriggerConfigure<br>imgSessionTriggerDrive<br>imgSessionTriggerRead | Specifies RTSI line 1. |
| IMG_EXT_RTSI2 | imgSessionTriggerConfigure<br>imgSessionTriggerDrive<br>imgSessionTriggerRead | Specifies RTSI line 2. |
| IMG_EXT_RTSI3 | imgSessionTriggerConfigure<br>imgSessionTriggerDrive<br>imgSessionTriggerRead | Specifies RTSI line 3. |
| IMG_EXT_RTSI4 | imgSessionTriggerConfigure<br>imgSessionTriggerDrive<br>imgSessionTriggerRead | Specifies RTSI line 4. |
| IMG_EXT_RTSI5 | imgSessionTriggerConfigure<br>imgSessionTriggerDrive<br>imgSessionTriggerRead | Specifies RTSI line 5. |

**Table A-2.** Constants Summary (Continued)

| Constant | Use With | Description |
|---|---|---|
| IMG_EXT_RTSI6 | imgSessionTriggerConfigure<br>imgSessionTriggerDrive<br>imgSessionTriggerRead | Specifies RTSI line 6. |
| IMG_EXT_TRIG0 | imgSessionTriggerConfigure<br>imgSessionTriggerDrive<br>imgSessionTriggerRead | Specifies the external trigger 0. |
| IMG_EXT_TRIG1 | imgSessionTriggerConfigure<br>imgSessionTriggerDrive<br>imgSessionTriggerRead | Specifies the external trigger 1. |
| IMG_EXT_TRIG2 | imgSessionTriggerConfigure<br>imgSessionTriggerDrive<br>imgSessionTriggerRead | Specifies the external trigger 2. |
| IMG_EXT_TRIG3 | imgSessionTriggerConfigure<br>imgSessionTriggerDrive<br>imgSessionTriggerRead | Specifies the external trigger 3. |
| IMG_FIELD_EVEN | imgGetAttribute<br>imgSetAttribute | Specifies the start field of the acquisition as even. |
| IMG_FIELD_MODE | imgGetAttribute<br>imgSetAttribute | Specifies the acquisition mode as field. |
| IMG_FIELD_ODD | imgGetAttribute<br>imgSetAttribute | Specifies the start field of the acquisition as odd. |
| IMG_FILTER_NONE | imgGetAttribute<br>imgSetAttribute | Specifies no video filter. |
| IMG_FILTER_NTSC | imgGetAttribute<br>imgSetAttribute | Specifies the video filter is NTSC. |
| IMG_FILTER_PAL | imgGetAttribute<br>imgSetAttribute | Specifies the video filter is PAL. |
| IMG_FRAME_DONE | imgSessionWaitSignal<br>imgSessionWaitSignalAsync<br>imgPulseCreate | Asserted at the end of acquisition into each image buffer. |
| IMG_FRAME_MODE | imgGetAttribute<br>imgSetAttribute | Specifies the acquisition mode as interlaced. |
| IMG_FRAME_START | imgSessionWaitSignal<br>imgSessionWaitSignalAsync<br>imgPulseCreate | Asserted at the start of acquisition into each image buffer. |
| IMG_FRAMETIME_10MINUTES | imgGetAttribute<br>imgSetAttribute | Specifies a frame timeout value of 10 min. |

**Table A-2.** Constants Summary (Continued)

| Constant | Use With | Description |
|---|---|---|
| IMG_FRAMETIME_10SECONDS | imgGetAttribute<br>imgSetAttribute | Specifies a frame timeout value of 10 s. |
| IMG_FRAMETIME_1MINUTE | imgGetAttribute<br>imgSetAttribute | Specifies a frame timeout value of 1 min. |
| IMG_FRAMETIME_1SECOND | imgGetAttribute<br>imgSetAttribute | Specifies a frame timeout value of 1 s. |
| IMG_FRAMETIME_2MINUTES | imgGetAttribute<br>imgSetAttribute | Specifies a frame timeout value of 2 min. |
| IMG_FRAMETIME_2SECONDS | imgGetAttribute<br>imgSetAttribute | Specifies a frame timeout value of 2 s. |
| IMG_FRAMETIME_5MINUTES | imgGetAttribute<br>imgSetAttribute | Specifies a frame timeout value of 5 min. |
| IMG_FRAMETIME_5SECONDS | imgGetAttribute<br>imgSetAttribute | Specifies a frame timeout value of 5 s. |
| IMG_FRAMETIME_STANDARD | imgGetAttribute<br>imgSetAttribute | Specifies a frame timeout value of 100 ms. |
| IMG_GAIN_0DB | imgGetAttribute<br>imgSetAttribute | Specifies the gain is +0 dB. |
| IMG_GAIN_3DB | imgGetAttribute<br>imgSetAttribute | Specifies the gain is +3 dB. |
| IMG_GAIN_6DB | imgGetAttribute<br>imgSetAttribute | Specifies the gain is +6 dB. |
| IMG_HOST_FRAME | imgCreateBuffer | Specifies the new buffer is created in host (computer) memory. |
| IMG_IMMEDIATE | imgPulseCreate | Causes the function to generate a pulse when the function is executed. |
| IMG_LAST_BUFFER | imgSessionExamineBuffer | Specifies to examine the last valid buffer in a live acquisition. |
| IMG_LUT_BINARY | imgGetAttribute<br>imgSetAttribute | Specifies a binary LUT, which converts the sampled data to a binary image of black and white. |
| IMG_LUT_INVERSE | imgGetAttribute<br>imgSetAttribute | Specifies an inverse LUT, which inverts the gray levels. |

**Table A-2.** Constants Summary (Continued)

| Constant | Use With | Description |
|---|---|---|
| IMG_LUT_INVERSE_BINARY | imgGetAttribute<br>imgSetAttribute | Specifies an inverse binary LUT, which converts the sampled data into a binary image of white and black. |
| IMG_LUT_INVERSE_LOG | imgGetAttribute<br>imgSetAttribute | Specifies an inverse log LUT, which converts the sampled data to a logarithmic form that produces greater contrast in the white region. |
| IMG_LUT_LOG | imgGetAttribute<br>imgSetAttribute | Specifies a log LUT, which converts the sampled data to a logarithmic form that produces greater contrast in the black region. |
| IMG_LUT_NORMAL | imgGetAttribute<br>imgSetAttribute | Specifies a normal LUT. |
| IMG_LUT_TYPE_DEFAULT | imgSessionSetUserLUT8bits<br>imgSessionSetUserLUT16bits | Default LUT used to initialize all LUTs. |
| IMG_LUT_TYPE_RED | imgSessionSetUserLUT8bits | Red channel LUT for RGB digital cameras or the PCI/PXI-1411. |
| IMG_LUT_TYPE_GREEN | imgSessionSetUserLUT8bits | Green channel LUT for RGB digital cameras or the PCI/PXI-1411. |
| IMG_LUT_TYPE_BLUE | imgSessionSetUserLUT8bits | Blue channel LUT for RGB digital cameras or the PCI/PXI-1411. |
| IMG_LUT_TYPE_TAP0 | imgSessionSetUserLUT8bits<br>imgSessionSetUserLUT16bits | Tap 0 LUT for digital boards. |
| IMG_LUT_TYPE_TAP1 | imgSessionSetUserLUT8bits<br>imgSessionSetUserLUT16bits | Tap 1 LUT for digital boards. |
| IMG_LUT_TYPE_TAP2 | imgSessionSetUserLUT8bits | Tap 2 LUT for digital boards. |
| IMG_LUT_TYPE_TAP3 | imgSessionSetUserLUT8bits | Tap 3 LUT for digital boards. |
| IMG_OLDEST_BUFFER | imgSessionExamineBuffer | Specifies the function to examine the oldest buffer in a live acquisition. |

**Table A-2.** Constants Summary (Continued)

| Constant | Use With | Description |
|---|---|---|
| IMG_OTHER_INTERFACE | imgGetAttribute | Specifies that the type of interface is not a plug-in board. |
| IMG_PULSE_POLAR_ACTIVEH | imgPulseCreate | Specifies the polarity of a pulse as active high. |
| IMG_PULSE_POLAR_ACTIVEL | imgPulseCreate | Specifies the polarity of a pulse as active low. |
| IMG_SCALE_DIV2 | imgGetAttribute<br>imgSetAttribute | Specifies scaling by divisions of 2. |
| IMG_SCALE_DIV4 | imgGetAttribute<br>imgSetAttribute | Specifies scaling by divisions of 4. |
| IMG_SCALE_DIV8 | imgGetAttribute<br>imgSetAttribute | Specifies scaling by divisions of 8. |
| IMG_SCALE_NONE | imgGetAttribute<br>imgSetAttribute | Specifies no scaling. |
| IMG_TRIG_ACTION_BUFFER | imgSessionTriggerDrive | Specifies each buffer is acquired based on a trigger. |
| IMG_TRIG_ACTION_BUFLIST | imgSessionTriggerDrive | Specifies acquisition of buffer list starts on a trigger. |
| IMG_TRIG_ACTION_CAPTURE | imgSessionTriggerDrive | Specifies trigger starts acquisition. |
| IMG_TRIG_ACTION_NONE | imgSessionTriggerDrive | Specifies triggering is disabled. |
| IMG_TRIG_DRIVE_AQ_DONE | imgSessionTriggerDrive | Specifies that the trigger line is driven on AQ_DONE. |
| IMG_TRIG_DRIVE_AQ_IN_PROGRESS | imgSessionTriggerDrive | Specifies that the trigger line is driven on AQ_IN_PROGRESS. |
| IMG_TRIG_DRIVE_ASSERTED | imgSessionTriggerDrive | Specifies to immediately drive the trigger line asserted. |
| IMG_TRIG_DRIVE_DISABLED | imgSessionTriggerDrive | Specifies that the trigger line is not driven. |
| IMG_TRIG_DRIVE_FRAME_DONE | imgSessionTriggerDrive | Specifies that the trigger line is driven with frame done. |

**Table A-2.** Constants Summary (Continued)

| Constant | Use With | Description |
|---|---|---|
| IMG_TRIG_DRIVE_FRAME_START | imgSessionTriggerDrive | Specifies that the trigger line is driven with frame start. |
| IMG_TRIG_DRIVE_HSYNC | imgSessionTriggerDrive | Specifies that the trigger line is driven on HSYNC. |
| IMG_TRIG_DRIVE_PIXEL_CLK | imgSessionTriggerDrive | Specifies that the trigger line is driven with pixel clock. |
| IMG_TRIG_DRIVE_UNASSERTED | imgSessionTriggerDrive | Specifies to immediately drive the trigger line unasserted. |
| IMG_TRIG_DRIVE_VSYNC | imgSessionTriggerDrive | Specifies that the trigger line is driven on VSYNC. |
| IMG_TRIG_NONE | imgGetAttribute<br>imgSetAttribute | Specifies no trigger for the session. |
| IMG_TRIG_POLAR_ACTIVEH | imgSessionTriggerConfigure<br>imgSessionTriggerDrive<br>imgSessionTriggerRead | Specifies the polarity of a trigger as active HIGH. |
| IMG_TRIG_POLAR_ACTIVEL | imgSessionTriggerConfigure<br>imgSessionTriggerDrive<br>imgSessionTriggerRead | Specifies the polarity of a trigger as active LOW. |
| IMG_TRIGMODE_NONE | imgGetAttribute<br>imgSetAttribute | Specifies a disabled trigger mode. Triggers do not cause an acquisition to occur. |
| IMG_TRIGMODE_NOREPEAT | imgGetAttribute<br>imgSetAttribute | Specifies a trigger mode of no-repeat. Enabled triggers cause an acquisition to occur on the buffers that have their buffer trigger attribute set. The sequence does not re-arm after the last buffer. |
| IMG_TRIGMODE_REPEAT | imgGetAttribute<br>imgSetAttribute | Specifies a trigger mode of repeat. Enabled triggers cause an acquisition to occur on the specified buffers, and the sequence automatically re-arms to acquire the sequence again after the last buffer. |

**Table A-2.** Constants Summary (Continued)

| Constant | Use With | Description |
|---|---|---|
| IMGPLOT_COLOR_RGB24 | imgPlot | Specifies a 24-bit color RGB image. |
| IMGPLOT_COLOR_RGB32 | imgPlot | Specifies a 32-bit color RGB image. |
| IMGPLOT_INVERT | imgPlot | Specifies to invert the image when plotted. |
| IMGPLOT_MONO_10 | imgPlot | Specifies a 10-bit monochrome image. |
| IMGPLOT_MONO_12 | imgPlot | Specifies a 12-bit monochrome image. |
| IMGPLOT_MONO_14 | imgPlot | Specifies a 14-bit monochrome image. |
| IMGPLOT_MONO_16 | imgPlot | Specifies a 16-bit monochrome image. |
| IMGPLOT_MONO_32 | imgPlot | Specifies a 32-bit monochrome image. |
| IMGPLOT_MONO_8 | imgPlot | Specifies an 8-bit monochrome image. |
| INTERFACE_NAME_SIZE | imgInterfaceQueryNames | Specifies the size of each array element in the interface names array. |
| PULSE_MODE_SINGLE | imgPulseCreate | Specifies a single pulse that generates one pulse on the assertion edge of the specified signal. |
| PULSE_MODE_SINGLE_REARM | imgPulseCreate | Specifies a pulse that generates a pulse on all assertion edges of the specified signal until imgPulseStop is called. |
| PULSE_MODE_TRAIN | imgPulseCreate | Specifies a continuous pulse train that generates a pulse until imgPulseStop is called. |
| PULSE_TIMEBASE_100KHZ | imgPulseCreate | Specifies a 100 kHz timebase to use for pulse generation. |

**Table A-2.** Constants Summary (Continued)

| Constant | Use With | Description |
|---|---|---|
| PULSE_TIMEBASE_50MHZ | `imgPulseCreate` | Specifies a 50 MHz timebase to use for pulse generation. |
| PULSE_TIMEBASE_PIXELCLK | `imgPulseCreate` | Specifies the incoming pixel clock from the camera to use as a timebase for pulse generation. |

# B

# Status Codes

This appendix describes the status codes returned by NI-IMAQ.

Each NI-IMAQ function returns a status code that indicates whether the function was performed successfully. A summary of the status codes is listed in Table B-1.

**Table B-1.** Status Code Summary

| Error Code | Status Name | Description |
|---|---|---|
| –1074397183 | IMG_ERR_NCAP | Function not implemented. |
| –1074397182 | IMG_ERR_OVRN | Too many interfaces open. |
| –1074397181 | IMG_ERR_EMEM | Not enough memory to perform the operation. |
| –1074397180 | IMG_ERR_OSER | Operating system error occurred. |
| –1074397179 | IMG_ERR_PAR1 | Function-specific; see function description. |
| –1074397178 | IMG_ERR_PAR2 | Function-specific; see function description. |
| –1074397177 | IMG_ERR_PAR3 | Function-specific; see function description. |
| –1074397176 | IMG_ERR_PAR4 | Function-specific; see function description. |
| –1074397175 | IMG_ERR_PAR5 | Function-specific; see function description. |
| –1074397174 | IMG_ERR_PAR6 | Function-specific; see function description. |
| –1074397173 | IMG_ERR_PAR7 | Function-specific; see function description. |
| –1074397172 | IMG_ERR_MXBF | Too many buffers already allocated. |
| –1074397171 | IMG_ERR_DLLE | DLL internal error; bad logic state. |
| –1074397170 | IMG_ERR_BSIZ | Buffer size used is too small for minimum acquisition frame. |
| –1074397169 | IMG_ERR_MXBI | Exhausted buffer IDs. |
| –1074397168 | IMG_ERR_ELCK | Cannot lock buffers down; no more memory. |

**Table B-1.** Status Code Summary (Continued)

| Error Code | Status Name | Description |
|---|---|---|
| –1074397167 | IMG_ERR_DISE | Error releasing the image buffer. |
| –1074397166 | IMG_ERR_BBUF | Bad buffer pointer in list. |
| –1074397165 | IMG_ERR_NLCK | Buffer list is not locked. |
| –1074397164 | IMG_ERR_NCAM | No camera defined for this channel. |
| –1074397163 | IMG_ERR_BINT | Bad interface. |
| –1074397162 | IMG_ERR_BROW | Rowbytes is less than region of interest. |
| –1074397161 | IMG_ERR_BROI | ROI width is greater than rowbytes. |
| –1074397160 | IMG_ERR_BCMF | Bad camera file (check syntax). |
| –1074397159 | IMG_ERR_NVBL | Not successful because of hardware limitations. |
| –1074397158 | IMG_ERR_NCFG | Invalid action; no buffers configured for session. |
| –1074397157 | IMG_ERR_BBLF | Buffer list does not contain a valid final command. |
| –1074397156 | IMG_ERR_BBLE | Buffer list contains an invalid command. |
| –1074397155 | IMG_ERR_BBLB | A buffer list buffer is NULL. |
| –1074397154 | IMG_ERR_NAIP | No acquisition in progress. |
| –1074397153 | IMG_ERR_VLCK | Cannot get lock on video source. |
| –1074397152 | IMG_ERR_BDMA | Bad DMA transfer. |
| –1074397151 | IMG_ERR_AIOP | Cannot perform request; acquisition in progress. |
| –1074397150 | IMG_ERR_TIMO | Wait timed out; acquisition not complete. |
| –1074397149 | IMG_ERR_NBUF | No buffers available; too early in acquisition. |
| –1074397148 | IMG_ERR_ZBUF | Zero buffer size; no bytes filled. |
| –1074397147 | IMG_ERR_HLPR | Bad parameter to low-level; check attributes and high-level arguments. |

**Table B-1.** Status Code Summary (Continued)

| Error Code | Status Name | Description |
|---|---|---|
| –1074397146 | IMG_ERR_BTRG | Trigger loopback problem; cannot drive trigger with action enabled. |
| –1074397145 | IMG_ERR_NINF | No interface found. |
| –1074397144 | IMG_ERR_NDLL | Unable to load DLL (LabWindows/CVI only). |
| –1074397143 | IMG_ERR_NFNC | Unable to find API function in DLL (LabWindows/CVI only). |
| –1074397142 | IMG_ERR_NOSR | Unable to allocate system resources (LabWindows/CVI only). |
| –1074397141 | IMG_ERR_BTAC | No trigger action; acquisition will time out. |
| –1074397140 | IMG_ERR_FIFO | FIFO overflow caused acquisition to halt. |
| –1074397139 | IMG_ERR_MLCK | Memory lock error; cannot perform acquisition. |
| –1074397138 | IMG_ERR_ILCK | Interface locked. |
| –1074397137 | IMG_ERR_NEPK | No external pixel clock. |
| –1074397136 | IMG_ERR_SCLM | Field scaling mode not supported. |
| –1074397135 | IMG_ERR_SCC1 | Channel not set to 1 when using StillColor RGB acquisition. |
| –1074397134 | IMG_ERR_SMALLALLOC | Error during small buffer allocation. |
| –1074397133 | IMG_ERR_ALLOC | Error during large buffer allocation. |
| –1074397132 | IMG_ERR_BADCAMTYPE | Bad camera type; camera needs to be of type NTSC or PAL. |
| –1074397131 | IMG_ERR_BADPIXTYPE | Camera not supported; must be an 8-bit camera. |
| –1074397130 | IMG_ERR_BADCAMPARAM | Bad camera parameter in configuration file. |
| –1074397129 | IMG_ERR_PALKEYDTCT | PAL key detection error. |
| –1074397128 | IMG_ERR_BFRQ | Bad frequency values. |
| –1074397127 | IMG_ERR_BITP | Bad interface type. |

**Table B-1.** Status Code Summary (Continued)

| Error Code | Status Name | Description |
|---|---|---|
| –1074397126 | IMG_ERR_HWNC | Hardware not capable of supporting this function. |
| –1074397125 | IMG_ERR_SERIAL | Serial port error. |
| –1074397124 | IMG_ERR_MXPI | Exhausted pulse IDs. |
| –1074397123 | IMG_ERR_BPID | Bad pulse ID. |
| –1074397121 | IMG_ERR_SERIAL_TIMO | Serial transmit/receive timeout. |
| –1074397120 | IMG_ERR_PG_TOO_MANY | Too many pattern generation transitions defined. |
| –1074397119 | IMG_ERR_PG_BAD_TRANS | Bad pattern generation transition time. |
| –1074397118 | IMG_ERR_PLNS | Pulse not started. |
| –1074397117 | IMG_ERR_BPMD | Bad pulse mode. |
| –1074397116 | IMG_ERR_NSAT | Nonsettable attribute. |
| –1074397115 | IMG_ERR_HYBRID | Cannot mix system and onboard memory buffers. |
| –1074397114 | IMG_ERR_BADFILFMT | Pixel depth not supported with this file format. |
| –1074397113 | IMG_ERR_BADFILEXT | File extension not supported. |
| –1074397112 | IMG_ERR_NRTSI | Too many RTSI triggers mapped; you can use only four RTSI lines at once. |
| –1074397111 | IMG_ERR_MXTRG | Exhausted trigger resources. |
| –1074397110 | IMG_ERR_MXRC | Exhausted resource (general). |
| –1074397109 | IMG_ERR_OOR | Parameter out of range. |
| –1074397108 | IMG_ERR_NPROG | FPGA not programmed. |
| –1074397107 | IMG_ERR_NEOM | Not enough onboard memory to perform the operation. |
| –1074397106 | IMG_ERR_BDTYPE | Bad display type; buffer cannot be displayed with `imgPlot`. |
| –1074397105 | IMG_ERR_THRDACCDEN | Thread denied access to function. |

**Table B-1.** Status Code Summary (Continued)

| Error Code | Status Name | Description |
|---|---|---|
| –1074397104 | IMG_ERR_BADFILWRT | Could not write the file. |
| –1074397103 | IMG_ERR_AEXM | Already called ExamineBuffer once; call ReleaseBuffer. |
| 1073086471 | IMG_WRN_WLOR | Warning; white level out of range. |
| 1073086470 | IMG_WRN_OATTR | Warning; old attribute used. |
| 1073086469 | IMG_WRN_BRST | Warning; bad quality colorburst (StillColor mode). |
| 1073086468 | IMG_WRN_BLKG | Warning; unstable blanking reference (StillColor mode). |
| 1073086467 | IMG_WRN_ILCK | Warning; interface still locked. |
| 1073086466 | IMG_WRN_CONF | Warning; change requires reconfiguration to take effect. |
| 1073086465 | IMG_WRN_BCAM | Warning; corrupt camera file detected. |

# C

# Calibration Information for the IMAQ PCI/PXI-1411

The IMAQ PCI/PXI-1411 (revision C and higher ) is factory calibrated to improve measurement accuracy and board-to-board consistency. You can programmatically set target values for Color Contrast, Color Saturation, and Color Brightness using `imgSetAttribute` in LabWindows/CVI or C, or the IMAQ Property Node in LabVIEW. The IMAQ PCI/PXI-1411 uses these values to set the properties on the board. Generally, there is a small difference between the user-supplied values and the actual board settings, since the board uses discrete values for these hardware settings. This difference is known as resolution error.

The IMAQ PCI/PXI-1411 and the NI-IMAQ driver software allow you to correct for this error in your application. Use `imgGetAttribute` in LabWindows/CVI or C, or the IMAQ Property Node in LabVIEW to get the actual hardware values for these settings. You can use the actual values returned by these functions to improve the accuracy of the measurement.

Using the actual values can improve the accuracy of your measurement. For example, the typical accuracy measurement of a luminance value in the IMAQ PCI/PXI-1411 is 2.5%. After correction, the actual contrast increases the accuracy to approximately 0.5%.

Use the following procedure to improve measurement resolution using the actual contrast:

1. Set the contrast to a known value.
2. Acquire an image using `imgSnap` or the IMAQ Snap VI.
3. Get the actual contrast using the IMAQ Property Node or `imgGetAttribute`.
4. Divide the input value of contrast by the actual contrast value.
5. Multiply the image data, in pixel values, by this ratio.

# D

# Technical Support Resources

## Web Support

National Instruments Web support is your first stop for help in solving installation, configuration, and application problems and questions. Online problem-solving and diagnostic resources include frequently asked questions, knowledge bases, product-specific troubleshooting wizards, manuals, drivers, software updates, and more. Web support is available through the Technical Support section of `ni.com`

## NI Developer Zone

The NI Developer Zone at `ni.com/zone` is the essential resource for building measurement and automation systems. At the NI Developer Zone, you can easily access the latest example programs, system configurators, tutorials, technical news, as well as a community of developers ready to share their own techniques.

## Customer Education

National Instruments provides a number of alternatives to satisfy your training needs, from self-paced tutorials, videos, and interactive CDs to instructor-led hands-on courses at locations around the world. Visit the Customer Education section of `ni.com` for online course schedules, syllabi, training centers, and class registration.

## System Integration

If you have time constraints, limited in-house technical resources, or other dilemmas, you may prefer to employ consulting or system integration services. You can rely on the expertise available through our worldwide network of Alliance Program members. To find out more about our Alliance system integration solutions, visit the System Integration section of `ni.com`

# Worldwide Support

National Instruments has offices located around the world to help address your support needs. You can access our branch office Web sites from the Worldwide Offices section of `ni.com`. Branch office Web sites provide up-to-date contact information, support phone numbers, e-mail addresses, and current events.

If you have searched the technical support resources on our Web site and still cannot find the answers you need, contact your local office or National Instruments corporate. Phone numbers for our worldwide offices are listed at the front of this manual.

# 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

| | |
|---|---|
| – | negative of, or minus |
| Ω | ohm |
| / | per |
| % | percent |
| ± | plus or minus |
| + | positive of, or plus |

## A

| | |
|---|---|
| A | Amperes. |
| AC | Alternating current. |
| A/D | Analog-to-digital. |
| acquisition window | The image size specific to a video standard or camera resolution. |

| | |
|---|---|
| active line region | The region of lines actively being stored. Defined by a line start (relative to the vertical synchronization signal [VSYNC]) and a line count. |
| active pixel region | The region of pixels actively being stored. Defined by a pixel start (relative to the horizontal synchronization signal [HSYNC]) and a pixel count. |
| ADC | Analog-to-digital converter. An electronic device, often an integrated circuit, that converts an analog voltage to a digital value. |
| address | Value that identifies a specific location (or series of locations) in memory. |
| ANSI | American National Standards Institute. |
| antichrominance filter | Removes the color information from the video signal. |
| API | Application programming interface. |
| area | A rectangular portion of an acquisition window or frame that is controlled and defined by software. |
| array | Ordered, indexed set of data elements of the same type. |
| ASIC | Application-Specific Integrated Circuit. A proprietary semiconductor component designed and manufactured to perform a set of specific functions for specific customer needs. |
| aspect ratio | The ratio of a picture or image's width to its height. |

## 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. |
| back porch | The area of the video signal between the rising edge of the horizontal synchronization signal (HSYNC) and the active video information. |
| black reference level | The level that represents the darkest an image can get. *See also* white reference level. |

| | |
|---|---|
| buffer | Temporary storage for acquired data. |
| bus | A group of conductors that interconnect individual circuitry in a computer, such as the PCI bus; typically the expansion vehicle to which I/O or other devices are connected. |

## C

| | |
|---|---|
| C | Celsius. |
| cache | High-speed processor memory that buffers commonly used instructions or data to increase processing throughput. |
| CCIR | Comite Consultatif International des Radiocommunications. A committee that developed standards for video signals. Also used to describe signals, boards, and cameras that adhere to the CCIR standards. |
| CMOS | Complementary metal-oxide semiconductor. |
| 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. |
| conversion device | Device that transforms a signal from one form to another. For example, analog-to-digital converters (ADCs) for analog input and digital-to-analog converters (DACs) for analog output. |
| CPU | Central processing unit. |

## D

| | |
|---|---|
| D/A | Digital-to-analog. |
| DAC | Digital-to-analog converter. An electronic device, often an integrated circuit, that converts a digital number into a corresponding analog voltage or current. |

| | |
|---|---|
| DAQ | Data acquisition. (1) Collecting and measuring electrical signals from sensors, transducers, and test probes or fixtures and inputting them to a computer for processing. (2) Collecting and measuring the same kinds of electrical signals with A/D or DIO boards plugged into a computer, and possibly generating control signals with D/A and/or DIO boards in the same computer. |
| dB | Decibel. The unit for expressing a logarithmic measure of the ratio of two signal levels: $dB = 20\log_{10} V1/V2$, for signals in volts. |
| DC | Direct current. |
| default setting | A default parameter value recorded in the driver. In many cases, the default input of a control is a certain value (often 0). |
| DIN | Deutsche Industrie Norme. A format for electrical connectors. |
| 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. |
| DMA | Direct memory access. A method by which data can be transferred between computer memory and a device or memory on the bus while the processor does something else. DMA is the fastest method of transferring data to/from computer memory. |
| DRAM | Dynamic RAM. |
| driver | Software that controls a specific hardware device, such as an IMAQ or DAQ device. |
| dynamic range | The ratio of the largest signal level a circuit can handle to the smallest signal level it can handle (usually taken to be the noise level), normally expressed in decibels. |

## E

| | |
|---|---|
| EEPROM | Electrically erasable programmable read-only memory. ROM that can be erased with an electrical signal and reprogrammed. |
| external trigger | A voltage pulse from an external source that triggers an event such as A/D conversion. |

## F

| | |
|---|---|
| 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. |
| FIFO | First-in first-out memory buffer. The first data stored is the first data sent to the acceptor. FIFOs are used on IMAQ devices to temporarily store incoming data until that data can be retrieved. |
| flash ADC | An ADC whose output code is determined in a single step by a bank of comparators and encoding logic. |
| frame | A complete image. In interlaced formats, a frame is composed of two fields. |
| front porch | The area of a video signal between the start of the horizontal blank and the start of the horizontal synchronization signal (HSYNC). |
| 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. |
| genlock | The process of synchronizing a video source to the signal from a separate video source. The circuitry aligns the video timing signals by locking together the horizontal, vertical, and color subcarrier frequencies and phases and generates a pixel clock that clocks pixel data into memory for display or into another circuit for processing. |

| | |
|---|---|
| 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. |
| HSYNC | Horizontal synchronization signal. The synchronization pulse signal produced at the beginning of each video scan line that keeps a video monitor's horizontal scan rate in step with the transmission of each new line. |
| 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. |
| Hz | Hertz. Frequency in units of 1/second. |

## I

| | |
|---|---|
| IC | Integrated circuit. |
| IEEE | Institute of Electrical and Electronics Engineers. |
| in. | Inches. |
| INL | Integral nonlinearity. A measure in LSB of the worst-case deviation from the ideal A/D or D/A transfer characteristic of the analog I/O circuitry. |
| instrument driver | A set of high-level software functions, such as NI-IMAQ, that control specific plug-in computer boards. Instrument drivers are available in several forms, ranging from a function callable from a programming language to a virtual instrument (VI) in LabVIEW. |
| interlaced | A video frame composed of two interleaved fields. The number of lines in a field are half the number of lines in an interlaced frame. |

| | |
|---|---|
| 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. |
| interrupt | A computer signal indicating that the CPU should suspend its current task to service a designated activity. |
| interrupt level | The relative priority at which a device can interrupt. |
| IRE | A relative unit of measure (named for the Institute of Radio Engineers). 0 IRE corresponds to the blanking level of a video signal, 100 IRE to the white level. Note that for CCIR/PAL video, the black level is equal to the blanking level or 0 IRE, while for RS-170/NTSC video, the black level is at 7.5 IRE. |
| IRQ | Interrupt request. *See* interrupt. |

## 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. |
| Kword | 1,024 words of memory. |

## L

| | |
|---|---|
| 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. |

| | |
|---|---|
| LSB | Least significant bit. |
| LUT | Lookup 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 lookup 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. |
| MB | Megabyte of memory. |
| Mbytes/s | A unit for data transfer that means 1 million or $10^6$ bytes/s. |
| memory buffer | *See* buffer. |
| memory window | Continuous blocks of memory that can be accessed quickly by changing addresses on the local processor. |
| MSB | Most significant bit. |
| MTBF | Mean time between failure. |
| mux | Multiplexer. A switching device with multiple inputs that selectively connects one of its inputs to its output. |

## N

| | |
|---|---|
| NI-IMAQ | Driver software for National Instruments IMAQ hardware. |
| noninterlaced | A video frame where all the lines are scanned sequentially, instead of divided into two frames as in an interlaced video frame. |
| NTSC | National Television Standards Committee. The committee that developed the color video standard used primarily in North America, which uses 525 lines per frame. *See also* PAL. |
| NVRAM | Nonvolatile RAM. RAM that is not erased when a device loses power or is turned off. |

## O

| | |
|---|---|
| operating system | Base-level software that controls a computer, runs programs, interacts with users, and communicates with installed hardware or peripheral devices. |

## P

| | |
|---|---|
| PAL | Phase Alternation Line. One of the European video color standards. PAL uses 625 lines per frame. *See also* NTSC. |
| PCI | Peripheral Component Interconnect. A high-performance expansion bus architecture originally developed by Intel to replace ISA and EISA. PCI offers a theoretical maximum transfer rate of 132 Mbytes/s. |
| picture aspect ratio | The ratio of the active pixel region to the active line region. For standard video signals like RS-170 or CCIR, the full-size picture aspect ratio normally is 4/3 (1.33). |
| 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 aspect ratio | The ratio between the physical horizontal size and the vertical size of the region covered by the pixel. An acquired pixel should optimally be square, thus the optimal value is 1.0, but typically it falls between 0.95 and 1.05, depending on camera quality. |
| pixel clock | Divides the incoming horizontal video line into pixels. |
| pixel count | The total number of pixels between two horizontal synchronization signals (HSYNCs). The pixel count determines the frequency of the pixel clock. |
| PLL | Phase-locked loop. Circuitry that provides a very stable pixel clock that is referenced to another signal, such as an incoming horizontal synchronization signal (HSYNC). |
| protocol | The exact sequence of bits, characters, and control codes used to transfer data between computers and peripherals through a communications channel. |
| pts | Points. |

# 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. |
| relative accuracy | A measure in LSB of the accuracy of an ADC; it includes all nonlinearity and quantization errors but does not include offset and gain errors of the circuitry feeding the ADC. |
| 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. |
| 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). |
| ribbon cable | A flat cable in which the conductors are side by side. |
| 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. |
| ROM | Read-only memory. |
| RS-170 | The U.S. standard used for black-and-white television. |
| RTSI bus | Real-Time System Integration Bus. The National Instruments timing bus that connects IMAQ and DAQ boards directly by means of connectors on top of the boards for precise synchronization of functions. |

## 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. |
| scaling down circuitry | Circuitry that scales down the resolution of a video signal. |
| scatter-gather DMA | A type of DMA that allows the DMA controller to reconfigure on-the-fly. |
| SRAM | Static RAM. |
| StillColor | A post-processing algorithm that allows the acquisition of high-quality color images generated either by an RGB or composite (NTSC or PAL) camera using a monochrome video acquisition board. |
| sync | Tells the display where to put a video picture. The horizontal sync indicates the picture's left-to-right placement and the vertical sync indicates top-to-bottom placement. |
| syntax | Set of rules to which statements must conform in a particular programming language. |
| system RAM | RAM installed on a personal computer and used by the operating system, as contrasted with onboard RAM. |

## T

| | |
|---|---|
| transfer rate | The rate, measured in bytes/s, at which data is moved from source to destination after software initialization and set up operations. The maximum rate at which the hardware can operate. |
| trigger | Any event that causes or starts some form of data capture. |
| trigger control and mapping circuitry | Circuitry that routes, monitors, and drives external and RTSI bus trigger lines. You can configure each of these lines to start or stop acquisition on a rising or falling edge. |
| TTL | Transistor-transistor logic. |

## U

| | |
|---|---|
| UV plane | *See* YUV. |

## V

| | |
|---|---|
| V | Volts. |
| VCO | Voltage-controlled oscillator. An oscillator that changes frequency depending on a control signal. Use VCO in a phase-locked loop to generate a stable pixel clock. |
| 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. |
| video line | A video line consists of a horizontal synchronization signal, back porch, active pixel region, and a front porch. |
| VSYNC | Vertical synchronization signal. The synchronization pulse generated at the beginning of each video field that tells the video monitor when to start a new field. |

## W

| | |
|---|---|
| white reference level | The level that defines what is white for a particular video system. *See also* black reference level. |

## 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

acquisition functions, 4-1 to 4-13
- imgMemLock, 4-2
- imgMemUnlock, 4-3
- imgSessionAbort, 4-4
- imgSessionAcquire, 4-5
- imgSessionConfigure, 4-6
- imgSessionCopyArea, 4-7 to 4-8
- imgSessionCopyBuffer, 4-9
- imgSessionExamineBuffer, 4-11 to 4-12
- imgSessionReleaseBuffer, 4-13

arrays, 1-2
attribute functions, 4-14 to 4-23
- imgGetAttribute, 4-15
- imgGetCameraAttributeNumeric, 4-16
- imgGetCameraAttributeString, 4-17
- imgSessionGetLostFramesList, 4-18
- imgSessionSetUserLUT8bits, 4-19
- imgSessionSetUserLUT16bits, 4-20
- imgSetAttribute, 4-21
- imgSetCameraAttributeNumeric, 4-22
- imgSetCameraAttributeString, 4-23

attribute summary (table), A-1 to A-7

## B

buffer management functions, 4-24 to 4-34
- imgCreateBuffer, 4-25 to 4-26
- imgCreateBufList, 4-27
- imgDisposeBuffer, 4-28
- imgDisposeBufList, 4-29
- imgGetBufferElement, 4-30
- imgSessionClearBuffer, 4-31
- imgSetBufferElement, 4-32 to 4-33

## C

calibration, C-1
code examples, 1-6
constants summary (table), A-8 to A-18
conventions used in manual, *iv*
customer education, D-1

## D

data types. *See* variable data types.

## E

examples of code, 1-6

## F

telephone support numbers, C-2
functions
- generic, 2-1 to 2-4
- high-level, 3-1 to 3-43
  - grab functions, 3-5 to 3-9
  - miscellaneous functions, 3-38 to 3-43
  - ring and sequence functions, 3-10 to 3-16
  - signal I/O functions, 3-17 to 3-37
  - snap functions, 3-1 to 3-4
- LabWindows/CVI function tree (table), 1-3 to 1-6
- low-level, 4-1 to 4-43
  - acquisition functions, 4-1 to 4-13
  - attribute functions, 4-14 to 4-23
  - buffer management functions, 4-24 to 4-33
  - interface functions, 4-34 to 4-36
  - utility functions, 4-37 to 4-43

## G

generic functions, 2-1 to 2-4
    imgClose, 2-4
    imgInterfaceOpen, 2-2
    imgSessionOpen, 2-3
grab functions, 3-5 to 3-9
    imgGrab, 3-7
    imgGrabArea, 3-8 to 3-9
    imgGrabSetup, 3-6

## H

high-level functions, 3-1 to 3-43
    grab functions, 3-5 to 3-9
    miscellaneous functions, 3-38 to 3-43
    ring and sequence functions, 3-10 to 3-16
    signal I/O functions, 3-17 to 3-37
    snap functions, 3-1 to 3-4
        imgSnap, 3-2
        imgSnapArea, 3-3 to 3-4

## I

IMAQ PCI/PXI-1411 calibration information, C-1
imgClose function, 2-4
imgCreateBuffer function, 4-25 to 4-26
imgCreateBufList function, 4-27
imgDisposeBuffer function, 4-28
imgDisposeBufList function, 4-29
imgGetAttribute function, 4-15
imgGetBufferElement function, 4-30
imgGetCameraAttributeNumeric function, 4-16
imgGetCameraAttributeString function, 4-17
imgGrab function, 3-7
imgGrabArea function, 3-8 to 3-9
imgGrabSetup function, 3-6
imgInterfaceOpen function, 2-2
imgInterfaceQueryNames function, 4-35
imgInterfaceReset function, 4-36
imgMemLock function, 4-2
imgMemUnlock function, 4-3
imgPlot function, 4-38 to 4-39
imgPlotDC, 4-40 to 4-41
imgPulseCreate function, 3-31 to 3-33
imgPulseDispose function, 3-34
imgPulseRate function, 3-35
imgPulseStart function, 3-36
imgPulseStop function, 3-37
imgRingSetup function, 3-11 to 3-12
imgSequenceSetup function, 3-13 to 3-14
imgSessionAbort function, 4-4
imgSessionAcquire function, 4-5
imgSessionClearBuffer function, 4-31
imgSessionConfigure function, 4-6
imgSessionCopyArea function, 4-7 to 4-8
imgSessionCopyBuffer function, 4-9
imgSessionExamineBuffer function, 4-11 to 4-12
imgSessionGetBufferSize function, 3-43
imgSessionGetLostFramesList function, 4-18
imgSessionGetROI function, 3-42
imgSessionLineTrigSource function, 3-20 to 3-21
imgSessionOpen function, 2-3
imgSessionReleaseBuffer function, 4-13
imgSessionSaveBufferEx function, 4-42
imgSessionSetROI function, 3-40 to 3-41
imgSessionSetUserLUT8bits, 4-19
imgSessionSetUserLUT16bits, 4-20
imgSessionStartAcquisition function, 3-15
imgSessionStatus function, 3-39
imgSessionStopAcquisition function, 3-16
imgSessionTriggerClear function, 3-22
imgSessionTriggerConfigure function, 3-18 to 3-19
imgSessionTriggerDrive function, 3-23 to 3-24

imgSessionTriggerRead function, 3-25 to 3-26
imgSessionWaitSignal function, 3-27 to 3-28
imgSessionWaitSignalAsync function, 3-29 to 3-30
imgSetAttribute function, 4-21
imgSetBufferElement function, 4-32 to 4-33
imgSetCameraAttributeNumeric function, 4-22
imgSetCameraAttributeString function, 4-23
imgShowError function, 4-43
imgSnap function, 3-2
imgSnapArea function, 3-3 to 3-4
interface functions, 4-34 to 4-36
    imgInterfaceQueryNames, 4-35
    imgInterfaceReset, 4-36

## L

LabVIEW software, 1-2
LabWindows/CVI software
    function tree for image acquisition (table), 1-3 to 1-6
    programming considerations, 1-3
low-level functions, 4-1 to 4-43
    acquisition functions, 4-1 to 4-12
    attribute functions, 4-14 to 4-23
    buffer management functions, 4-24 to 4-33
    interface functions, 4-34 to 4-36
    utility functions, 4-37 to 4-43

## M

miscellaneous high-level functions, 3-35 to 3-40
    imgSessionGetBufferSize, 3-40
    imgSessionGetROI, 3-39
    imgSessionSetROI, 3-37 to 3-38
    imgSessionStatus, 3-36

## N

National Instruments Web support, D-1
NI Developer Zone, D-1
NI-IMAQ attributes (table), A-1 to A-7

## P

programming language considerations, 1-2 to 1-6
    code examples, 1-6
    LabVIEW, 1-2
    LabWindows/CVI, 1-3 to 1-6
    third-party programming environments, 1-6

## R

region of interest, 1-6
ring and sequence functions, 3-10 to 3-16
    imgRingSetup, 3-11 to 3-12
    imgSequenceSetup, 3-13 to 3-14
    imgSessionStartAcquisition, 3-15
    imgSessionStopAcquisition, 3-16

## S

sequence functions. *See* ring and sequence functions.
signal I/O functions, 3-17 to 3-37
    imgPulseCreate, 3-31 to 3-33
    imgPulseDispose, 3-34
    imgPulseRate, 3-35
    imgPulseStart, 3-36
    imgPulseStop, 3-37
    imgSessionLineTrigSource, 3-20 to 3-21
    imgSessionTriggerClear, 3-22

imgSessionTriggerConfigure, 3-18 to 3-19
imgSessionTriggerDrive, 3-23 to 3-24
imgSessionTriggerRead, 3-25 to 3-26
imgSessionWaitSignal, 3-27 to 3-28
imgSessionWaitSignalAsync, 3-29 to 3-30
snap functions, 3-1 to 3-4
imgSnap, 3-2
imgSnapArea, 3-3 to 3-4
status codes, B-1 to B-5
format, 1-1
system integration, by National Instruments, D-1

## T

technical support resources, D-1

## U

utility functions, 4-37 to 4-43
imgPlot, 4-38 to 4-39
imgPlotDC, 4-40 to 4-41
imgSessionSaveBufferEx, 4-42
imgShowError, 4-43

## V

variable data types
arrays, 1-2
primary types (table), 1-2
Windows 95 and Windows NT, 1-1

## W

Web support from National Instruments, D-1
worldwide technical support, D-2