NI-IMAQ User Manual for Mac OS

Image Acquisition Software for PCI

June 1997 Edition
Part Number 371641A-01

© Copyright 1997 National Instruments Corporation. All Rights Reserved.

Internet Support

support@natinst.com
E-mail: info@natinst.com
FTP Site: ftp.natinst.com
Web Address: http://www.natinst.com

Bulletin Board Support

BBS United States: (512) 794-5422
BBS United Kingdom: 01635 551422
BBS France: 01 48 65 15 59

Fax-on-Demand Support

(512) 418-1111

Telephone Support (U.S.)

Tel: (512) 795-8248
Fax: (512) 794-5678

International Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00,
Finland 09 725 725 11, France 01 48 14 24 24, Germany 089 741 31 30,
Hong Kong 2645 3186, Israel 03 5734815, Italy 02 413091, Japan 03 5472 2970,
Korea 02 596 7456, Mexico 5 520 2635, Netherlands 0348 433466, Norway 32 84 84 00,
Singapore 2265886, Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 200 51 51,
Taiwan 02 377 1200, U.K. 01635 523545

National Instruments Corporate Headquarters

6504 Bridge Point Parkway   Austin, TX 78730-5039   Tel: (512) 794-0100

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 manual 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

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

Trademarks

LabVIEW®, RTSI®, and IMAQ™ are trademarks of National Instruments Corporation.

Product and company names listed are trademarks or trade names of their respective companies.

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.

Table of Contents

About This Manual

Chapter 1
Introduction to NI-IMAQ

Chapter 2
Fundamentals of Building Applications with NI-IMAQ

Chapter 3
Software Overview

Chapter 4
Programming with NI-IMAQ

Customer Communication

Glossary

Index

Figures

Tables

About This Manual

NI-IMAQ software is a powerful application programming interface (API) between your image acquisition application and the National Instruments image acquisition (IMAQ) devices for PCI bus computers. This manual explains how to use your NI-IMAQ software.

How to Use the NI-IMAQ Manual Set

To install your software and documentation set, you should begin by reading the NI-IMAQ release notes. Then read Chapter 1, *Introduction*, of *Getting Started with Your IMAQ PCI-1408 and the NI-IMAQ Software for Mac OS*, which contains a flowchart that illustrates the sequence of steps you should take to learn about and get started with NI-IMAQ.

When you are familiar with the material in this manual, you can use the *NI-IMAQ Function Reference Manual for Mac OS*, which contains detailed descriptions of the NI-IMAQ functions.

Organization of This Manual

The *NI-IMAQ User Manual for Mac OS* is organized as follows:

- Chapter 1, *Introduction to NI-IMAQ*, describes the NI-IMAQ software and lists the application development environments compatible with NI-IMAQ.
- Chapter 2, *Fundamentals of Building Applications with NI-IMAQ*, describes the fundamentals of creating NI-IMAQ applications for Mac OS, describes the files used to build these applications, and tells you where to find sample programs.
- Chapter 3, *Software Overview*, describes the classes of NI-IMAQ functions and briefly describes each function.
- Chapter 4, *Programming with NI-IMAQ*, contains an overview of the NI-IMAQ library, a description of the programming flow of NI-IMAQ, and programming examples.

- The *Customer Communication* appendix contains forms you can use to request help from National Instruments or to comment on our products and manuals.
- The *Glossary* contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.
- The *Index* contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

Conventions Used in This Manual

The following conventions are used in this manual:

bold	Bold text denotes menus, menu items, or dialog box buttons or options.
bold italic	Bold italic text denotes a note, caution, or warning.
italic	Italic text denotes emphasis, a cross reference, or an introduction to a key concept.
`italic monospace`	Italic text in this font denotes that you must supply the appropriate words or values in the place of these items.
`monospace`	Lowercase text in this font denotes text or characters that are to be literally input 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, variables, filenames, and extensions, and for statements and comments taken from program code.

The *Glossary* lists abbreviations, acronyms, metric prefixes, mnemonics, symbols, and terms.

About the National Instruments Documentation Set

The *NI-IMAQ User Manual for Mac OS* is one piece of the documentation set for your system. You could have any of several types of documents, depending on the hardware and software in your system. Use the documents you have as follows:

- Your IMAQ documentation—These documents have detailed information about the IMAQ hardware that plugs into or is connected to your computer. Use these manuals for hardware installation and configuration instructions, hardware specification information, and application hints.
- Software documentation—Examples of software documentation you might have are the LabVIEW documentation and the NI-IMAQ documentation. After you have set up your hardware system, use either the LabVIEW application software or the NI-IMAQ documentation to help you write your application. If you have a large and complicated system, it is worthwhile to look through the software documentation before you configure your hardware.

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. These forms are in the *Customer Communication* appendix at the end of this manual.

Chapter 1

Introduction to NI-IMAQ

This chapter describes the NI-IMAQ software and lists the application development environments compatible with NI-IMAQ.

About the NI-IMAQ Software

Thank you for buying a National Instruments image acquisition (IMAQ) device for PCI, which includes NI-IMAQ software. NI-IMAQ is a set of functions that controls the National Instruments plug-in IMAQ devices for image acquisition and Real-Time System Integration (RTSI) bus multiboard synchronization.

NI-IMAQ has both high-level I/O functions for maximum ease of use and low-level I/O functions for maximum flexibility and performance. Examples of high-level functions are snap and grab image acquisition. Examples of low-level functions are buffer setup and video configuration. NI-IMAQ enhances the performance of National Instruments IMAQ devices because it lets multiple devices operate at their peak performance.

NI-IMAQ includes a buffer and data manager that uses sophisticated techniques for handling and managing image acquisition buffers so that you can simultaneously acquire and process data. NI-IMAQ uses direct memory access (DMA) to transfer all data.

NI-IMAQ is a library of routines that work with National Instruments IMAQ devices. NI-IMAQ contains methods for overcoming difficulties ranging from simple device initialization to advanced high-speed real-time image acquisition. The number of services you need for your applications depends on the types of IMAQ devices you have and the complexity of your applications.

Application Development Environments

This release of NI-IMAQ supports the following Application Development Environments (ADEs) for Mac OS:

- LabVIEW version 4.*x*
- Metrowerks CodeWarrior 10 or later

☞ **Note:** ***Although NI-IMAQ has been tested and found to work with these ADEs, other ADEs or higher versions of the ADEs listed above may also work.***

Files on the NI-IMAQ software diskettes may be compressed. Always run the NI-IMAQ installation utility to extract the files you want. For a brief description of the directories produced by the install programs and the names and purposes of the uncompressed files, consult the `ReadMe` file.

Fundamentals of Building Applications with NI-IMAQ

Chapter 2

This chapter describes the fundamentals of creating NI-IMAQ applications for Mac OS, describes the files used to build these applications, and tells you where to find sample programs.

The NI-IMAQ Libraries

The NI-IMAQ for Mac OS function libraries are shared libraries that you can add to your application project file to access the IMAQ functions.

Shared libraries contain information about their exported functions. They indicate the presence and location of the IMAQ routines. Depending on the development tools you are using, you may give the shared library functions information through shared libraries or through function declarations.

Because using functional prototypes is a good programming practice, NI-IMAQ is packaged with functional prototype files for different Mac OS development tools.

Creating an Application

This section outlines the process for developing NI-IMAQ applications using C for Mac OS. Detailed instructions on creating project and source files are not included. For information on creating and managing project files, consult the documentation included with your particular development environment.

When programming, use the following guidelines:

- You must define the constant _NIMAC prior to including any NI-IMAQ header files. You can define this constant in your source files by using the `#define` *`directive`*; that is, `#define _NIMAC`. Or, you can add the definition to your project file's preprocessor definitions if your environment supports this feature.
- All C source files that use NI-IMAQ functions must include the `NIIMAQ.H` header file. Add this file to the top of your source files.
- You must add the `imaq.dll` shared library to your project. Some environments allow you to add shared libraries simply by inserting them into your list of project files.
- When compiling, you will need to indicate where the compiler can find the NI-IMAQ header files and shared libraries. Most of the files you need for development are located under the NI-IMAQ target installation directory. The target installation directory is the `NI-IMAQ` folder. The include files are located under the `Headers` subfolder. The shared libraries are installed into the `Extensions` folder in your `System Folder` and include `imaq.dll`, `NIINI`, and `NIMGR`. If you are using the IMAQ PCI-1408, a PPC driver called `IMAQ_1408` is also installed.

Sample Programs

Please refer to the `ReadMe` file located in your target installation directory for the latest details on NI-IMAQ sample programs. These programs are installed in the `Sample` subfolder under the target installation folder if you elected to install the sample files.

Chapter 3

Software Overview

This chapter describes the classes of NI-IMAQ functions and briefly describes each function.

Introduction

NI-IMAQ functions are grouped according to the following classes:

- Generic functions
- High-level functions
  - Snap functions
  - Grab functions
  - Ring and sequence functions
  - Miscellaneous high-level functions
- Low-level functions
  - Interface functions
  - Session functions
  - Miscellaneous low-level functions

Generic Functions

Use generic functions in both high-level and low-level applications.

`imgInterfaceOpen`	Opens by name an interface as specified in the NI-IMAQ configuration utility, `IMAQconf`
`imgSessionOpen`	Opens a session of an unknown type and returns a session ID
`imgClose`	Closes a session or interface and unlocks and releases all buffers associated with the data type

High-Level Functions

Use high-level functions to quickly and easily capture images. If you need more advanced functionality, you can mix high-level functions with low-level functions.

Snap Functions

Snap functions program the session to capture all or a portion of a single frame or field to the user buffer.

`imgSnap`	Performs a single frame and field acquisition
`imgSnapArea`	Performs an area-specific frame or field acquisition

Grab Functions

Grab functions start a continuous image acquisition to a user buffer. Any frame or field can be copied from the grab buffer to another user buffer.

`imgGrabSetup`	Configures and optionally starts a continuous acquisition
`imgGrab`	Performs a transfer from a continuous acquisition session. Call this function only after calling `imgGrabSetup`.
`imgGrabArea`	Performs a transfer from a continuous acquisition. Call this function only after calling `imgGrabSetup`.

Ring and Sequence Functions

Ring and sequence functions start and stop a continuous acquisition of multiple fields or frames.

`imgRingSetup`	Prepares a session for acquiring continuously and looping into buffer list
`imgSequenceSetup`	Prepares a session for acquiring a full sequence into the buffer list
`imgSessionStartAcquisition`	Starts a session acquisition identified by the session ID
`imgSessionStopAcquisition`	Stops a session acquisition identified by the session ID

Miscellaneous Functions

Miscellaneous functions set and get the acquisition window's region of interest and return information such as session status and buffer sizes.

`imgSessionStatus`	Gets the current session status
`imgSessionSetROI`	Sets acquisition origin and dimension
`imgSessionGetROI`	Gets acquisition origin and dimension
`imgSessionGetBufferSize`	Gets the minimum buffer size needed for frame buffer allocation

Low-Level Functions

Use low-level functions when you require more direct hardware control.

Interface Functions

Interface functions load and control the selected IMAQ board and cameras. These functions use information stored by `IMAQconf`, the graphical IMAQ configuration utility.

`imgInterfaceLock`	Locks a logical interface so that another process cannot use it
`imgInterfaceQueryNames`	Returns the interface name identified by the index parameter
`imgInterfaceReset`	Performs a hardware reset on the interface type and returns a status, either good or bad
`imgInterfaceUnlock`	Unlocks a logical interface, allowing another process to use it

Session Functions

Use the session functions to directly control the image session, through which you set up any image acquisition, such as a sequence capture.

`imgSessionAbort`	Stops an asynchronous acquisition or synchronous continuous acquisition immediately
`imgSessionAcquire`	Starts acquisition synchronously or asynchronously to the frame buffers in the associated session buffer list
`imgSessionClearBuffer`	Clears a session's image data to the specified pixel value
`imgSessionClearTriggers`	Disables all triggers and trigger modes on the corresponding session
`imgSessionConfigure`	Specifies the buffer list to use with this session
`imgSessionCopyArea`	Copies an area of a session's buffer to a user-specified buffer
`imgSessionCopyBuffer`	Copies a session's image data to a user buffer format
`imgSessionExamineBuffer`	Extracts a buffer from a live acquisition; lets you lock a buffer out of a continuous loop sequence for processing when you are performing a ring (continuous) acquisition
`imgSessionGetTriggerStatus`	Returns a status on a specified trigger line
`imgSessionReleaseBuffer`	Releases a buffer that was previously held with `imgSessionExamineBuffer`
`imgSessionSaveBuffer`	Saves a buffer of a session to disk in native operating-specific system format such as `bmp` or `PICT`
`imgSessionSetRTSImap`	Maps the internal RTSIbus triggers to the external RTSIbus connector lines

`imgSessionSetTrigger`	Configures the specified trigger line with a drive, an action, and a polarity
`imgSessionWait`	Waits for an asynchronous acquisition to complete
`imgSessionWaitAqdone`	Waits until the acquisition done interrupt is received by the IMAQ software; ensures that the last frame or field is completely finished transferring to memory
`imgSessionWaitVblank`	Waits for the start of the next camera vertical blank before returning

Miscellaneous Functions

You can use these functions to wait for host- or interface-specific events, change acquisition parameters, create buffers, and dispose of sessions and interfaces.

`imgCameraAction`	Sends camera control information to a camera (if applicable)
`imgCreateBuffer`	Creates a user frame buffer based on the geometric definitions of the associated session
`imgCreateBufList`	Creates a buffer list that is passed to `imgSessionConfigure`
`imgDisposeBuffer`	Disposes of a user frame buffer
`imgDisposeBufList`	Purges all image buffers associated with this buffer list
`imgGetAttribute`	Returns an attribute for an interface or session
`imgGetBufferElement`	Gets an element of a specific type from a buffer list
`imgMemLock`	Locks all session-associated image buffers in memory in preparation for an acquisition
`imgMemUnlock`	Unlocks all session-associated buffers

`imgPlot`	Plots a buffer to a window given a native window handle
`imgSetArrayPointerValue`	A Visual Basic helper function for constructing an array of 32-bit pointers
`imgSetAttribute`	Sets an attribute for an interface or session
`imgSetBufferElement`	Sets a buffer list element of a given type to a specific value
`imgShowError`	Returns a null terminated string describing the error code

Chapter 4

Programming with NI-IMAQ

This chapter contains an overview of the NI-IMAQ library, a description of the programming flow of NI-IMAQ, and programming examples. Flowcharts are included for the following operations: snap, grab, ring, sequence, and snap on external trigger, which mixes high-level and low-level entry points.

Introduction

The NI-IMAQ API is divided into two groups, the high-level functions and the low-level functions. With the high-level functions, you can write programs quickly without having to learn the details of the low-level API and driver. The low-level functions give you finer granularity and control over your image acquisition process, but you must understand the API and driver in greater detail.

☞ **Note:** ***The high-level functions call low-level functions and use certain attributes that are listed in the high-level function description in the*** *NI-IMAQ Function Reference Manual for Mac OS*___. Changing the value of these attributes while using low-level functions will affect the operation of the high-level functions.___

High-Level Functions

The high-level function set supports four basic types of image acquisition:

- *Snap* acquires a single frame or field to a buffer
- *Grab* performs an acquisition that loops continually on one buffer; you obtain a copy of the acquisition buffer by *grabbing* a copy to a separate buffer that can be used for analysis
- *Ring* performs an acquisition that loops continually on a specified number of buffers
- *Sequence* performs an acquisition that acquires a specified number of buffers, then stops

Low-Level Functions

The low-level function set supports all types of acquisition and can be used to:

- Create a custom acquisition sequence or ring
- Create and manage your own buffers
- Set session and interface attributes to adjust image quality and size
- Start a synchronous or asynchronous acquisition
- Extract buffers out of a live acquisition for analysis
- Set up and control triggered acquisitions

Establishing Interface Connections and Sessions

To acquire images using the high-level or low-level functions, you must first learn how to establish a connection to an interface and create a session. See the *Interface Functions* and *Session Functions* sections in this chapter for information on how to manage interfaces and sessions, then refer to the high-level or low-level samples for information on acquiring images.

Interface Functions

Use interface functions to query the number of available interfaces, establish a connection to, control access to, and initialize hardware such as the IMAQ-1408. All interfaces in NI-IMAQ are specified by a name. By default, the system creates default names for the number of boards in your system. These names observe the convention shown in Table 4-1.

Table 4-1. Interface Naming Convention

Interface Name	Board Installed
`img0`	Board 0
`img1`	Board 1
…	…
`img`*n*	Board *n*

You can edit existing or create new interfaces by using the IMAQconf configuration utility. You also can use IMAQconf to configure the board serial number and the default state of a particular interface.

Before you can acquire image data successfully, you must open an interface by using the imgInterfaceOpen function. imgInterfaceOpen requires an interface name and returns a handle to this interface. NI-IMAQ then uses this handle to reference this interface when using other NI-IMAQ functions.

To establish a connection to the first board in your system, use the following program example:

```
INTERFACE_ID   interfaceID;
if (imgInterfaceOpen("img0", &interfaceID) == IMG_ERR_GOOD)
{
  … user code …
  imgClose(interfaceID, FALSE);
}
```

This example opens an interface to img0. When the program is finished with the interface, it closes the interface using the imgClose function.

For a complete list of the available interface functions, refer to the *NI-IMAQ Function Reference Manual for Mac OS*.

Session Functions

Use session functions to configure the type of acquisition you want to perform on a particular interface. After you have established a connection to an interface, you need to create a session and configure it to perform the type of acquisition you require.

To create a session, call the imgSessionOpen function. This function requires a valid interface handle and returns a handle to a session. NI-IMAQ then uses this session handle to reference this session when using other NI-IMAQ calls.

To create a session, use the following example program:

```
INTERFACE_ID   interfaceID;
SESSION_ID     sessionID;
if (imgInterfaceOpen(“img0”, &interfaceID) == IMG_ERR_GOOD)
{
  if (imgSessionOpen(interfaceID, &sessionID) == IMG_ERR_GOOD)
  {
    … user code …
    imgClose(sessionID, FALSE);
  }
  imgClose(interfaceID, FALSE);
}
```

This example opens an interface to `img0` and then creates a session to acquire images. When the program is finished with the interface and session, it then closes both handles using the `imgClose` function.

For a complete list of the available session functions, refer to the *NI-IMAQ Function Reference Manual for Mac OS*.

Managing Buffers

Buffer management can be performed either by you or automatically by NI-IMAQ. If the high-level acquisition routines (`imgSnap`, `imgGrab`, `imgSequenceSetup`, and `imgRingSetup`) are initiated with NULL pointers for buffer addresses, NI-IMAQ will automatically allocate a buffer and return the value of the buffer pointer to you. After you have a buffer pointer, you can use this pointer in successive calls.

For greater control of the acquisition buffers, such as creating buffers larger than the image size for adding borders, you can create them by calling a memory allocation routine (for example, `malloc`) or using the low-level function `imgCreateBuffer`. When creating buffers using either approach, dispose of the buffers using `free` or `imgDisposeBuffer` when applicable to free PC memory for maximum performance.

Introductory Programming Examples

This section introduces some examples for performing the different types of image acquisition and includes both high-level and low-level function examples. The error codes that NI-IMAQ returns are not included in the examples. In your programs, always check the return code for errors.

High-Level Snap Functions

A *snap* acquires a single image into a memory buffer. Snap functions include `imgSnap` and `imgSnapArea`. Use these functions to acquire a single frame or field to a buffer. To use these functions, you must have a valid session handle.

When you invoke a snap, it initializes the board and acquires the next incoming video frame (or field) to a buffer. A snap is appropriate for low-speed or single-capture applications where ease of programming is essential. Figure 4-1 illustrates a typical snap programming order.

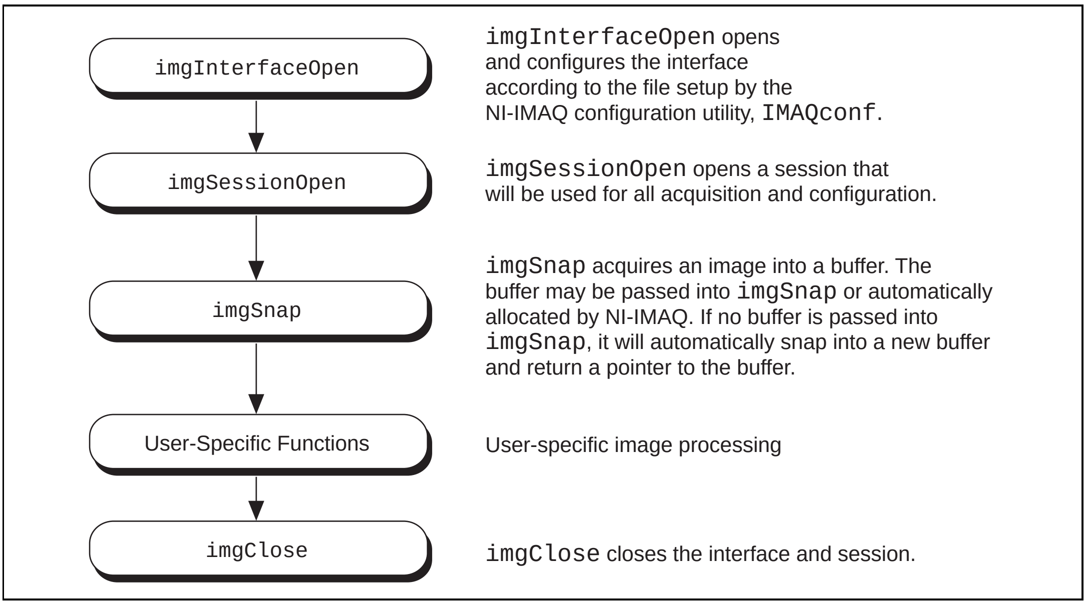

Figure 4-1. Snap Programming Flowchart

Example 1 demonstrates how to perform a single snap using `imgSnap`.

```
int main(void)
{
 INTERFACE_ID  interfaceID;
 SESSION_ID    sessionID;
 Int8*         buffer = NULL;
 IMG_ERR       error;
 uInt32        top, left, height, width, rowBytes;

 // open an interface and a session
 error = imgInterfaceOpen(“img0”, &interfaceID);
 error = imgSessionOpen(interfaceID, &sessionID);

 // pass a pointer to a NULL pointer and the driver will allocate
 // a buffer of the appropriate size for you.
 error = imgSnap(sessionID, &buffer);

 // get the image dimensions. These default dimensions depend on the type
 // of camera that is currently configured.
 error = imgSessionGetROI(sessionID, &top, &left, &height, &width);
 error = imgGetAttribute(sessionID, IMG_ATTR_ROWBYTES, &rowBytes);

 // process the image
 my_process_function(buffer, top, left, height, width, rowBytes);

 // close this interface and free all resources associated with it,
 // such as the buffer that was allocated by the driver during imgSnap
 imgClose(interfaceID, TRUE);

 return(0);
}
```

Example 1 opens an interface and a session and then performs a single snap. The buffer pointer that is passed to `imgSnap` is initialized to NULL, which instructs `imgSnap` to automatically allocate a buffer for the image. The size of the buffer is calculated based on the region of interest (ROI) and the rowByte attributes: ROI height multiplied by rowByte. When you open a session, the ROI values are initialized from the acquisition window (ACQWINDOW) dimensions that are configured in `IMAQconf`. The ACQWINDOW dimensions will vary depending on the type of camera you are using.

The sample then calls a process function to analyze the image. When the program is finished, it calls `imgClose` with the interface handle and sets the **freeResources** flag to TRUE. This instructs NI-IMAQ to free all of the resources associated with this interface, which releases the session as well as the memory buffer allocated by `imgSnap`.

High-Level Grab Functions

A *grab* is a continuous high-speed acquisition of data to a single buffer in host memory. Grab functions include `imgGrabSetup`, `imgGrab` and `imgGrabArea`. You can use these functions to perform an acquisition that loops continually on one buffer. A copy of the acquisition buffer is obtained by grabbing a copy to a separate buffer. To use these functions, you must have a valid session handle.

Calling `imgGrabSetup` initializes a session for a grab acquisition. After `imgGrabSetup`, each successive grab will copy the last acquired buffer into a user buffer where you can perform processing on the image. A grab is appropriate for high-speed applications where you need processing performed on only one image at a time. Figure 4-2 illustrates a typical grab programming order.

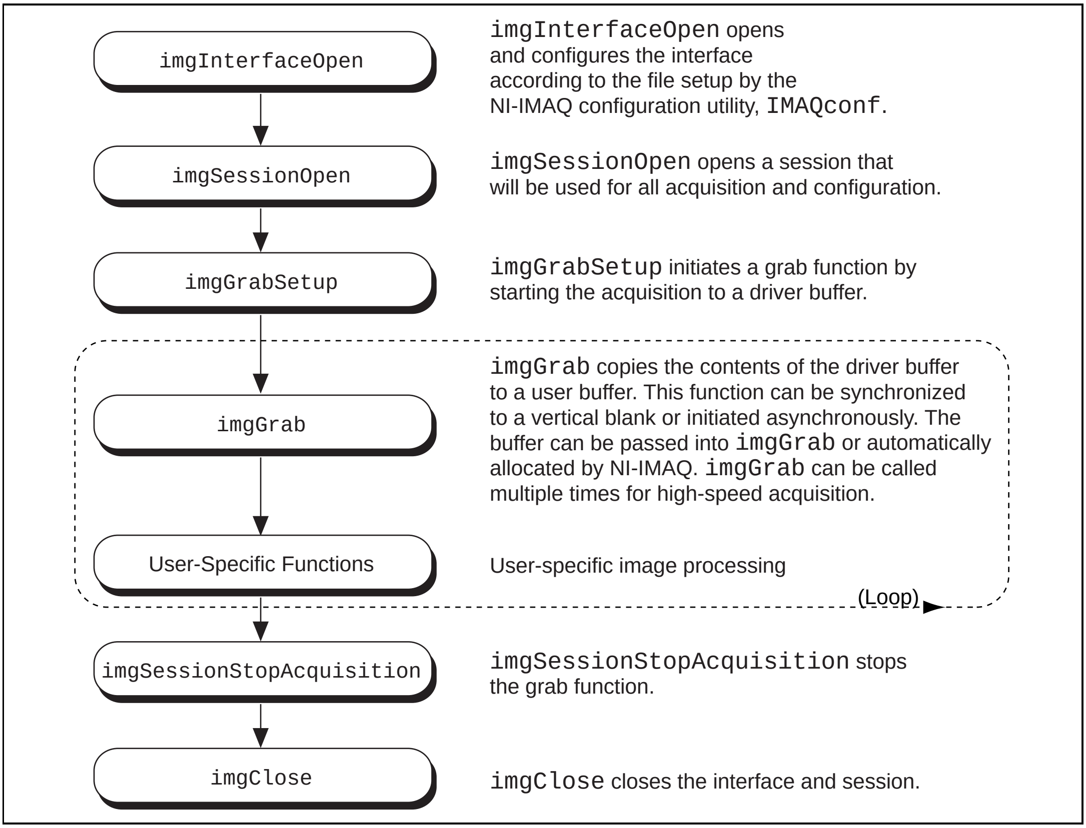

Figure 4-2. Grab Programming Flowchart

Example 2 demonstrates how to perform a grab using `imgGrabArea`.

```
int main(void)
{
  INTERFACE_ID  interfaceID;
  SESSION_ID    sessionID;
  Int8*         buffer;
  IMG_ERR       error;
  uInt32        top, left, height, width, rowBytes, bytesPerPixel;

  // open an interface and a session
  error = imgInterfaceOpen(“img0”, &interfaceID);
  error = imgSessionOpen(interfaceID, &sessionID);
```

```
  // configure the session for a grab but do not start the acquisition yet
  error = imgGrabSetup(sessionID, FALSE);

  // get the image dimensions
  error = imgSessionGetROI(sessionID, &top, &left, &height, &width);
  error = imgGetAttribute(sessionID, IMG_ATTR_ROWBYTES, &rowBytes);
  error = imgGetAttribute(sessionID, IMG_ATTR_BYTESPERPIXEL, &bytesPerPixel);

  // calculate a new grab area dimension
  height = height >> 1;
  width = width >> 1;
  rowBytes = width * bytesPerPixel;

  // start the acquisition
  error = imgSessionStartAcquisition(sessionID);

  // allocate our own buffer for storing a copy
  buffer = (Int8*) malloc(rowBytes * height);

  // grab until some condition is met
  while(1)
  {
   // grab a copy of the acquisition buffer into my own user buffer for analysis
   error = imgGrabArea(sessionID, &buffer, TRUE, top, left, height, width, rowBytes);

   // process the image, if some condition is met, then terminate
   if (my_process_function(buffer, top, left, height, width, rowBytes))
    break;

   ... calculate new dimensions and reallocate buffer if necessary ...
  }

  // stop the grab acquisition
  error = imgSessionStopAcquisition(sessionID);

  // free our user buffer
  free(buffer);

  // close this interface, free all resources
  imgClose(interfaceID, TRUE);

  return(0);
}
```

Example 2 performs multiple grabs until an appropriate condition is met. The program configures the session to perform a grab operation by calling the `imgGrabSetup` function. The program then calculates the area to grab using the current ROI, rowBytes, and BYTESPERPIXEL, and the acquisition is started by calling `imgSessionStartAcquisition`. In this example, we allocate our own user buffer for grabbing and pass this buffer to `imgGrabArea`. When the acquisition is complete, it stops. The program then frees the user buffer and all of the resources associated with this interface by calling `imgClose`.

High-Level Ring and Sequence Functions

Ring and sequence functions include `imgRingSetup`, `imgSequenceSetup`, `imgSessionStartAcquisition` and `imgStopAcquisition`. Use these functions to perform a continuous acquisition that loops or stops after a certain number of images have been captured.

A *ring* initiates a continuous high-speed acquisition to multiple buffers. Calling `imgRingSetup` initiates a ring. `imgRingSetup` specifies both the buffer list that will be used for transfers and the number of buffers. After `imgRingSetup` is called, you can monitor the status of the transfer and perform processing on any of the buffers in the ring. A ring is appropriate for high-speed applications where you need to perform processing on every image. You must use multiple buffers because processing times may vary depending on other applications and processing results. You can configure a ring to acquire every frame or to skip a fixed number of frames between each acquisition.

For certain applications, you can temporarily extract a buffer from the ring to prevent it from being overwritten during the ring's next pass. Use the `imgSessionExamineBuffer` and `imgSessionReleaseBuffer` functions to do this. Figure 4-3 illustrates a typical ring programming order.

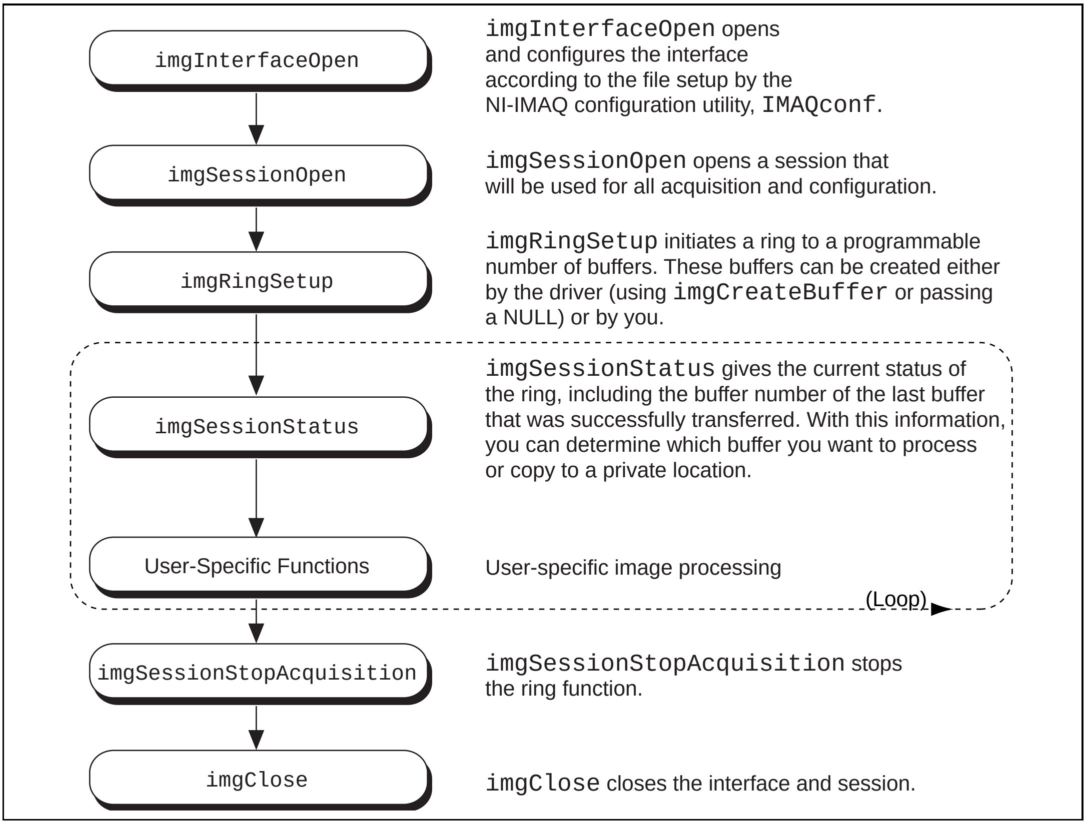

Figure 4-3. Ring Programming Flowchart

A *sequence* initiates a variable-length and variable-delay transfer to multiple buffers. You can configure the delay between acquisitions with `SequenceSetup` and specify both the buffer list that will be used for transfers and the number of buffers. After `imgSequenceSetup`, you can monitor the status of the transfer and perform processing on any of the buffers in the sequence or you can wait until the acquisition completes and process all buffers simultaneously.

A sequence is appropriate for applications where you need to perform processing on multiple images. You can configure a sequence to acquire every frame or skip a variable number of frames between each image. Figure 4-4 illustrates a typical sequence programming order.

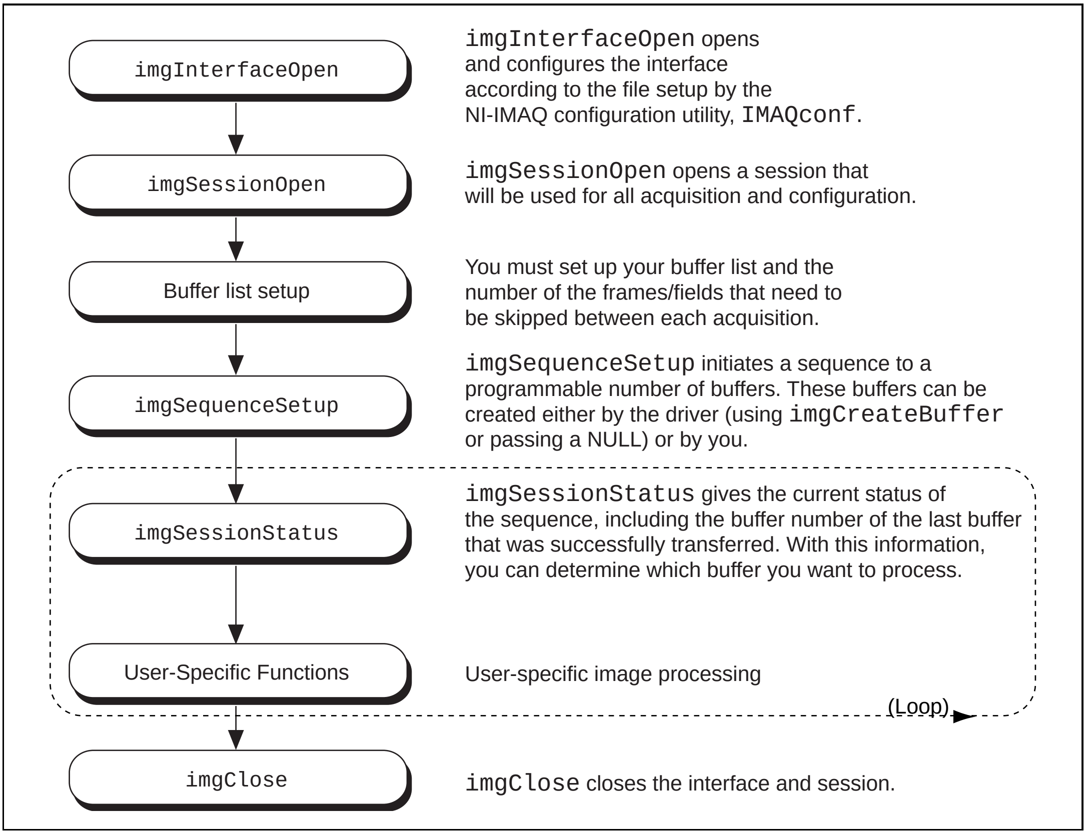

Figure 4-4. Sequence Programming Flowchart

Example 3 demonstrates how to perform a ring acquisition using `imgRingSetup`.

```
int main(void)
{
  INTERFACE_ID  interfaceID;
  SESSION_ID    sessionID;
  uInt32        top, left, height, width, rowBytes;
  Int32         error = 0;
  uInt32        status, lastBufNum, currBufNum;
  Int8*         buffers[6];
```

```
  // open an interface and a session
  error = imgInterfaceOpen("img0", &interfaceID);
  error = imgSessionOpen(interfaceID, &sessionID);

  // initialize buffer pointers to NULL
  memset(buffers, 0x00, sizeof(buffers));

  // configure the session for a ring with six buffers; have the driver allocate the
  // buffers
  // only acquire every third video frame and do not start the acquisition yet
  error = imgRingSetup(sessionID, 6, (void**)buffers, 3, FALSE);

  // get the image dimensions
  error = imgSessionGetROI(sessionID, &top, &left, &height, &width);
  error = imgGetAttribute(sessionID, IMG_ATTR_ROWBYTES, &rowBytes);

  // start the acquisition
  error = imgSessionStartAcquisition(sessionID);

  // run until some condition is met
  lastBufNum = currBufNum = 0xFFFFFFFF;
  while(1)
  {
    // spin waiting for the most current buffer
    // until the first image is acquired, imgSessionStatus will return 0xFFFFFFFF for
    // the current buffer number
    while(lastBufNum == currBufNum)
      imgSessionStatus(sessionID, &status, &currBufNum);

    // the buffer number returned will be in the range from 0 to 5
    lastBufNum = currBufNum;

    // process the image, if some condition then terminate
    if (my_process_function(buffers[currBufNum], top, left, height, width, rowBytes))
      break;
  }

  // stop the ring acquisition
  imgSessionStopAcquisition(sessionID);

  // close this interface, free all resources
  imgClose(interfaceID, TRUE);

  return(0);
}
```

Example 3 sets up a ring containing six buffers and sets the skip count to three, which causes the program to acquire on every third frame. A *skip count* is the number of frames skipped prior to acquiring an image to a buffer. The program then loops, waiting for the next buffer to be acquired. The `imgSessionStatus` function queries NI-IMAQ for the buffer number of the last valid buffer that has been acquired. The last valid buffer is defined as the buffer that contains the most recent video image. This process will continue until a designated condition is met and then the acquisition stops.

Example 4 demonstrates how to perform a sequence acquisition using `imgSequenceSetup`.

```
int main(void)
{
  INTERFACE_ID  interfaceID;
  SESSION_ID    sessionID;
  uInt32        i, top, left, height, width, rowBytes, bytesPerPixel;
  Int32         error = 0;
  Int8*         buffers[10];
  uInt32        skips[10];

  // open an interface and a session
  error = imgInterfaceOpen("img0", &interfaceID);
  error = imgSessionOpen(interfaceID, &sessionID);

  // get the image dimensions
  error = imgSessionGetROI(sessionID, &top, &left, &height, &width);
  error = imgGetAttribute(sessionID, IMG_ATTR_ROWBYTES, &rowBytes);
  error = imgGetAttribute(sessionID, IMG_ATTR_BYTESPERPIXEL, &bytesPerPixel);

  // calculate a new ROI
  height = height >> 1;
  width = width >> 1;
  rowBytes = width * bytesPerPixel;
  top += 10;
  left += 10;

  // now reset the ROI and the rowBytes
  // these attributes are used by imgSequenceSetup
  imgSessionSetROI(sessionID, top, left, height, width);
  imgSetAttribute(sessionID, IMG_ATTR_ROWBYTES, rowBytes);
```

```
  // initialize buffer pointers and skip count arrays
  // we will give a skip count equal to i, skip 1 frame, 2 frames, 3 frames, and so on
  for (i = 0; i < 10; i++)
  {
    buffers[i] = (Int8*) malloc(rowBytes * height);
    skips[i] = i;
  }

  // configure the session for a sequence with 10 buffers using buffers that we have
  // allocated
  // start the acquisition now and signal that the call is synchronous
  error = imgSequenceSetup(sessionID, 10, (void**)buffers, skips, TRUE, FALSE);

  // now call our analysis function for each buffer
  for (i = 0; i < 10; i++)
    my_process_function(buffers[i], top, left, height, width, rowBytes);

  // free our buffers
  for (i = 0; i < 10; i++)
    free(buffers[i]);

  // close this interface, free all resources
  imgClose(interfaceID, TRUE);

  return(0);
}
```

Example 4 sets up a sequence that uses 10 user-allocated buffers. Unlike the ring example, each buffer in the sequence has its own skip count associated with it. The skip count is the number of frames to skip prior to acquiring the next image. The acquisition is started at setup time and the setup call is synchronous.

Performing a Snap Using Low-Level Functions

Example 5 demonstrates how to perform a snap acquisition using low-level calls.

```
int main(void)
{
  uInt32        width, height;
  uInt32        bufSize;
  INTERFACE_ID  interfaceID = 0;
  SESSION_ID    sessionID = 0;
  BUFLIST_ID    buflistID = 0;
  Int8*         buffer = NULL;
  Int32         error = 0;

  // open an interface and a session
  error = imgInterfaceOpen(“img0”, &interfaceID);
  error = imgSessionOpen(interfaceID, &sessionID);

  // create a buffer list with one element
  error = imgCreateBufList(1, &buflistID);

  // these attributes default to the maximum width and height for the
  // default camera
  error = imgGetAttribute(sessionID, IMG_ATTR_ACQWINDOW_WIDTH, &width);
  error = imgGetAttribute(sessionID, IMG_ATTR_ACQWINDOW_HEIGHT, &height);

  // make the width longword aligned
  width = ((width + 3) & 0xFFFFFFFC);

  // set acquisition window width and rowBytes to the value */
  error = imgSetAttribute(sessionID, IMG_ATTR_ACQWINDOW_WIDTH, width);
  error = imgSetAttribute(sessionID, IMG_ATTR_ROWBYTES, width);

  // set the ROI width to the same
  error = imgSetAttribute(sessionID, IMG_ATTR_ROI_WIDTH, width);

  // calculate buffer size needed for acquisition buffer
  bufSize = width * height;

  // create a buffer and configure the buffer list
  error = imgCreateBuffer(sessionID, FALSE, bufSize, &buffer);

  // the next three calls assigns the following to buffer list element 0:
  //   1) buffer pointer that will contain image
  //   2) size of the buffer for buffer element 0
  //   3) command to stop acquisition when this element is reached
  error = imgSetBufferElement(buflistID, 0, IMG_BUFF_ADDRESS, (uInt32)buffer);
  error = imgSetBufferElement(buflistID, 0, IMG_BUFF_SIZE, bufSize);
```

```
  error = imgSetBufferElement(buflistID, 0, IMG_BUFF_COMMAND, IMG_CMD_STOP);

  // lock down the buffers contained in the buffer list
  error = imgMemLock(buflistID);

  // configure the session to use this buffer list */
  error = imgSessionConfigure(sessionID, buflistID);

  // start the acquisition, synchronous, no callback function
  error = imgSessionAcquire(sessionID, FALSE, NULL);

  // process the image
  my_process_function(buffer, ...);

  // unlock the buffers in the buffer list
  imgMemUnlock(buflistID);

  // dispose of the buffer
  imgDisposeBuffer(buffer);

  // close this buffer list
  imgDisposeBufList(buflistID, FALSE);

  // close session
  imgClose(sessionID, FALSE);

  // close interface
  imgClose(interfaceID, FALSE);

  return(0);
}
```

Example 5 sets up a single-frame acquisition to a buffer allocated by NI-IMAQ. The program retrieves the acquisition window width of the selected camera and aligns it on a 32-bit boundary. You must align both the acquisition window width and rowBytes on a 32-bit boundary to ensure that your image is acquired properly. The software does not perform this alignment for you unless you select a scaling option. Although `IMAQconf` performs this alignment for you when you acquire an image with it, you must perform the alignment yourself if you use window widths not aligned on a 32-bit boundary.

After the program sets the ROI, it locks the memory and acquires the image. If you choose to plot the image using the `imgPlot` function, you must align the image width on a 32-bit boundary as well.

Performing a Grab Using Low-Level Functions

Example 6 demonstrates how to perform a grab acquisition using low-level calls.

```
int main(void)
{
 uInt32       width, height;
 uInt32       bufSize, bufNum;
 INTERFACE_ID interfaceID = 0;
 SESSION_ID   sessionID = 0;
 BUFLIST_ID   buflistID = 0;
 Int8*        buffer = NULL;
 uInt8*       copyBuffer = NULL;
 Int32        error = 0;

 // open an interface and a session
 error = imgInterfaceOpen(“img0”, &interfaceID);
 error = imgSessionOpen(interfaceID, &sessionID);

 // create a buffer list with one element
 error = imgCreateBufList(1, &buflistID);

 // these attributes default to the maximum width and height for the
 // default camera
 error = imgGetAttribute(sessionID, IMG_ATTR_ACQWINDOW_WIDTH, &width);
 error = imgGetAttribute(sessionID, IMG_ATTR_ACQWINDOW_HEIGHT, &height);

 // make the width longword aligned.
 width = ((width + 3) & 0xFFFFFFFC);

 // set acquisition window width and rowBytes to the value
 error = imgSetAttribute(sessionID, IMG_ATTR_ACQWINDOW_WIDTH, width);
 error = imgSetAttribute(sessionID, IMG_ATTR_ROWBYTES, width);

 // set the ROI width to the same
 error = imgSetAttribute(sessionID, IMG_ATTR_ROI_WIDTH, width);

 // calculate buffer size needed for acquisition buffer
 bufSize = width * height;

 // allocate our own buffer for storing copy
 copyBuffer = malloc(bufSize);

 // create a buffer and configure the buffer list
 error = imgCreateBuffer(sessionID, FALSE, bufSize, &buffer);
```

```
// the next three calls assigns the following to buffer list element 0:
//    1) buffer pointer that will contain image
//    2) size of the buffer for buffer element 0
//    3) command to stop acquisition when this element is reached

error = imgSetBufferElement(buflistID, 0, IMG_BUFF_ADDRESS, (uInt32)buffer);
error = imgSetBufferElement(buflistID, 0, IMG_BUFF_SIZE, bufSize);
error = imgSetBufferElement(buflistID, 0, IMG_BUFF_COMMAND, IMG_CMD_LOOP);

// lock down the buffers contained in the buffer list
error = imgMemLock(buflistID);

// configure the session to use this buffer list
error = imgSessionConfigure(sessionID, buflistID);

// start the acquisition, asynchronous
error = imgSessionAcquire(sessionID, TRUE, NULL);

// grab until some condition is met
while(1)
{
  // save a copy to copyBuffer, wait for VBLANK period
  error = imgSessionCopyBuffer(sessionID, 0, copyBuffer, TRUE);

  // process the image, if some condition is met, then terminate
  if (my_process_function(copyBuffer, ...))
    break;
}

// stop the acquisition
imgSessionAbort(sessionID, &bufNum);

// unlock the buffers in the buffer list
imgMemUnlock(buflistID);

// dispose of the buffer
imgDisposeBuffer(buffer);

// close this buffer list
imgDisposeBufList(buflistID, FALSE);
```

```
  // close this session
  imgClose(sessionID, FALSE);

  // close interface handle
  imgClose(interfaceID, FALSE);

  // free our copy buffer
  free(copyBuffer);

  return(0);
}
```

Example 6 sets up a continuous acquisition to a single user-allocated buffer. As described in Example 5, the program retrieves the acquisition window width of the selected camera and aligns it on a 32-bit boundary. The program creates a buffer list to describe the acquisition buffers. Next, the program sets the ROI to the acquisition window width. The program performs a calculation to determine the correct memory requirements of the user buffer. The program creates the buffer and configures buffer element 0 for a single continuous acquisition. the program then locks the memory and starts the image acquisition asynchronously. The main processing loop of the code shows how to wait for vertical blank and copy the buffer to an analysis buffer.

Keep your analysis code fast to minimize the number of missed frames during analysis. If you need more time to examine a buffer, set up a multiple-buffer ring and call `imgSessionExamineBuffer` to extract the desired buffer from the live sequence.

Performing a Ring Acquisition Using Low-Level Functions

Example 7 demonstrates how to perform a ring acquisition using low-level calls.

```
int main(void)
{

  uInt32        top, left, width, height, rowBytes, i;
  uInt32        bufSize;
  INTERFACE_ID  interfaceID = 0;
  SESSION_ID    sessionID = 0;
  BUFLIST_ID    buflistID = 0;
  uInt32        lastBufNum, currBufNum, bufCmd, bufNum;
  Int8*         buffers[6];
  Int32         error = 0;

  // initialize buffer pointers to NULL
  memset(buffers, 0x00, sizeof(buffers));

  // open an interface and a session
  error = imgInterfaceOpen(“img0”, &interfaceID);
  error = imgSessionOpen(interfaceID, &sessionID);

  // create a buffer list with 6 elements
  error = imgCreateBufList(6, &buflistID);

  // these attributes default to the maximum width and height for the
  // default camera
  error = imgGetAttribute(sessionID, IMG_ATTR_ACQWINDOW_WIDTH, &width);
  error = imgGetAttribute(sessionID, IMG_ATTR_ACQWINDOW_HEIGHT, &height);

  // make the width longword aligned
  width = ((width + 3) & 0xFFFFFFFC);

  // set acquisition window width and rowBytes to the value
  error = imgSetAttribute(sessionID, IMG_ATTR_ACQWINDOW_WIDTH, width);
  error = imgSetAttribute(sessionID, IMG_ATTR_ROWBYTES, width);

  // set the ROI width to the same
  error = imgSetAttribute(sessionID, IMG_ATTR_ROI_WIDTH, width);

  // calculate buffer size needed for acquisition buffer
  bufSize = width * height;
```

```
 // the next set of functions assign the following to the buffer list elements:
 //    1) buffer pointer that will contain image
 //    2) size of the buffer for each buffer element
 //    3) command to go to next buffer or loop when the last element is reached

 for (i = 0; i < 6; i++)
{
 error = imgCreateBuffer(sessionID, FALSE, bufSize, &buffers[i]);
 error = imgSetBufferElement(buflistID, i, IMG_BUFF_ADDRESS, (uInt32)buffers[i]);
 error = imgSetBufferElement(buflistID, i, IMG_BUFF_SIZE, bufSize);
 bufCmd = (i == 5) ? IMG_CMD_LOOP : IMG_CMD_NEXT;
 error = imgSetBufferElement(buflistID, i, IMG_BUFF_COMMAND, bufCmd);
}

 // lock down the buffers contained in the buffer list
 error = imgMemLock(buflistID);

 // configure the session to use this buffer list
 error = imgSessionConfigure(sessionID, buflistID);

 // start the acquisition, asynchronous
 error = imgSessionAcquire(sessionID, TRUE, NULL);

 // run until user wants to stop */
 lastBufNum = currBufNum = 0xFFFFFFFF;
 while(1)
 {

   // spin waiting for the most current buffer
   // until the first image is acquired, imgSessionStatus will return 0xFFFFFFFF for
   // the current buffer number

   while (lastBufNum == currBufNum)
     imgGetAttribute(sessionID, IMG_ATTR_LAST_VALID_BUFFER, &currBufNum);

   lastBufNum = currBufNum;

   // process the image, if some condition then terminate
   if (my_process_function(buffers[currBufNum], top, left, height, width, rowBytes))
     break;
 }

 // stop the acquisition
 error = imgSessionAbort(sessionID, &bufNum);
```

```
  // unlock the buffers in the buffer list
  error = imgMemUnlock(buflistID);

  // dispose of the buffers
  for (i = 0; i < 6; i++)
  imgDisposeBuffer(buffers[i]);

  // close this buffer list
  error = imgDisposeBufList(buflistID, FALSE);

  // close this session
  error = imgClose(sessionID, FALSE);

  // close interface handle
  error = imgClose(interfaceID, FALSE);

  return(0);
}
```

Example 7 sets up a continuous acquisition to multiple buffers allocated by NI-IMAQ. As described in Example 5, the program retrieves the acquisition window width of the selected camera and aligns it on a 32-bit boundary. It then creates a buffer list to describe the acquisition buffers. Next, the program sets the ROI to the acquisition window width. The program calculates the correct memory requirements of the frame buffer. However, this is not necessary if you choose to use the default acquisition window width, rowBytes, and ROI. In this case, NI-IMAQ will allocate the correct size buffer if you pass a NULL as the size parameter to `imgCreateBuffer`. The buffer is created and the buffer list is configured for each buffer element in the ring. The memory is locked and the image acquisition is started asynchronously.

The main processing loop of the code shows how to wait for the first buffer to be filled and subsequently processed. NI-IMAQ returns a value of `0xFFFFFFFF` as the IMG_ATTR_LAST_VALID_BUFFER attribute until the successful acquisition of the first buffer. To guarantee that you wait for the acquisition of a new buffer in a ring with more than one buffer, you can loop on the attribute IMG_ATTR_LAST_VALID_BUFFER until it changes. If your buffer analysis requires many computations, call `imgSessionExamineBuffer` to extract the desired buffer from the live sequence. When using `imgSessionExamineBuffer`, the buffer requested is literally pulled from the looping sequence for the duration of the analysis. Use `imgSessionReleaseBuffer` to return the buffer to the continuous sequence.

Performing a Sequence Acquisition Using Low-Level Functions

Example 8 demonstrates how to perform a sequence acquisition using low-level calls.

```
int main(void)
{
  uInt32        width, height, i;
  uInt32        bufSize;
  INTERFACE_ID  interfaceID = 0;
  SESSION_ID    sessionID = 0;
  BUFLIST_ID    buflistID = 0;
  Int32         error = 0;
  uInt32        bufCmd;
  Int8*         buffers[10];
  uInt32        skips[10];


  // initialize buffer pointers and skip count arrays
  for (i = 0; i < 10; i++)
  {
    buffers[i] = NULL;
    skips[i] = i;
  }

  // open an interface and a session
  error = imgInterfaceOpen("img0", &interfaceID);
  error = imgSessionOpen(interfaceID, &sessionID);

  // create a buffer list with n elements */
  error = imgCreateBufList(NUM_SEQ_BUFFERS, &buflistID);

  // these attributes default to the maximum width and height for the
  // default camera
  error = imgGetAttribute(sessionID, IMG_ATTR_ACQWINDOW_WIDTH, &width);
  error = imgGetAttribute(sessionID, IMG_ATTR_ACQWINDOW_HEIGHT, &height);

  // make the width longword aligned
  width = ((width + 3) & 0xFFFFFFFC);

  // set acquisition window width and rowBytes to the value
  error = imgSetAttribute(sessionID, IMG_ATTR_ACQWINDOW_WIDTH, width);
  error = imgSetAttribute(sessionID, IMG_ATTR_ROWBYTES, width);
```

```
// set the ROI width to the same
error = imgSetAttribute(sessionID, IMG_ATTR_ROI_WIDTH, width);

// calculate buffer size needed for acquisition buffer
bufSize = width * height;

// the next set of functions assign the following to the buffer list elements:
//    1) buffer pointer that will contain image
//    2) size of the buffer for each buffer element
//    3) command to go to next buffer or loop when the last element is reached

for (i = 0; i < 10; i++)
{
  error = imgCreateBuffer(sessionID, FALSE, bufSize, &buffers[i]);
  error = imgSetBufferElement(buflistID, i, IMG_BUFF_ADDRESS,(uInt32)buffers[i]);
  error = imgSetBufferElement(buflistID, i, IMG_BUFF_SIZE, bufSize);
  bufCmd = (i == 9) ? IMG_CMD_STOP : IMG_CMD_NEXT;
  error = imgSetBufferElement(buflistID, i, IMG_BUFF_COMMAND, bufCmd);
}

// lock down the buffers contained in the buffer list
error = imgMemLock(buflistID);

// configure the session to use this buffer list
error = imgSessionConfigure(sessionID, buflistID);

// start the acquisition, synchronous
error = imgSessionAcquire(sessionID, FALSE, NULL);

// now call our analyses function for each buffer
for (i = 0; i < 10; i++)
  my_process_function(buffers[i], ...);

// unlock the buffers in the buffer list
error = imgMemUnlock(buflistID);

// dispose of the buffers
for (i = 0; i < 10; i++)
  imgDisposeBuffer(buffers[i]);

// close this buffer list
error = imgDisposeBufList(buflistID, FALSE);

// close this session
error = imgClose(sessionID, FALSE);
```

```
    // close interface handle
    error = imgClose(interfaceID, FALSE);

    return(0);
}
```

Example 8 sets up a sequence acquisition to multiple buffers allocated by NI-IMAQ. As described in Example 5, the program retrieves the acquisition window width of the selected camera and aligns it on a 32-bit boundary. It creates a buffer list to describe the acquisition buffers. Next, the program sets the ROI to the acquisition window width. The program calculates the correct memory requirements of the frame buffer. However, this is not necessary if you choose to use the default acquisition window width, rowBytes, and ROI. In this case, NI-IMAQ will allocate the correct size buffer if you pass a NULL as the size parameter to `imgCreateBuffer`. The program creates the buffer and configures the buffer list for each buffer element in the ring. The program locks the memory and starts the image acquisition asynchronously.

The main processing loop of the code shows how to process each buffer acquired in sequential order.

Snap on Trigger Programming

Snap acquires a single image into a memory buffer. As shown in Figure 4-5, the program will not initiate the snap until a rising edge of external trigger 0 occurs.

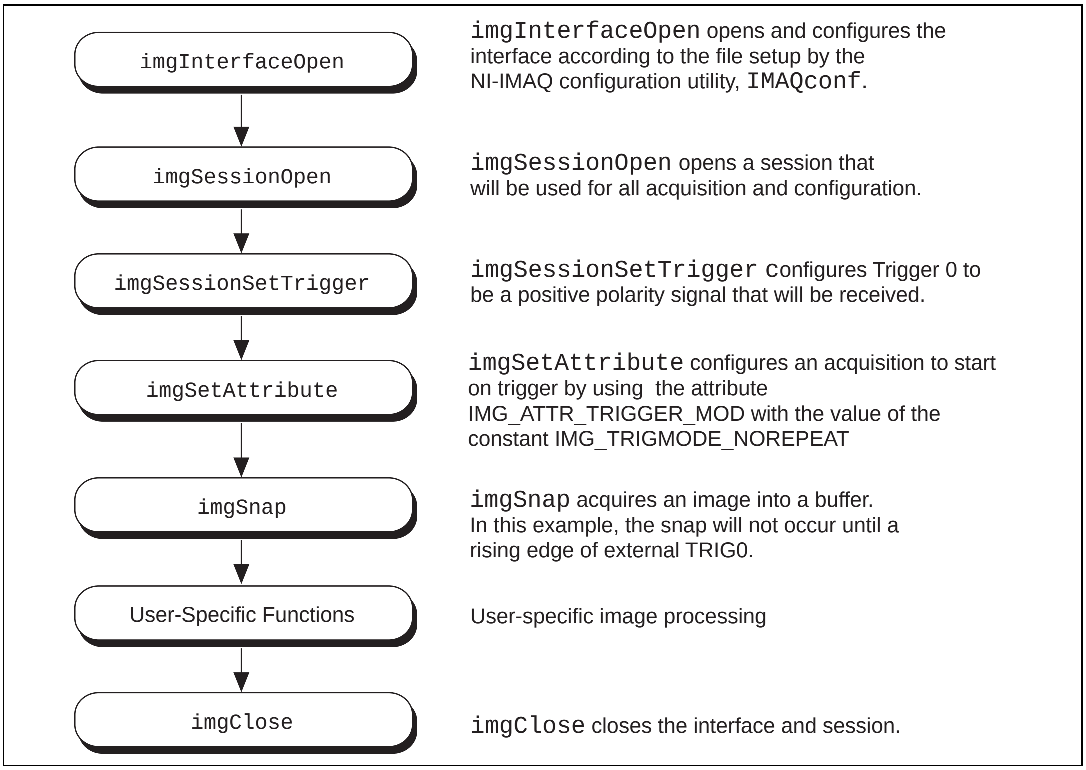

Figure 4-5. Snap on Trigger Programming Flowchart

Appendix

Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve your technical problems and a form you can use to comment on the product documentation. When you contact us, we need the information on the Technical Support Form and the configuration form, if your manual contains one, about your system configuration to answer your questions as quickly as possible.

National Instruments has technical assistance through electronic, fax, and telephone systems to quickly provide the information you need. Our electronic services include a bulletin board service, an FTP site, a Fax-on-Demand system, and e-mail support. If you have a hardware or software problem, first try the electronic support systems. If the information available on these systems does not answer your questions, we offer fax and telephone support through our technical support centers, which are staffed by applications engineers.

Electronic Services

Bulletin Board Support

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions. From these sites, you can also download the latest instrument drivers, updates, and example programs. For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call (512) 795-6990. You can access these services at:

United States: (512) 794-5422
Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 01 48 65 15 59
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

FTP Support

To access our FTP site, log on to our Internet host, `ftp.natinst.com`, as `anonymous` and use your Internet address, such as `joesmith@anywhere.com`, as your password. The support files and documents are located in the `/support` directories.

Fax-on-Demand Support

Fax-on-Demand is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access Fax-on-Demand from a touch-tone telephone at (512) 418-1111.

E-Mail Support (currently U.S. only)

You can submit technical support questions to the applications engineering team through e-mail at the Internet address listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

`support@natinst.com`

Telephone and Fax Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

	Telephone	Fax
Australia	03 9879 5166	03 9879 6277
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	09 725 725 11	09 725 725 55
France	01 48 14 24 24	01 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Israel	03 5734815	03 5734816
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	5 520 2635	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
U.K.	01635 523545	01635 523154

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name ____

Company ____

Address ____

____

Fax (___)____________________ Phone (___) ______________________________

Computer brand ________________ Model ________________ Processor____________________

Operating system (include version number) ____

Clock speed ______MHz RAM _____MB Display adapter ___________________________

Mouse ___yes ___no Other adapters installed _______________________________________

Hard disk capacity _____MB Brand ____

Instruments used ____

____

National Instruments hardware product model __________ Revision ________________________

Configuration ____

National Instruments software product ___________________________ Version ____________

Configuration ____

The problem is: ____

____

____

____

____

List any error messages: ____

____

____

The following steps reproduce the problem:____

____

____

____

____

____

NI-IMAQ Hardware and Software Configuration Form

Record the settings and revisions of your hardware and software on the line to the right of each item. Complete a new copy of this form each time you revise your software or hardware configuration, and use this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

National Instruments Products

IMAQ hardware ____

Interrupt level of hardware ____

DMA channels of hardware ____

Base I/O address of hardware ____

Programming choice ____

NI-IMAQ, IMAQ Vision, or LabVIEW version ____

Other boards in system ____

Base I/O address of other boards ____

DMA channels of other boards ____

Interrupt level of other boards ____

Other Products

Computer make and model ____

Microprocessor ____

Clock frequency or speed ____

PCI chipset ____

Type of video board installed ____

Operating system version ____

Operating system mode ____

Programming language ____

Programming language version ____

Other boards in system ____

Base I/O address of other boards ____

DMA channels of other boards ____

Interrupt level of other boards ____

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: *NI-IMAQ User Manual for Mac OS*

Edition Date: June 1997

Part Number: 371641A-01

Please comment on the completeness, clarity, and organization of the manual.

____

____

____

____

____

____

____

If you find errors in the manual, please record the page numbers and describe the errors.

____

____

____

____

____

____

____

Thank you for your help.

Name ____

Title ____

Company ____

Address ____

____

Phone (___)_____________________ Fax (___) _____________________

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway
Austin, TX 78730-5039

Fax to: Technical Publications
National Instruments Corporation
(512) 794-5678

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}

Numbers/Symbols

%	percent
+	positive of, or plus
+5V	5 V signal
-	negative of, or minus
/	per
±	plus or minus
Ω	ohm

A

A	amperes
AC	alternating current
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 VSYNC) and a line count
active pixel region	the region of pixels actively being stored; defined by a pixel start (relative to HSYNC) and a pixel count
A/D	analog-to-digital
ADC	analog-to-digital converter—an electronic device, often an integrated circuit, that converts an analog voltage to a digital number
address	character code 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
AQ_DONE	signals that the acquisition of a frame or field is completed
AQ_IN_PROGRESS	signals that the acquisition of video data is in progress
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 a specific customer
aspect ratio	the ratio of a signal’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 used to denote 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 sync signal 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	the 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 color video signals
chrominance	the color information in a video signal
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

CSYNC	composite sync signal; a combination of the horizontal and vertical sync pulses
CSYNCIN	composite sync in signal
CSYNCOUT	composite sync out signal

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=20log10 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) that means *use the current default setting*.
DIN	Deutsche Industrie Norme
DMA	direct memory access—a method by which data can be transferred to and from computer memory from and to 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
drivers	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 dB

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. For example, an analog input FIFO stores the results of A/D conversions until the data can be retrieved into system memory, a process that requires the servicing of interrupts and often the programming of the DMA controller. This process can take several milliseconds in some cases. During this time, data accumulates in the FIFO for future retrieval.
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 sync
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; examples of functions are:

y = COS (x)
status = `AO_config`(**board, channel, range**)

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 — circuitry that aligns the video timing signals by locking together the horizontal, vertical, and color subcarrier frequencies and phases and generates a pixel clock to clock pixel data into memory for display or into another circuit for processing

GND — ground signal

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, cables, and so on

HSYNC — horizontal sync 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

HSYNCIN — horizontal sync input signal

Hz — hertz—the number of scans read or updates written per second

I

IC	integrated circuit
ID	identification
IEEE	Institute of Electrical and Electronics Engineers
`IMAQconf`	a configuration and diagnostic utility included with IMAQ devices
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 controls 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
I/O	input/output—the transfer of data to/from a computer system involving communications channels, operator interface devices, and/or data acquisition and control interfaces
IRQ	interrupt request

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	look-up table—a selection in the `IMAQconf` configuration utility that contains formulas that let you implement simple imaging operations such as contrast enhancement, data inversion, gamma manipulation, or other nonlinear transfer functions

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	megabytes 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; 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; it is achieving widespread acceptance as a standard for PCs and workstations and offers a theoretical maximum transfer rate of 132 Mbytes/s
PCLK	pixel clock signal—times the sampling of pixels on a video line

PCLKIN	pixel clock in signal
PFI	programmable function input
PGIA	programmable gain instrumentation amplifier
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 HYSNCs; 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, for example, an incoming HSYNC signal
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	the smallest signal increment that can be detected by a measurement system; resolution can be expressed in bits, in proportions, or in percent of full scale. For example, a system has 12-bit resolution, one part in 4,096 resolution, and 0.0244 percent of full scale.
ribbon cable	a flat cable in which the conductors are side by side
ROI	region-of-interest; 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
RTSIbus	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
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
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	the 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
TRIG	trigger signal
trigger	any event that causes or starts some form of data capture
trigger control and mapping circuitry	circuitry that routes, monitors, and drives the external and RTSIbus trigger lines; you can configure each of these lines to start or stop acquisition on a rising or falling edge.
TTL	transistor-transistor logic

V

V	volts
VCO	voltage-controlled oscillator—an oscillator that changes frequency depending on a control signal; used in a PLL 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 HSYNC, back porch, active pixel region, and a front porch
VSYNC	vertical sync signal—the synchronization pulse generated at the beginning of each video field that tells the video monitor when to start a new field
VSYNCIN	vertical sync in signal

W

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

Index

I

L

M

N

P

R

S

T