IMAQ™

NI-IMAQ™ User Manual

Image Acquisition Software

NATIONAL INSTRUMENTS™

June 2001 Edition
Part Number 370160B-01

Important Information

Conventions

The following conventions are used in this manual:

💡	This icon denotes a tip, which alerts you to advisory information.
📝	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.
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, procedures, 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 5
Programming with ActiveX Controls

Appendix A
Color Basics and StillColor

Appendix B
Variable Height Acquisition

Appendix C
Technical Support Resources

Glossary

Index

1 Introduction to NI-IMAQ

This chapter describes the NI-IMAQ software, lists the application development environments compatible with NI-IMAQ, describes the fundamentals of creating NI-IMAQ applications for Windows 2000/NT and Windows Me/98/95, describes the files used to build these applications, and tells you where to find sample programs.

About the NI-IMAQ Software

Thank you for buying a National Instruments image acquisition (IMAQ) device, 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 manager that lets you 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 performing tasks ranging from simple device initialization to advanced high-speed image acquisition. The services you need for your application depends on the types of IMAQ devices you have and the complexity of your application.

Application Development Environments

This release of NI-IMAQ supports the following Application Development Environments (ADEs) for Windows 2000/NT and Windows Me/98/95:

- LabVIEW version 5.1 and higher
- LabWindows/CVI version 5.0 and higher
- Borland C++ Builder 3.0 and higher
- Microsoft Visual C/C++ version 6.0 and higher
- Microsoft Visual Basic version 5.0 and higher

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

If you are using Visual Basic, NI-IMAQ supports the CWIMAQ control in IMAQ Vision for Measurement Studio. For more information on the CWIMAQ control, see Chapter 5, *Programming with ActiveX Controls*.

Fundamentals of Building Applications with NI-IMAQ

Architecture

A block diagram of the NI-IMAQ architecture shown in Figure 1-1 illustrates the low- and mid-level architecture for IMAQ devices.

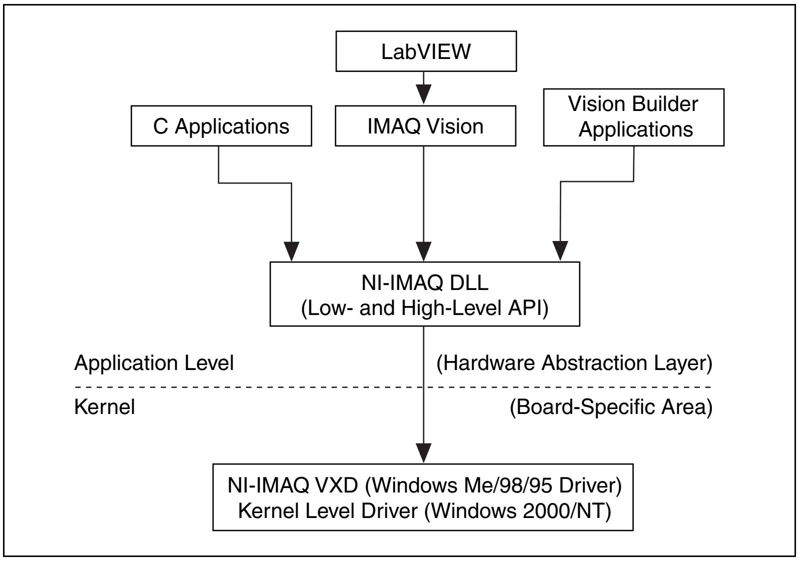
LabVIEW
C Applications
IMAQ Vision
Vision Builder Applications
NI-IMAQ DLL (Low- and High-Level API)
Application Level
(Hardware Abstraction Layer)
Kernel
(Board-Specific Area)
NI-IMAQ VXD (Windows Me/98/95 Driver)
Kernel Level Driver (Windows 2000/NT)

Figure 1-1. NI-IMAQ Architecture

The architecture uses a *hardware abstraction layer,* which separates software API capabilities, such as general acquisition and control functions, from hardware-specific information. This layer lets you use new IMAQ hardware without having to recompile your applications.

The NI-IMAQ Libraries

The NI-IMAQ for Windows 2000/NT/Me/9*x* function libraries are dynamic link libraries (DLLs), which means that NI-IMAQ routines are not linked into the executable files of applications. Only the information about the NI-IMAQ routines in the NI-IMAQ import libraries is stored in the executable files.

Import libraries contain information about their DLL-exported functions. They indicate the presence and location of the DLL routines. Depending on the development tools you are using, you may give the DLL routines information through import libraries or through function declarations. Your NI-IMAQ software kit contains function prototypes for all routines.

Creating an Application

This section outlines the process for developing NI-IMAQ applications using C for Windows 2000/NT/Me/9*x*. 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:

- Include the `NIIMAQ.H` header file in all C source files that use NI-IMAQ functions. Add this file to the top of your source files.
- Add the `IMAQ.LIB` import library to your project. Some environments allow you to add import libraries simply by inserting them into your list of project files. Other environments allow you to specify import libraries under the linker settings portion of the project file.
- When compiling, indicate where the compiler can find the NI-IMAQ header files and shared libraries. You can find most of the files you need for development under the NI-IMAQ target installation directory. If you choose the default directory during installation, the target installation directory is `C:\Program Files\National Instruments\NI-IMAQ`. You can find the include files under the `include` subdirectory. The import libraries are located under the `lib\<environment>` subdirectory for the following platforms:

Table 1-1. Import Libraries

Development Environment	Directory
Microsoft Visual C++	`lib\msvc`
Borland C++	`lib\Borland`

Sample Programs

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

2

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
  - Signal I/O functions
  - Miscellaneous functions
- Low-level functions
  - Acquisition functions
  - Attribute functions
  - Buffer management functions
  - Interface functions
  - Utility functions

The generic and high-level functions appear within each function class in the logical order you might need to use them. The low-level functions appear within each function class in alphabetical order.

Generic Functions

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

`imgInterfaceOpen`	Opens an interface by name.
`imgSessionOpen`	Opens a session 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 capture all or a portion of a single frame or field to the user buffer.

`imgSnap`	Performs a single frame or 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`	Acquires the most current frame into the specified buffer. Call this function only after calling `imgGrabSetup`.

`imgGrabArea`	Performs a transfer from a continuous acquisition using the given parameters. 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 a buffer list.
`imgSequenceSetup`	Prepares a session for acquiring a sequence into a buffer list.
`imgSessionStartAcquisition`	Starts an acquisition.
`imgSessionStopAcquisition`	Stops an acquisition.

Signal I/O Functions

Signal I/O functions control the trigger lines on IMAQ devices.

`imgSessionTriggerConfigure`	Configures an acquisition to start based on an external trigger.
`imgSessionLineTrigSource`	Configures triggering per line for acquisition from a line scan camera.
`imgSessionTriggerClear`	Disables all triggers on the session.
`imgSessionTriggerDrive`	Configures the specified trigger line to drive a signal out.
`imgSessionTriggerRead`	Reads the current value of the specified trigger line.
`imgSessionWaitSignal`	Waits for a signal to be asserted. This function returns when the specified signal is asserted.

`imgSessionWaitSignalAsync`	Monitors for a signal to be asserted and invokes a user-defined callback when the signal is asserted.
`imgPulseCreate`	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 original level.
`imgPulseDispose`	Disposes of a pulse ID.
`imgPulseRate`	Converts delay and width into delay, width, and timebase values needed by `imgPulseCreate`.
`imgPulseStart`	Starts the generation of a pulse. You must call `imgPulseCreate` first to configure the pulse.
`imgPulseStop`	Stops the generation of a pulse.

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 the acquisition region of interest.
`imgSessionGetROI`	Gets the acquisition region of interest.
`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.

Acquisition Functions

Use acquisition functions to configure, start, and abort an image acquisition, or examine a buffer during an acquisition.

`imgMemLock`	Locks in memory all image buffers associated with the given buffer list in preparation for an acquisition.
`imgMemUnlock`	Unlocks all buffers associated with the given buffer list.
`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.
`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 buffer to a user-specified buffer.
`imgSessionExamineBuffer`	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 performing a ring (continuous) acquisition.
`imgSessionReleaseBuffer`	Releases a buffer that was previously held with `imgSessionExamineBuffer`.

Attribute Functions

Use attribute functions to examine and change NI-IMAQ or camera attributes.

`imgGetAttribute`	Returns an attribute for an interface or session.
`imgGetCameraAttributeNumeric`	Gets the value of numeric camera attributes.
`imgGetCameraAttributeString`	Gets the value of textual camera attributes.
`imgSessionGetLostFramesList`	Gets information about frames that were overwritten during a continuous acquisition.
`imgSessionSetUserLUT8bits`	Downloads a custom 8-bit lookup table to your IMAQ device.
`ImgSessionSetUserLUT16bits`	Downloads a custom 16-bit lookup table to your IMAQ device.
`imgSetAttribute`	Sets an attribute for an interface or session.
`imgSetCameraAttributeNumeric`	Sets the value of numeric camera attributes.
`imgSetCameraAttributeString`	Sets the value of textual camera attributes.

Buffer Management Functions

Use buffer management functions to set up objects such as buffer lists and buffers.

`imgCreateBuffer`	Creates a user frame buffer based on the geometric definitions of the associated session.
`imgCreateBufList`	Creates a buffer list.
`imgDisposeBuffer`	Disposes of a user frame buffer created by `imgCreateBuffer`.

`imgDisposeBufList`	Purges all image buffers associated with this buffer list.
`imgGetBufferElement`	Gets an element of a specific type from a buffer list.
`imgSessionClearBuffer`	Clears a session's image data to the specified pixel value.
`imgSetBufferElement`	Sets a buffer list element of a given type to a specific value.

Interface Functions

Interface functions load and control the selected IMAQ device and cameras. These functions use information stored by Measurement & Automation Explorer.

`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.

Utility Functions

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

`imgPlot`	Plots a buffer to a window.
`imgPlotDC`	Plots a buffer to device context.
`imgSessionSaveBufferEx`	Saves a buffer of a session to disk in bitmap, TIFF, or PNG format.
`imgShowError`	Returns a null terminated string describing the error code.

3 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, sequence, ring, and StillColor acquisitions.

Introduction

The NI-IMAQ application programming interface (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*. 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.
- *Sequence* performs an acquisition that acquires a specified number of buffers, then stops.
- *Ring* performs an acquisition that loops continually on a specified number of buffers.

The high-level function set also allows simple triggered acquisitions and the generation of external signals on the trigger lines.

Low-Level Functions

The low-level function set supports all types of acquisition. You can use low-level functions to do the following:

- 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.

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. All parameters configured in Measurement & Automation Explorer for an IMAQ board are associated with an interface name. You can have one board associated with more than one interface name, which allows you to have several different configurations for one board. You use the interface name to refer to the board in your programming environment.

Interface name information is stored in an interface (`.iid`) file and includes the board serial number, the camera file associated with each channel on the board, and the default channel.

NI-IMAQ specifies all interfaces 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 3-1.

Table 3-1. Interface Naming Convention

Interface Name	Board Installed
`img0`	Board 0
`img1`	Board 1

Table 3-1. Interface Naming Convention (Continued)

Interface Name	Board Installed
...	...
`img`*n*	Board *n*

You can edit existing interfaces or create new interfaces by using Measurement & Automation Explorer. You also can use Measurement & Automation Explorer to configure 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 named `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*.

Session Functions

Use session functions to configure the type of acquisition you want to perform using a particular interface. After you have established a connection to an interface, 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 named `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*.

Managing Buffers

NI-IMAQ can automatically perform your buffer management, or you may perform buffer management manually. If the high-level acquisition routines (`imgSnap`, `imgGrab`, `imgSequenceSetup`, and `imgRingSetup`) are initialized with NULL pointers for buffer addresses, NI-IMAQ automatically allocates a buffer and returns the value of the buffer pointer. After you obtain a buffer pointer, you can use this pointer in successive calls.

For greater control of the acquisition buffers, create buffers with a memory allocation routine (for example, `malloc`) or use 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.

Camera Attributes

The camera attributes allow you to control camera-specific functions, such as integration time and pixel binning, directly from NI-IMAQ. You can also set these attributes in Measurement & Automation Explorer on the **Advanced** tab. Information about specific attributes for your camera is contained in `<my camera>.txt`, which can be found in the `ni-imaq/camera info` directory. For more information about camera attributes and their uses, please consult your camera documentation.

Note Currently, camera attributes are supported only by the IMAQ PCI/PXI-1409, IMAQ PCI/PXI-1422, IMAQ PCI-1424, and IMAQ PCI-1428.

All parameters configured for a camera type are stored in a camera (`.icd`) file. The camera file is then associated with a specific channel on an IMAQ board. The camera file includes information about the video signal timing and the input signal range of the video signal.

The camera attribute file lists all attributes for the camera. Each attribute description contains four fields—**Attribute Name**, **Description**, **Data Type**, and **Possible Values**. The **Attribute Name** field contains the name of the attribute in quotes. The **Data Type** field contains the data type of the attribute—**String**, **Integer**, or **Float**. **String** indicates that there are several valid values for this attribute that are expressed as strings. The list of valid values is indicated in **Possible Values**. **Integer** indicates that the attribute value is a numeric value of type integer. **Float** indicates that the attribute value is a numeric value of type floating point. The valid integer and float values are indicated in **Possible Values**.

Use the `imgSetCameraAttributeString` and `imgGetCameraAttributeString` functions to set and get the value of **String**, **Float,** and **Integer** attributes. Use the `imgSetCameraAttributeNumeric` and `imgGetCameraAttributeNumeric` functions to set and get the value of **Float** and **Integer** attributes.

Note The spelling and syntax of the **Attribute Name** and string values must match the camera attribute file exactly.

NI-IMAQ Status Signals

NI-IMAQ contains several status signals that you can use to trigger the generation of a pulse or invoke a callback function. **Acquisition in Progress** (AQ_IN_PROGRESS) indicates that the board is acquiring image data. This signal goes TRUE when the board initiates the acquisition either through a software or hardware triggered start. When the last piece of image data is transferred to memory, this signal goes FALSE. If the acquisition is a sequence, acquisition in progress stays TRUE throughout the acquisition until the entire sequence is completed. **Acquisition Done** (AQ_DONE) is the inverse of **Acquisition in Progress**. This signal goes TRUE when the last piece of data is transferred to memory, indicating that the acquisition has completed.

Frame Start (FRAME_START**)** and **Frame Done** (FRAME_DONE) indicate the status of an acquisition on a buffer basis. **Frame Start** indicates that a buffer is being acquired. This signal goes TRUE when the board detects the first valid pixel in the current region of interest. The signal goes FALSE when the board detects the last valid pixel of the frame. If the acquisition is a sequence, a ring, or a grab, **Frame Start** and **Frame Done** pulse for every buffer in the acquisition. **Frame Done** is the reverse of **Frame Start** and indicates when the image is transferred from the camera to the IMAQ board.

Buffer Complete (BUF_COMPLETE) indicates when the image data has been transferred to memory and is available for image processing. **Buffer Complete** goes TRUE when the data in an image buffer has been transferred to memory (either onboard or system memory, depending on the acquisition).

Figure 3-1 illustrates the values of the signals during a three-buffer sequence acquisition.

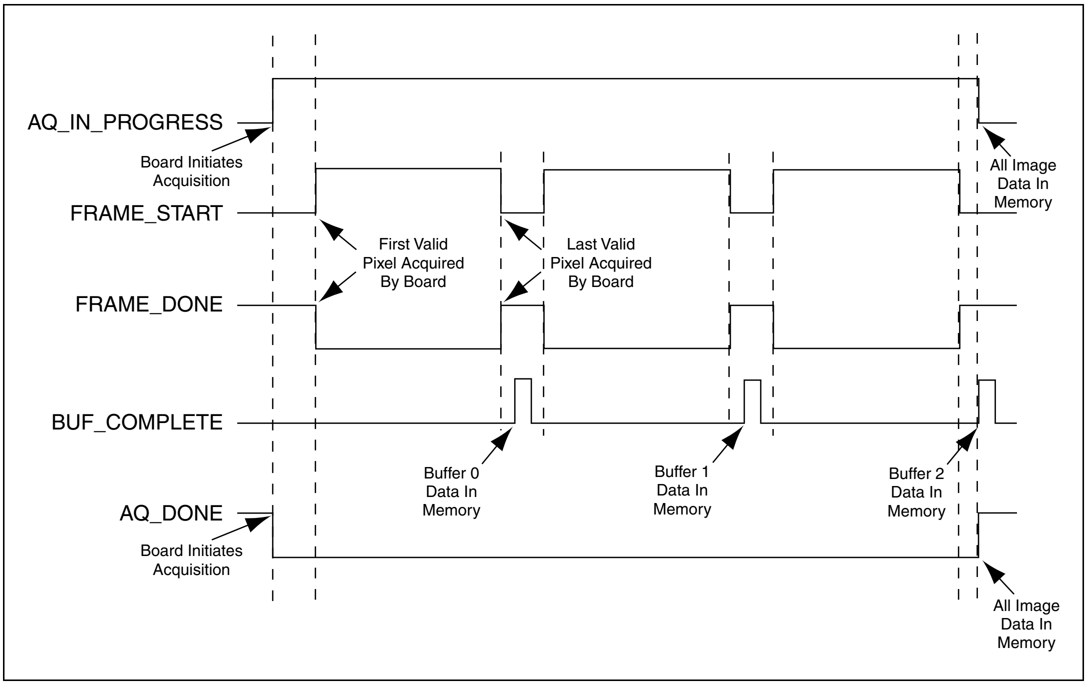

Figure 3-1. NI-IMAQ Status Signals

You can use the NI-IMAQ status signals for many purposes. You can generate pulses based on the assertion of any of these signals. Pulse generation allows you to generate specific timing pulses based on acquisitions to control other aspects of your system, such as a strobe light. Furthermore, you can configure callback functions that are invoked based on any of these signals. For example, you may want to initiate an image processing routine as soon as an image is in memory. You can configure a callback containing image processing code to be invoked when **Buffer Complete** is asserted.

Line Scan Image Acquisition

Unlike area scan cameras, line scan cameras output only one line at a time. However, the programming interface for area scan and line scan cameras is identical. The driver builds up the lines acquired into a 2D image. The height of this image is set in Measurement & Automation Explorer and can be changed with the region of interest height attribute. You can also configure NI-IMAQ to acquire a variable number of lines. See Appendix B, *Variable Height Acquisition*, for more information.

Triggering line scan images is similar to triggering area scan images. Using the `imgSessionTriggerConfigure` function, you can trigger the start of each buffer. The `imgSessionLineTrigSource` function also allows you to trigger each line of the image, not just the start of the image. For example, if you are using a conveyor belt, you can use an encoder to trigger each line of the image and synchronize the movement of the conveyor belt and the image acquisition.

FRAME_START and FRAME_DONE status signals are not valid for a line scan acquisition unless each buffer is triggered. Skip count is not supported for line scan acquisitions. A continuous line scan acquisition into onboard memory is not supported. A continuous line scan acquisition into system memory is supported.

Geometric Definitions

The following defines several terms you should be familiar with when performing image acquisition tasks:

- A *sync window* is the area defined by the horizontal synchronization pulse (HSYNC) and the vertical synchronization pulse (VSYNC).
- An *acquisition window* is the image size specific to a video standard or camera resolution. The default is set by your specific camera file. The window's starting position varies according to camera.
- A *region of interest* (ROI) is a hardware-programmable rectangular portion of the acquisition window. This is a specific area of the image to acquire.

Figure 3-2 illustrates the geometric relationship of these terms.

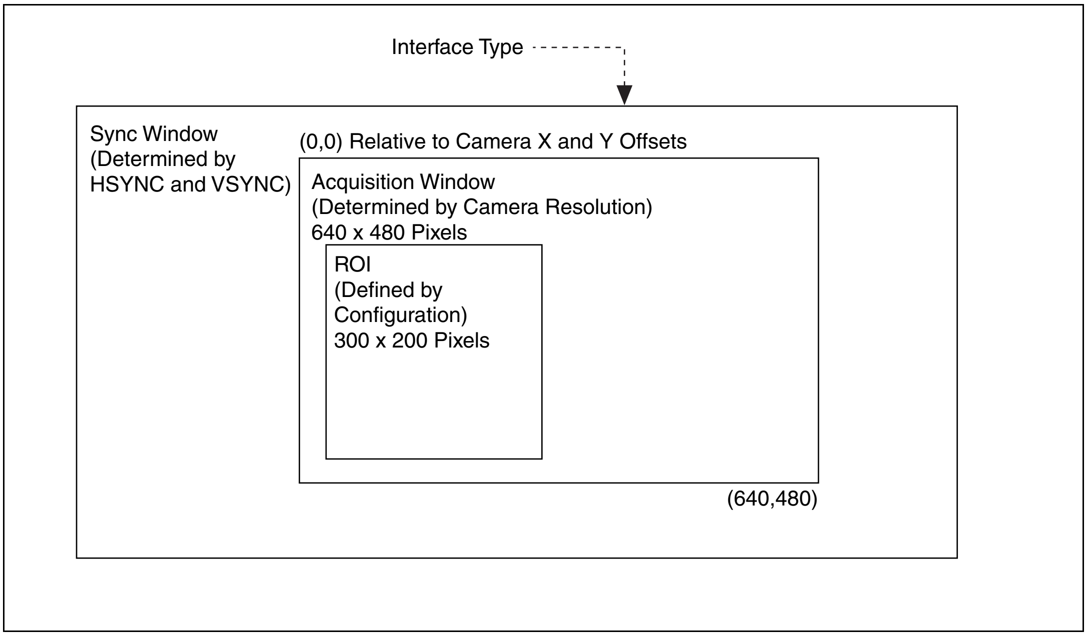

Figure 3-2. Geometric Relationship

Introductory Programming Examples

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

Note You can find the code examples discussed in this section in the `ni-imaq\samples` directory.

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. Use a snap for low-speed or single-capture applications in which ease of programming is essential. Figure 3-3 illustrates a typical snap programming order.

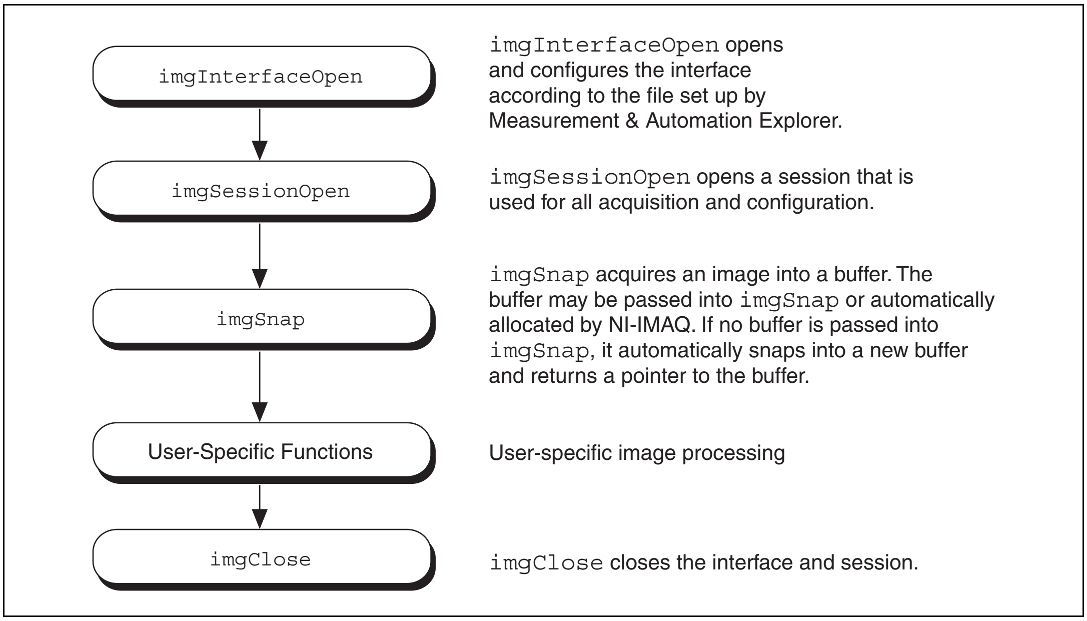
imgInterfaceOpen
imgSessionOpen
imgSnap
User-Specific Functions
imgClose
imgInterfaceOpen opens and configures the interface according to the file set up by Measurement & Automation Explorer.
imgSessionOpen opens a session that is used for all acquisition and configuration.
imgSnap acquires an image into a buffer. The buffer may be passed into imgSnap or automatically allocated by NI-IMAQ. If no buffer is passed into imgSnap, it automatically snaps into a new buffer and returns a pointer to the buffer.
User-specific image processing
imgClose closes the interface and session.

Figure 3-3. Snap Programming Flowchart

The `hlsnap.c` example demonstrates how to perform a single snap using `imgSnap`. The example 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 rowPixel attributes: ROI height multiplied by rowPixel multiplied by the number of bytes per pixel. When you open a session, the ROI values are initialized from the acquisition window (ACQWINDOW) dimensions that are configured in Measurement & Automation Explorer. The ACQWINDOW dimensions 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. Use grab for high-speed applications where you need processing performed on only one image at a time. Figure 3-4 illustrates a typical grab programming order.

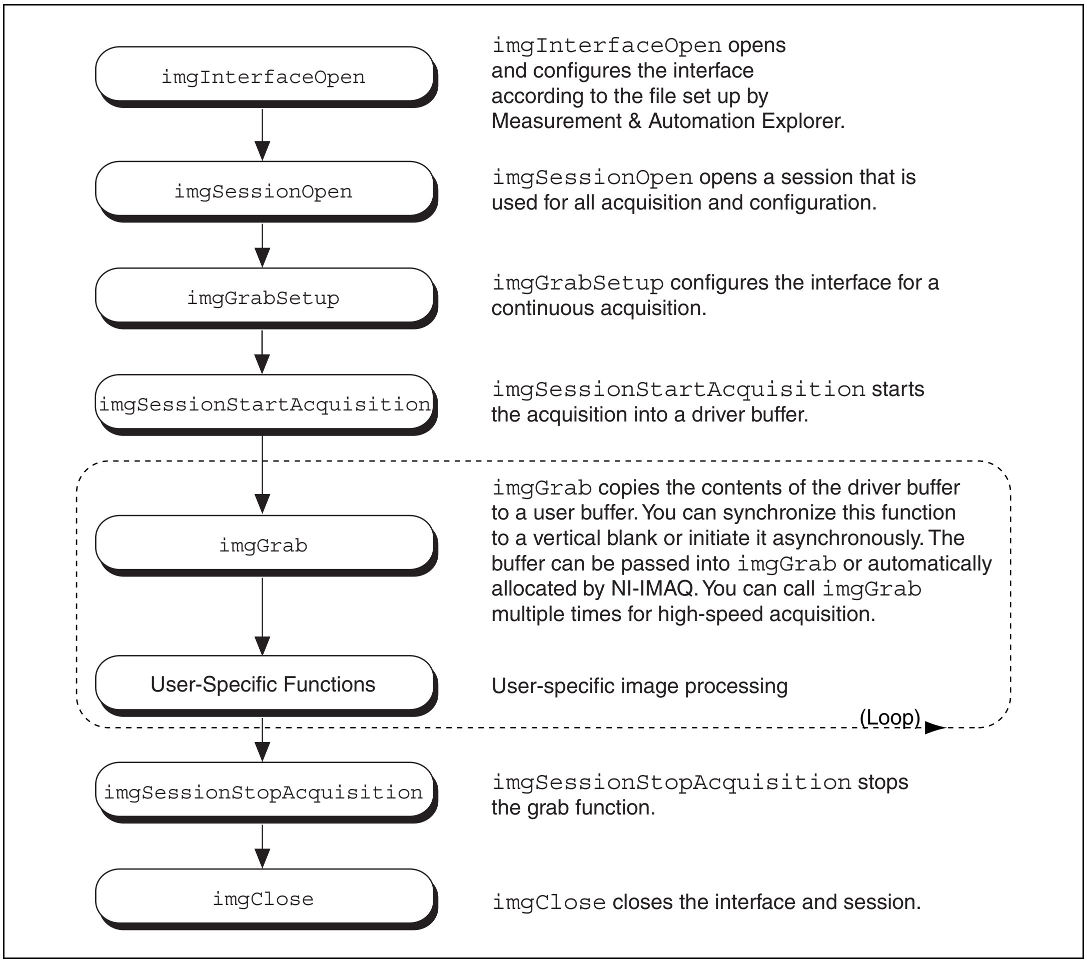
imgInterfaceOpen
imgInterfaceOpen opens and configures the interface according to the file set up by Measurement & Automation Explorer.
imgSessionOpen
imgSessionOpen opens a session that is used for all acquisition and configuration.
imgGrabSetup
imgGrabSetup configures the interface for a continuous acquisition.
imgSessionStartAcquisition
imgSessionStartAcquisition starts the acquisition into a driver buffer.
imgGrab
imgGrab copies the contents of the driver buffer to a user buffer. You can synchronize this function to a vertical blank or initiate it asynchronously. The buffer can be passed into imgGrab or automatically allocated by NI-IMAQ. You can call imgGrab multiple times for high-speed acquisition.
User-Specific Functions
User-specific image processing
(Loop)
imgSessionStopAcquisition
imgSessionStopAcquisition stops the grab function.
imgClose
imgClose closes the interface and session.

Figure 3-4. Grab Programming Flowchart

The `hlgrab.c` example demonstrates how to perform a grab using `imgGrabArea`. The example 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, `rowPixels`, and `bytesPerPixel`, and starts the acquisition by calling `imgSessionStartAcquisition`. In this example, the program allocates a user buffer for grabbing and passes 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 Sequence Functions

Sequence functions include `imgSequenceSetup`, `imgSessionStartAcquisition`, and `imgStopAcquisition`. A *sequence* initiates a variable-length and variable-delay transfer to multiple buffers. You can configure the delay between acquisitions with `imgSequenceSetup` and specify both the buffer list 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.

Use a sequence in 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 3-5 illustrates a typical sequence programming order.

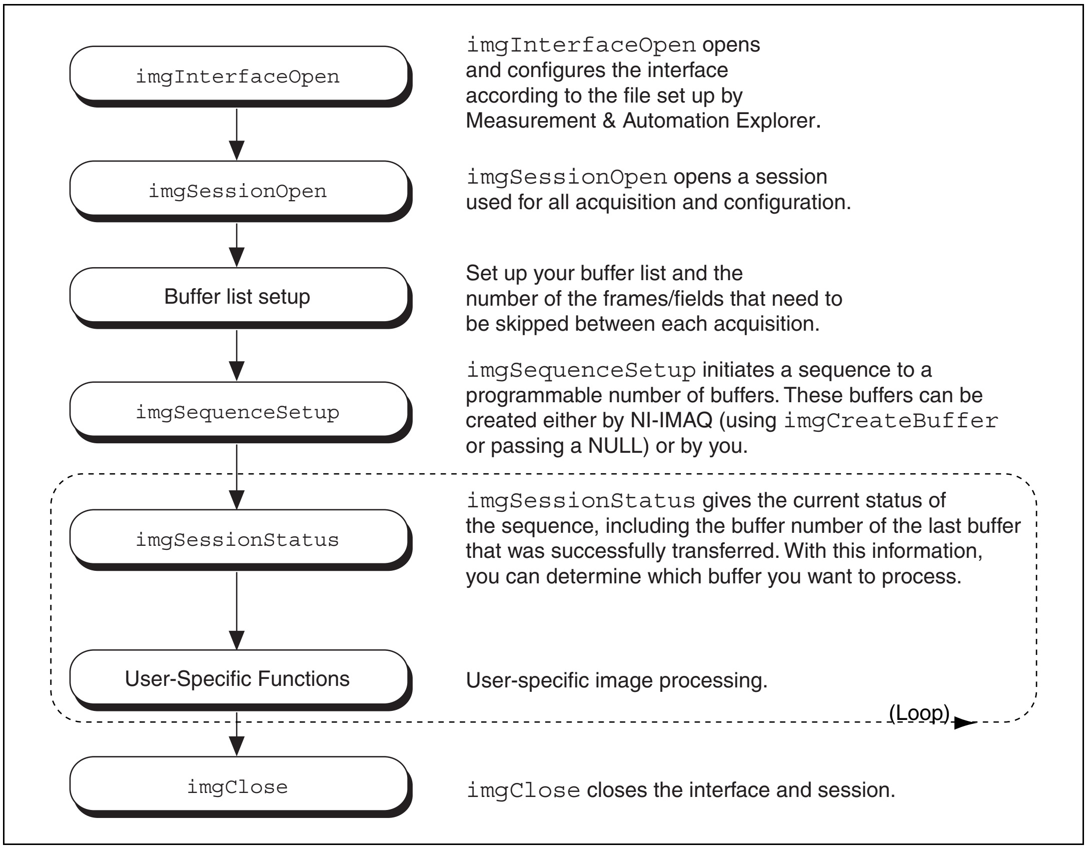
imgInterfaceOpen
imgInterfaceOpen opens and configures the interface according to the file set up by Measurement & Automation Explorer.
imgSessionOpen
imgSessionOpen opens a session used for all acquisition and configuration.
Buffer list setup
Set up your buffer list and the number of the frames/fields that need to be skipped between each acquisition.
imgSequenceSetup
imgSequenceSetup initiates a sequence to a programmable number of buffers. These buffers can be created either by NI-IMAQ (using imgCreateBuffer or passing a NULL) or by you.
imgSessionStatus
imgSessionStatus gives the current status of the sequence, including the buffer number of the last buffer that was successfully transferred. With this information, you can determine which buffer you want to process.
User-Specific Functions
User-specific image processing.
(Loop)
imgClose
imgClose closes the interface and session.

Figure 3-5. Sequence Programming Flowchart

The `HLSeq.c` example demonstrates how to perform a sequence acquisition using `imgSequenceSetup`. The example sets up a sequence that uses 10 user-allocated buffers. 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.

High-Level Ring Functions

Ring and sequence functions include `imgRingSetup`, `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 used for transfers and the number of buffers. After you call `imgRingSetup`, you can monitor the status of the transfer and perform processing on any of the buffers in the ring. Use a ring 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 3-6 illustrates a typical ring programming order.

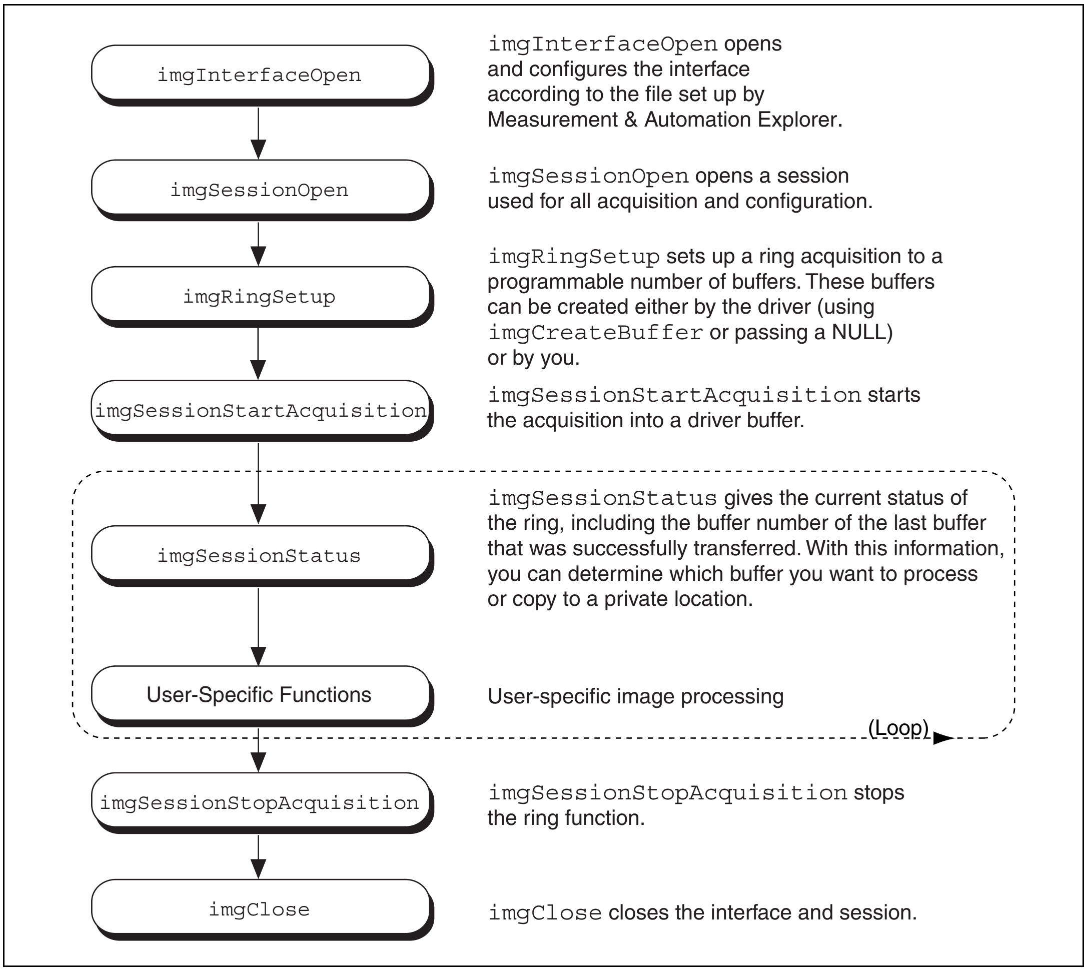
imgInterfaceOpen
imgInterfaceOpen opens and configures the interface according to the file set up by Measurement & Automation Explorer.
imgSessionOpen
imgSessionOpen opens a session used for all acquisition and configuration.
imgRingSetup
imgRingSetup sets up a ring acquisition to a programmable number of buffers. These buffers can be created either by the driver (using imgCreateBuffer or passing a NULL) or by you.
imgSessionStartAcquisition
imgSessionStartAcquisition starts the acquisition into a driver buffer.
imgSessionStatus
imgSessionStatus gives the current status of the ring, including the buffer number of the last buffer that was successfully transferred. With this information, you can determine which buffer you want to process or copy to a private location.
User-Specific Functions
User-specific image processing
(Loop)
imgSessionStopAcquisition
imgSessionStopAcquisition stops the ring function.
imgClose
imgClose closes the interface and session.

Figure 3-6. Ring Programming Flowchart

The `HLRing.c` example demonstrates how to perform a ring acquisition using `imgRingSetup`. The example sets up a ring containing six buffers and sets the skip count to three, which causes the program to acquire on every fourth frame. Unlike the sequence example, the skip count is set to the same value for every buffer in the ring. 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 continues until a designated condition is met and then the acquisition stops.

High-Level Signal I/O Functions

The signal I/O functions fall into two categories—triggering acquisitions and driving the trigger lines. Use triggered acquisitions to acquire images precisely when an external event occurs, such as a sensor activating. Driving trigger lines allows you to control external devices in sync with the image acquisition. For example, you can fire a strobe light when a sequence acquisition begins.

You can initiate any of the four types of acquisitions from an external trigger source by using `imgSessionTriggerConfigure`. For sequence and ring acquisitions, you can choose to trigger only the first buffer in the list, or you can choose to trigger each buffer in the list. After you use this function to set up the trigger, any acquisition performed on the session waits for a trigger. Use `imgSessionTriggerClear` to remove the trigger settings from the session.

Some applications need to send signals out from the IMAQ hardware to an external device. You can drive many types of signals out of the trigger lines by using `imgSessionTriggerDrive`. This function's parameters include a trigger line number, the polarity of the line, and what to drive on the line. You can use a steady state value of high or low or one of the internal state signals of the hardware, such as acquisition in progress. When you need to generate specific pulses, use `imgPulseCreate` and `imgPulseStart`.

Figure 3-7 shows the outline of a program that waits for an external trigger on line 1 before acquiring a single image. It also configures the driver to assert RTSI trigger line 3 when the acquisition is finished. The `trigsnap.c` example contains C code that implements this program.

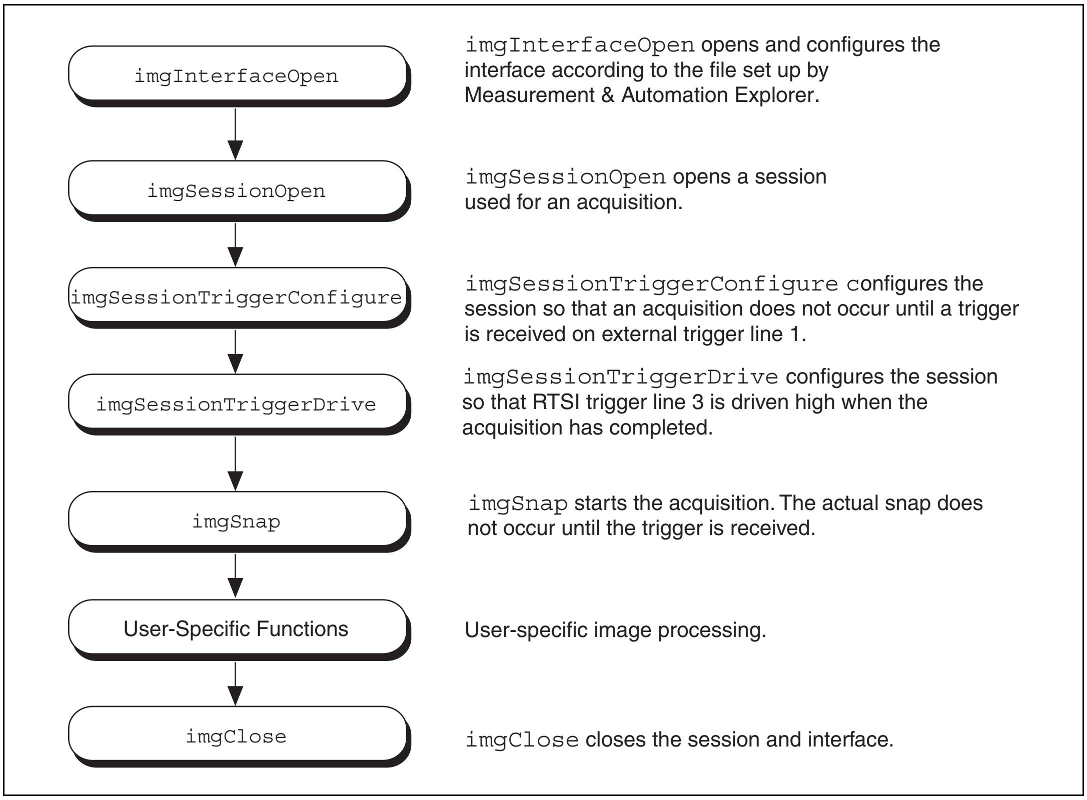
imgInterfaceOpen
imgInterfaceOpen opens and configures the interface according to the file set up by Measurement & Automation Explorer.
imgSessionOpen
imgSessionOpen opens a session used for an acquisition.
imgSessionTriggerConfigure
imgSessionTriggerConfigure configures the session so that an acquisition does not occur until a trigger is received on external trigger line 1.
imgSessionTriggerDrive
imgSessionTriggerDrive configures the session so that RTSI trigger line 3 is driven high when the acquisition has completed.
imgSnap
imgSnap starts the acquisition. The actual snap does not occur until the trigger is received.
User-Specific Functions
User-specific image processing.
imgClose
imgClose closes the session and interface.

Figure 3-7. Signal I/O Function Programming Flowchart

Advanced Programming Examples

Use low-level functions or combine high- and low-level functions for more advanced programming techniques, including snap, grab, sequence, ring, and color image acquisitions.

Performing a Snap Using Low-Level Functions

The `LLSnap.c` example demonstrates how to perform a snap acquisition using low-level calls. The example 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 `rowPixels` 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. 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

The `LLGrab.c` example demonstrates how to perform a grab acquisition using low-level calls. The example sets up a continuous acquisition to a single user-allocated buffer.

As described in the low-level snap example, 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 Sequence Acquisition Using Low-Level Functions

The `LLSeq.c` example demonstrates how to perform a sequence acquisition using low-level calls. The example sets up a sequence acquisition to multiple buffers allocated by NI-IMAQ. As described in the low-level snap example, 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 memory requirement calculation is not necessary if you choose to use the default acquisition window width, rowPixels, and ROI. In this case, NI-IMAQ allocates the correct size buffer if you pass 0 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.

Performing a Ring Acquisition Using Low-Level Functions

The `LLRing.c` example demonstrates how to perform a ring acquisition using low-level calls. The example sets up a continuous acquisition to multiple buffers allocated by NI-IMAQ.

As described in the low-level snap example, 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. The program creates the buffer and configures the buffer list for each buffer element in the ring. The program then locks the memory and starts the image acquisition 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 you use `imgSessionExamineBuffer`, the driver does not allow you to write new data into that buffer during the analysis.
Use `imgSessionReleaseBuffer` to return the buffer to the continuous sequence.

StillColor Snap Programming

You can use the high-level snap function to acquire StillColor images from either composite or RGB video sources. As shown in Figure 3-8, acquiring a StillColor image is identical to acquiring a monochrome image except for two session attribute settings. For more information on StillColor, see Appendix A, *Color Basics and StillColor*.

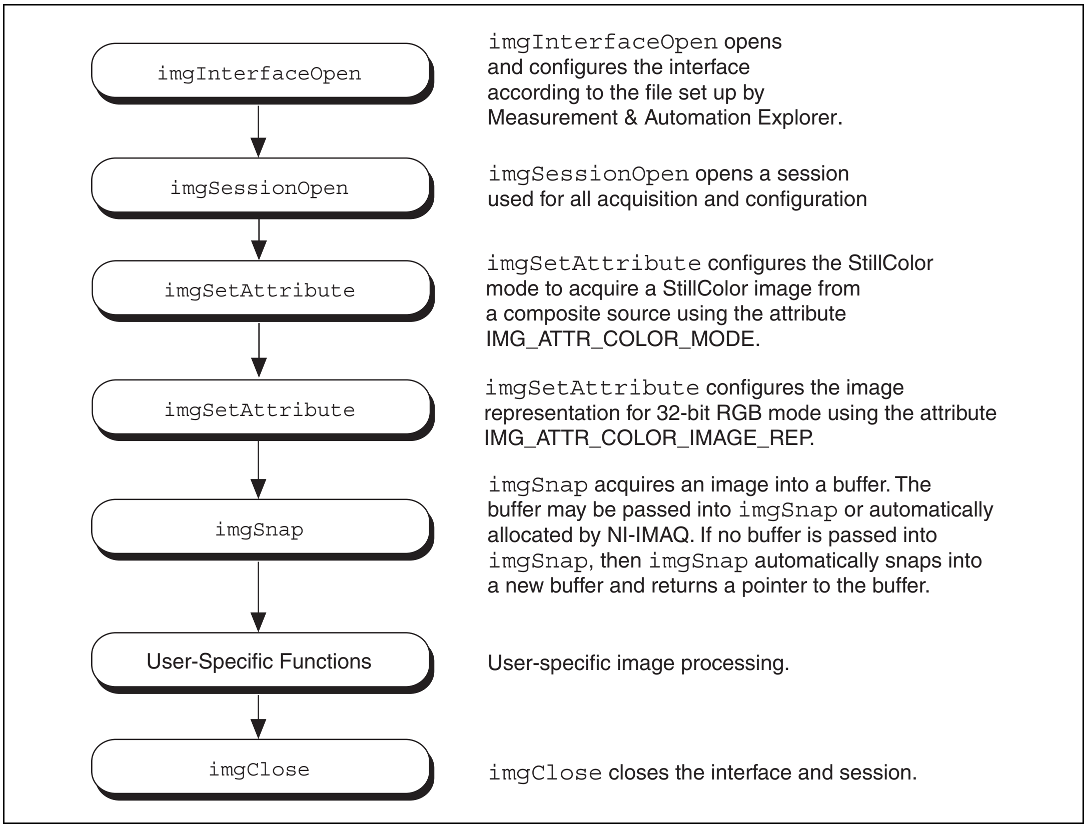
imgInterfaceOpen
imgInterfaceOpen opens and configures the interface according to the file set up by Measurement & Automation Explorer.
imgSessionOpen
imgSessionOpen opens a session used for all acquisition and configuration
imgSetAttribute
imgSetAttribute configures the StillColor mode to acquire a StillColor image from a composite source using the attribute IMG_ATTR_COLOR_MODE.
imgSetAttribute
imgSetAttribute configures the image representation for 32-bit RGB mode using the attribute IMG_ATTR_COLOR_IMAGE_REP.
imgSnap
imgSnap acquires an image into a buffer. The buffer may be passed into imgSnap or automatically allocated by NI-IMAQ. If no buffer is passed into imgSnap, then imgSnap automatically snaps into a new buffer and returns a pointer to the buffer.
User-Specific Functions
User-specific image processing.
imgClose
imgClose closes the interface and session.

Figure 3-8. Composite StillColor Snap Programming Flowchart

The `SCSnap.c` example demonstrates how to perform a single, composite StillColor snap. The example first opens an interface and a session. The example then uses `imgSetAttribute` to enable and configure StillColor mode to acquire a composite image. The example also configures the image data representation to 32-bit RGB mode. `imgSnap` acquires a StillColor image and returns the image data in the buffer. After the example processes the image, it calls `imgClose` to close the handles and free all of the resources associated with the interface.

4 Programming with NI-IMAQ VIs

This chapter describes how to use National Instruments programming and application software, such as LabVIEW and IMAQ Vision, with your IMAQ hardware and NI-IMAQ VIs.

Introduction

LabVIEW features interactive graphics and a state-of-the-art user interface. The NI-IMAQ VI Library, a series of virtual instruments (VIs) for using LabVIEW with your IMAQ device, is included with your NI-IMAQ software.

IMAQ Vision for LabVIEW is an image processing and analysis library that consists of more than 250 VIs. If you have not purchased the IMAQ Vision image processing and analysis libraries, you can use the four IMAQ Vision VIs included with your NI-IMAQ software. If you use these basic functions, you can later upgrade your programs to use IMAQ Vision without any changes to your image acquisition VIs.

Before you start building your IMAQ application, you should know the following basic IMAQ for LabVIEW concepts:

- Location of the NI-IMAQ examples
- Location of the NI-IMAQ VIs in LabVIEW
- Common NI-IMAQ VI parameters
- Buffer management
- NI-IMAQ acquisition types
- Acquisition VIs
- StillColor acquisition
- Triggering
- Image display
- Camera attributes
- Error handling
- Error code format

Location of NI-IMAQ Examples

The NI-IMAQ VI examples illustrate some common applications. You can find these examples in the `labview\examples\imaq` directory for LabVIEW. For a brief description of any example, open the example VI and choose **File»VI Info»Documentation** for a text description of the example.

Location of the NI-IMAQ VIs

You can find the NI-IMAQ VIs in the **Functions** palette from your LabVIEW block diagram. Select the **Motion and Vision** palette and then select the **Image Acquisition** palette, as shown in Figure 4-1.

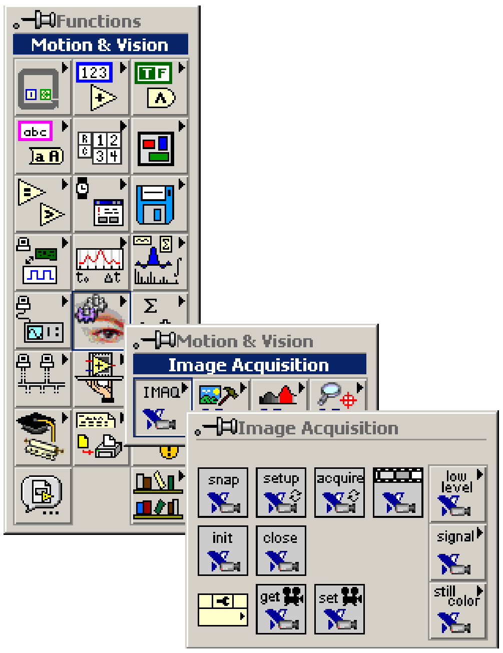

Figure 4-1. LabVIEW Functions Palette with IMAQ Palette

The most commonly used VIs are on the **Image Acquisition** palette. You can find VIs for basic acquisition and changing attributes. The **Motion and Vision»Image Acquisition»IMAQ Low Level** palette contains VIs for more advanced applications.

The **Motion and Vision»Image Acquisition»IMAQ Signal I/O** palette contains VIs for using triggers and pulse generation with IMAQ devices.

See the NI-IMAQ online help for more information on these VIs.

Common NI-IMAQ VI Parameters

IMAQ Session is a unique identifier that specifies the interface file used for the acquisition. It is produced by the IMAQ Init VI and used as an input to all other NI-IMAQ VIs. The NI-IMAQ VIs use **IMAQ Session Out**, which is identical to **IMAQ Session**, to simplify dataflow programming. **IMAQ Session Out** is similar to the duplicate file sessions provided by the file I/O VIs. The high-level acquisition VIs—IMAQ Snap, IMAQ Grab Setup, and IMAQ Sequence—require you to wire **IMAQ Session In** only if you are using an interface other than the default `img0`, if you are using multiple boards, or if you need to set IMAQ properties before the acquisition.

Many acquisition VIs require that you supply an image buffer to receive the captured image. You can create this image buffer with the IMAQ Create VI. Consult the *Buffer Management* section of this chapter for more information. The input that receives the image buffer is **Image in**. The **Image out** output returns the captured image.

The acquisition VIs use the **Region of Interest** input to specify a rectangular portion of an image frame to be captured. You can use **Region of Interest** to reduce the size of the image you want to capture. **Region of Interest** is an array of four elements with the elements defined as Left, Top, Right, Bottom. If **Region of Interest** is not wired, the entire image acquisition window is captured. You configure the default acquisition window using Measurement & Automation Explorer.

The acquisition VIs use the **Step x** and **Step y** inputs to specify a horizontal and vertical sampling step. The sampling step causes a reduction in spatial resolution.

Buffer Management

IMAQ Create and IMAQ Dispose manage image buffers in LabVIEW. IMAQ Create, shown in Figure 4-2, allocates an image buffer. **Image Name** is a label for the buffer created. Each buffer must have a unique name. **Image Type** specifies the type of image being created. Use **8 bits** for

8-bit monochrome images, **16 bits** for 10-, 12-, and 14-bit monochrome images, **RGB** for RGB color images, and **HSL** for HSL color images.

New Image contains information about the buffer, which is initially empty. When you wire **New Image** to the **Image in** input of an image acquisition VI, the image acquisition VI allocates the correct amount of memory for the acquisition. If you are going to process the image, you might need to wire to **Border Size**. **Border Size** is the width in pixels created around an image. Some image processing functions, such as labeling and morphology, require a border.

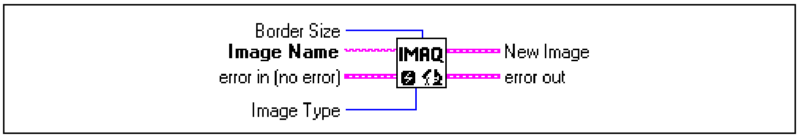

Figure 4-2. IMAQ Create

IMAQ Dispose, shown in Figure 4-3, frees the memory allocated for the image buffer. Call this VI only after the image is no longer required for processing.

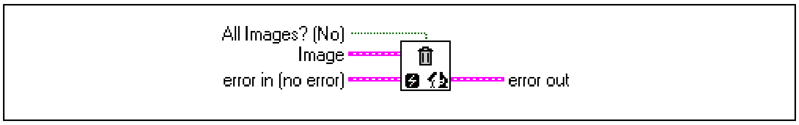

Figure 4-3. IMAQ Dispose

NI-IMAQ Acquisition Types

Four NI-IMAQ image acquisition types are available in LabVIEW—snap, grab, sequence, and ring. The following sections describe each acquisition type and give examples.

Snap

A *snap* acquires a single image into a memory buffer. Use this acquisition mode to acquire a single frame or field to a buffer. When you invoke a snap, it initializes the board and acquires the next incoming video frame (or field) to a buffer. Use a snap for low-speed or single-capture applications.

Use the IMAQ Snap VI for snap applications. Figure 4-4 shows a simplified block diagram for using IMAQ Snap.

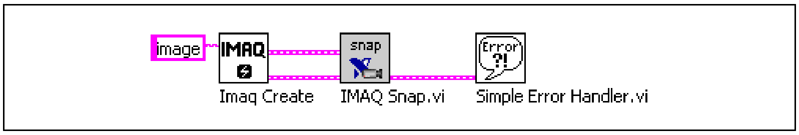

Figure 4-4. Acquiring an Image Using Snap

Grab

A *grab* is a continuous, high-speed acquisition of data to a single buffer in host memory. This function performs an acquisition that loops continually on one buffer. You can get a copy of the acquisition buffer by grabbing a copy to a LabVIEW image buffer.

You must use two VIs—IMAQ Grab Setup and IMAQ Grab Acquire—for a grab acquisition in LabVIEW. IMAQ Grab Setup, which you call only once, initializes the acquisition and starts capturing the image to an internal software buffer. IMAQ Grab Acquire, which you can call multiple times, copies the image currently stored in the internal buffer to a G image buffer. The **Immediate?** input to IMAQ Grab Acquire determines if the copy takes place immediately or if it waits for the next vertical blank. If **Immediate?** is FALSE, IMAQ Grab waits for the next vertical blank signal and then transfers the image from the internal buffer to the G image buffer. If **Immediate?** is TRUE, IMAQ Grab immediately transfers the image from the internal buffer to the G image buffer, which could result in portions of the image transferred being acquired at different times. A typical application for an immediate transfer is the acquisition of images of stationary objects. After the program finishes copying images, call IMAQ Close once to shut down the acquisition.

Figure 4-5 shows a simplified block diagram for using IMAQ Grab Setup and IMAQ Grab Acquire. In this example, you perform an immediate copy by wiring a TRUE to the **Immediate?** input.

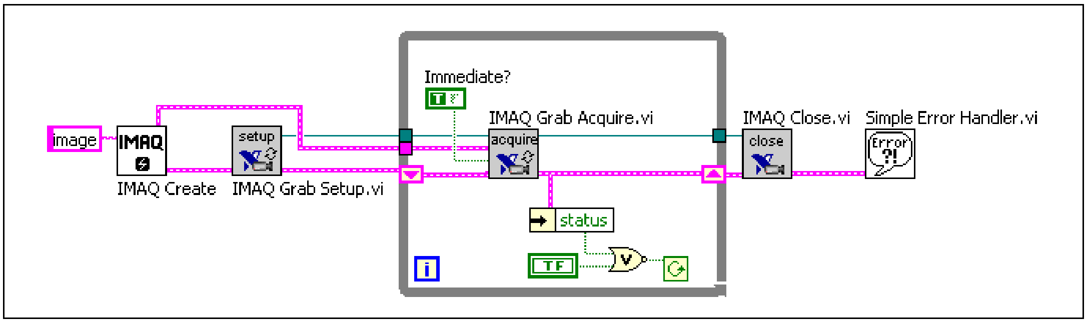

Figure 4-5. Acquiring Images Using Grab

Sequence

A sequence initiates a variable-length and variable-delay transfer to multiple buffers. Use a sequence for applications that process multiple images. You can configure a sequence to acquire every frame or skip a variable number of frames between each image.

Use IMAQ Sequence for sequence applications. IMAQ Sequence starts, acquires, and releases a sequence acquisition. The input **Skip Table** is an array containing the number of frames to skip between images. IMAQ Sequence does not return until the entire sequence is acquired.

Figure 4-6 shows a simplified block diagram for using IMAQ Sequence. Place IMAQ Create inside a For Loop to create an array of images for the **Images in** input to IMAQ Sequence. To Decimal and Concatenate create a unique name for each image in the array.

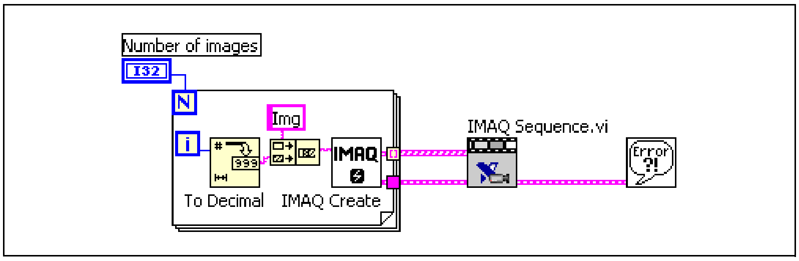

Figure 4-6. Acquiring Images Using Sequence

Ring

A ring initiates a continuous high-speed acquisition to multiple buffers. Use a ring 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 find an example of a ring acquisition in the `examples\imaq\IMAQ Low Level.llb` file.

You can configure a ring to acquire every frame or to skip a fixed number of frames between acquisitions. In LabVIEW, you must use the NI-IMAQ low-level VIs to perform a ring.

Acquisition VIs

Two acquisition VI types are available in LabVIEW—high-level and low-level.

High-Level

You can use the high-level acquisition VIs for basic image acquisition applications. VIs are included for snap, grab, and sequence as described in the *NI-IMAQ Acquisition Types* section of this chapter. You can find examples of using the high-level acquisition VIs in the `examples\imaq\IMAQ High Level.llb` file.

Low-Level

Use the low-level acquisition VIs for more advanced image acquisition applications, including ring acquisitions and acquisitions to onboard memory. The low-level VIs configure an acquisition, start an acquisition, retrieve the acquired images, and stop an acquisition. You can use these VIs in conjunction with the event VIs to construct advanced IMAQ applications.

Follow these general steps to perform a low-level acquisition:

1. Call IMAQ Init to initialize the board and create an **IMAQ Session.**
2. Configure the acquisition with IMAQ Configure List and IMAQ Configure Buffer. IMAQ Configure List configures a buffer list to be used in an acquisition. The buffer list contains a specific number of buffers that will contain the acquired images. The buffers can be stored either in system memory or in onboard memory (for boards with onboard memory such as the IMAQ PCI-1424).

3. Call IMAQ Configure Buffer once for each buffer in the buffer list. The buffer contains the channel from which to acquire and how many frames to skip before acquiring into the buffer.
4. After configuring the buffer list and individual buffers, call IMAQ Start to start the acquisition asynchronously. IMAQ Start returns immediately after the acquisition has started.
5. Access the acquired images using either IMAQ Get Buffer or IMAQ Extract Buffer. IMAQ Get Buffer returns acquired images from the buffer list and is normally used for snap and sequence acquisitions. IMAQ Get Buffer waits until the requested buffer has been acquired to return the image. You can also use this VI to return all images in the buffer list. IMAQ Get Buffer can retrieve images from a continuous acquisition only if the acquisition has been stopped.

   IMAQ Extract Buffer extracts a buffer from a continuous acquisition and allows for the examination of a buffer during acquisition. This VI removes the buffer from the acquisition. NI-IMAQ does not write new data into the buffer until this VI is called again. Use IMAQ Extract Buffer in ring acquisitions when you must process images during the acquisition. IMAQ Copy returns a copy of an acquired image. IMAQ Copy allows you to create a copy of any buffer at any time during the acquisition.
6. After an acquisition, release the resources associated with the acquisition using IMAQ Close. IMAQ Close also stops the acquisition if one is in progress. If you want to stop the acquisition without releasing the resources (such as the image buffers), use IMAQ Stop.

Examples of the low-level acquisition VIs are included in `examples/imaq/IMAQ Low Level.llb`.

StillColor Acquisition

Composite Snap

A StillColor composite snap acquires a single color image into a memory buffer from a color composite camera. Use a StillColor composite snap for high-quality color images of still or very slowly moving objects. For more information on StillColor, refer to Appendix A, *Color Basics and StillColor*.

To perform a StillColor composite snap, use the IMAQ Snap VI for acquisition. Figure 4-7 shows a simplified block diagram for performing a StillColor snap. Perform the following steps to acquire an image using a StillColor composite snap:

1. Use the IMAQ Init VI to generate an **IMAQ Session** for StillColor acquisition.
2. Use the IMAQ Property Node to set up the IMAQ device for StillColor.
3. Set the **StillColor»Color Mode** and **Color»Color Image Representation** properties. The input to **Color Mode** is a ring control.
   a. To create the ring control, pop up on the **Set StillColor Mode** input and choose **Create Constant**.
   b. Click on the created constant with the Operating Tool and select **Composite**. The input to **Color Image Representation** is also a ring control. This input specifies the type of image data to be returned by the IMAQ Snap VI, which is RGB 32-bit in this example.

      The IMAQ Create VI must create an image buffer that corresponds to the image type specified by **Color Image Representation**. Refer to Appendix A, *Color Basics and StillColor*, for information on image representations and corresponding image types.
4. Finally, use IMAQ Snap VI to acquire the image from the IMAQ device.

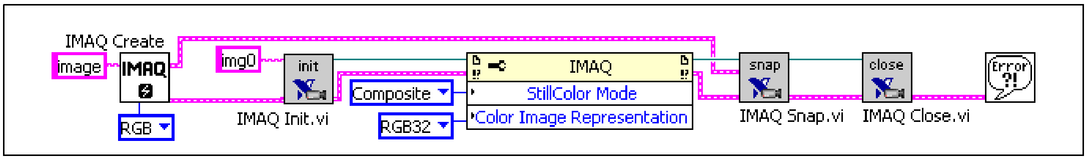

Figure 4-7. Acquiring a StillColor Composite Image

RGB Snap

An RGB snap acquires a single color image into a memory buffer from an RGB camera. Use an RGB snap for acquiring high-quality color images of still or very slowly moving objects. For more information on StillColor RBG refer to Appendix A, *Color Basics and StillColor*.

To perform an RGB snap, use the IMAQ Snap VI for acquisition. Figure 4-8 shows a simplified block diagram for performing an RGB snap. An RGB snap is identical to a composite snap except that the RGB snap's **Color Mode** property should be set to RGB.

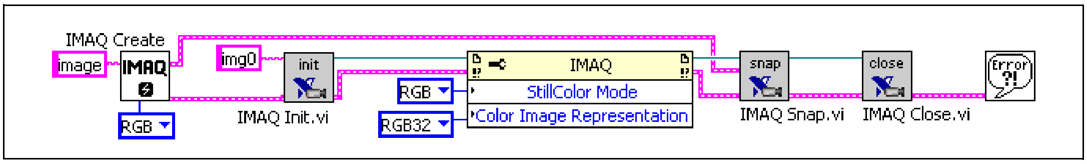

Figure 4-8. Acquiring an RGB Image

Triggering

Often you may need to link or coordinate a vision action or function with events external to the computer, such as receiving a strobe pulse for lighting or a pulse from an infrared detector that indicates the position of an item on an assembly line. A trigger on an IMAQ device can be any TTL-level signal. All of the trigger lines are fully bidirectional so that the device can generate or receive the triggers on any line. The IMAQ PCI/PXI-1407 and PCI/PXI-1411 have one external trigger line. The IMAQ PCI/PXI-1408, PCI/PXI-1409, PCI/PXI-1422, PCI-1424, and PCI-1428 have four external trigger lines and seven Real-Time System Integration (RTSI) bus lines for general purpose use. Use the RTSI triggers to coordinate your IMAQ device with other National Instruments boards, such as data acquisition (DAQ) boards.

Note You can use only four of the seven RTSI triggers at once.

Use IMAQ Configure Trigger to configure the trigger conditions for an acquisition. You must call IMAQ Configure Trigger before the acquisition VI. The **Trigger line** input specifies which external or RTSI trigger receives the incoming trigger signal. Each trigger line has a programmable polarity that is specified with **Trigger polarity**. **Frame timeout** specifies the amount of time to wait for the trigger.

Figure 4-9 shows how to use IMAQ Configure Trigger to perform a snap acquisition based on a trigger.

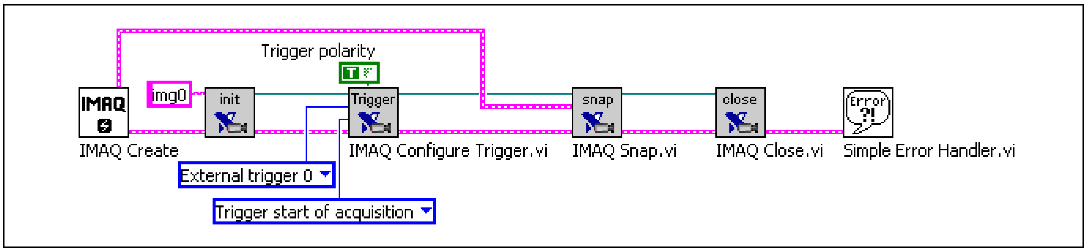

Figure 4-9. IMAQ Triggering

Image Display

Many image acquisition applications require that one or more images be displayed. You have three options for displaying images in LabVIEW.

If you have IMAQ Vision for LabVIEW, the image processing and analysis software for LabVIEW, you can use IMAQ WindDraw. IMAQ WindDraw **Motion and Vision»Vision Utilities»Display** displays an image in an image window. Figure 4-10 illustrates using IMAQ WindDraw to display an image acquired using IMAQ Snap. You can display images in the same way using any acquisition type. For more information on the display capabilities of IMAQ Vision, consult the *IMAQ Vision Concepts Manual*.

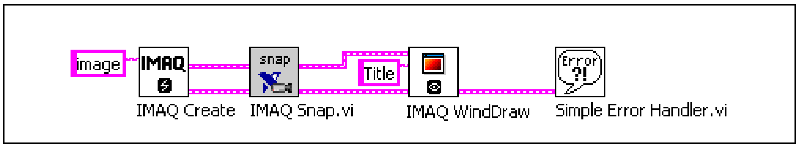

Figure 4-10. Displaying an Image Using IMAQ WindDraw

If you do not have IMAQ Vision, you can display an image on a LabVIEW Intensity Graph for 8-bit and 16-bit monochrome images or on a picture control for RGB images.

Note Using the LabVIEW Intensity Graph to display images requires significantly more processing time than using IMAQ WindDraw.

Before you can properly display an image on an Intensity Graph, you need to make some minor changes to the default properties of the Intensity Graph. Perform the following steps to modify the properties:

1. Place the Intensity Graph on the front panel, pop up on the graph, and choose **Transpose Array**.
2. Create the correct grayscale color palette by popping up on the marker labeled `50` on the color ramp and choosing **Delete Marker**. Also, change the maximum value on the color palette from 100 to the maximum pixel value in your image—255 for 8-bit images, 1,023 for 10-bit images, and 4,095 for 12-bit images.
3. Invert the y-axis. You might also need to change the ranges of the x- and y-axes to match the width and height of the image.

Your intensity graph now should appear similar to the image shown in Figure 4-11. For more information on the Intensity Graph, consult your LabVIEW documentation.

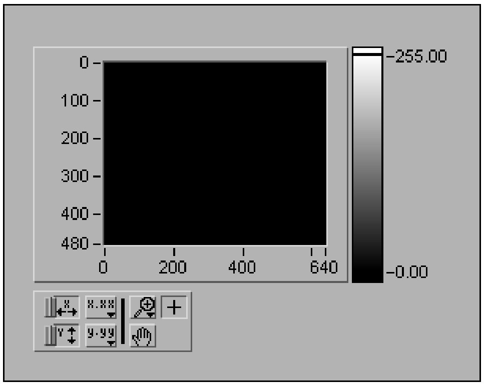

Figure 4-11. Intensity Graph for Image Display

Use the IMAQ ImageToArray VI to copy an image from an image buffer into a LabVIEW array. Then you can wire this array directly to an Intensity Graph for display. Figure 4-12 illustrates using an Intensity Graph to display an image acquired using IMAQ Snap.

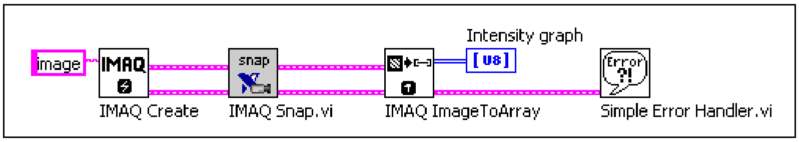

Figure 4-12. Displaying an Image Using an Intensity Graph

To display an RGB image on a picture control, place the picture control on the front panel of your VI. Use the IMAQ ColorImageToArray VI to copy an image from an image buffer into a LabVIEW array. Then you can wire this array to the Draw True-Color Pixmap VI. Wire the new image output from Draw True-Color Pixmap to the picture control indicator. For more information on the picture control, consult the LabVIEW online reference. Figure 4-13 illustrates using a picture control to display an RGB image acquired with IMAQ Snap.

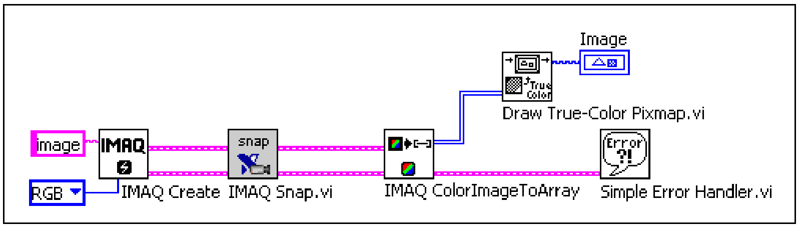

Figure 4-13. Using a Picture Control to Display an RGB Image

Camera Attributes

The camera attribute VIs allow you to control camera functions, such as integration time and pixel binning, directly from LabVIEW. These camera attributes are camera-specific, and you can also set them from Measurement & Automation Explorer on the **Advanced** tab in the Properties dialog box. You can find information about specific attributes for your camera in the `<my camera>.txt` file, which is in the `camera info` directory in your `ni-imaq` directory. For more information about your camera's attributes and their uses, please consult your camera documentation.

Note Currently only the IMAQ PCI/PXI-1409, IMAQ PCI/PXI-1422, IMAQ PCI-1424, and IMAQ PCI-1428 devices support camera attributes.

Use the Set Camera Attribute VI to set the value of a camera attribute. The camera attribute file mentioned above lists all attributes for the camera where each attribute description contains four fields: **Attribute Name**, **Description**, **Data Type**, and **Possible Values**. The **Attribute Name** field contains the name of the attribute in quotes. Wire this field to the **Camera Attribute** input on Set Camera Attribute VI. The **Data Type** field contains the data type of the attribute which can either be **String**, **Integer**, or **Float**. **String** indicates that there is a list of possible values which are listed in

Possible Values in quotes. To set the value of a string attribute, wire the desired string value to **Attribute Value** on Set Camera Attribute.

Note The spelling and syntax of the **Attribute Name** and string values must match the camera attribute file exactly.

A data type of **Integer** indicates that NI-IMAQ converts the string wired to **Attribute Value** to an integer. **Float** indicates that NI-IMAQ converts the string wired to **Attribute Value** to a floating point number. The valid numeric values for integer and float data types are listed in **Possible Values**. Use **Format into String** (**String** subpalette) to convert numerics into strings for use with the IMAQ Set Camera Attribute VI. Figure 4-14 shows how to use IMAQ Set Camera Attribute to set the value of a float camera attribute.

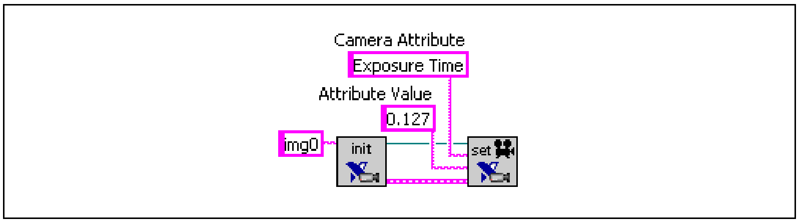

Figure 4-14. IMAQ Set Camera Attribute

Use the IMAQ Get Camera Attribute VI to get the value of a camera attribute. Use the camera attribute file described above to find information about the attributes for your camera. All camera attributes are returned in string format. If the data type of the attribute is integer or float, use the **Scan from String** (**String** subpalette) function to convert the string into a numeric. Figure 4-15 shows how to use IMAQ Get Camera Attribute with **Scan from String** to get the value of a float camera attribute.

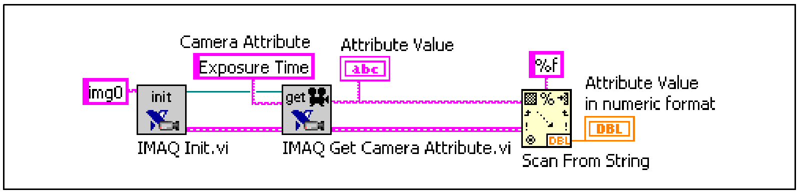

Figure 4-15. Using the IMAQ Get Camera Attribute

Error Handling

Every NI-IMAQ VI contains an **error in** input cluster and an **error out** output cluster, as shown in Figure 4-16. The clusters contain a Boolean value that indicates whether an error occurred, the code for the error, and the source or the name of the VI that returned the error. If **error in** indicates an error, the VI passes the error information to **error out** and does not execute any NI-IMAQ function.

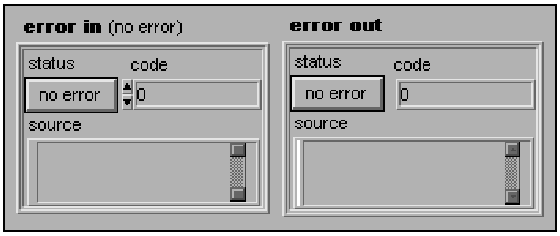

Figure 4-16. Error Clusters

You can use the Simple Error Handler VI (**Functions»Time&Dialog palette**) to check for errors that occur while executing a VI. If you wire an error cluster to the Simple Error Handler VI, the VI deciphers the error information and displays a dialog box that describes the error. If no error occurred, the Simple Error Handler VI does nothing. Figure 4-17 shows how to wire an NI-IMAQ VI to the Simple Error Handler VI.

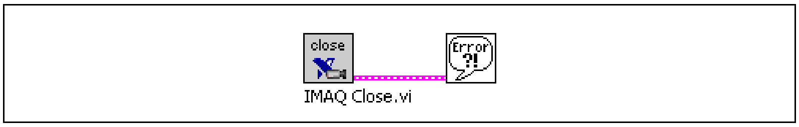

Figure 4-17. Error Checking using the Simple Error Handler VI

Error Code Format

Error format for all NI-IMAQ VIs is the same, as follows:

error in (no error) is a cluster that describes the error status before this VI executes. If **error in** indicates that an error occurred before this VI was called, this VI may choose not to execute its function, but just pass the error through to its error out cluster. If no error has occurred, then this VI executes normally and sets its own error status in error out. Use the error handler VIs to look up the error code and to display the corresponding error message. Using **error in** and **error out** clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

status is TRUE if an error occurred before this VI was called, or FALSE if not. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code.

code is the number identifying an error or warning. If status is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

source is a string that indicates the origin of the error, if any. Usually source is the name of the VI in which the error occurred.

error out is a cluster that describes the error status after this VI executes. If an error occurred before this VI was called, **error out** is the same as **error in**. Otherwise, **error out** shows the error, if any, that occurred in this VI. Use the error handler VIs to look up the error code and to display the corresponding error message. Using **error in** and **error out** clusters is a convenient way to check errors and to specify execution order by wiring the error output from one subVI to the error input of the next.

status is TRUE if an error occurred, or FALSE if not. If **status** is TRUE, code is a non-zero error code. If **status** is FALSE, code can be zero or a warning code.

code is the number identifying an error or warning. If **status** is TRUE, code is a non-zero error code. If status is FALSE, code can be zero or a warning code. Use the error handler VIs to look up the meaning of this code and to display the corresponding error message.

source is a string that indicates the origin of the error, if any. Usually **source** is the name of the VI in which the error occurred.

5 Programming with ActiveX Controls

Introduction

Programming with ActiveX controls allows you to easily configure and perform your image acquisition tasks using IMAQ Vision for Visual Basic and the CWIMAQ control.

This chapter discusses using National Instruments IMAQ Vision for Visual Basic software with your IMAQ image acquisition device, including the organization of IMAQ Vision for Visual Basic and using the CWIMAQ control.

Documentation and Examples

This chapter assumes that you are familiar with Visual Basic and can use ActiveX controls in Visual Basic. The following are good sources of information about Visual Basic and ActiveX controls:

- `msdn.microsoft.com`
- documentation that accompanies Microsoft Visual Studio

In addition to this manual, several documentation resources are available to help you create your vision application:

- *IMAQ Vision Concepts Manual*—If you are new to machine vision and imaging, read this manual to understand the concepts behind IMAQ Vision.
- IMAQ Vision for Visual Basic reference—If you need information about individual methods, properties, or objects, refer to this help file.
- Example programs—If you want examples of how to create specific applications, go to `MeasurementStudio\VB\Samples\MachineVision`.

- CWMachineVision source code—If you want to see the source code for the CWMachineVision control, go to `MeasurementStudio\VB\SourceCode\IMAQ`.
- Application Notes—If you want to know more about advanced IMAQ Vision concepts and applications, refer to the Application Notes located on the National Instruments Web site at `ni.com/appnotes.nsf/`.
- NI Developer Zone (NIDZ)—If you want even more information about developing your vision application, visit the NI Developer Zone at `ni.com/zone`. The NI Developer Zone contains example programs, tutorials, technical presentations, the Instrument Driver Network, a measurement glossary, an online magazine, a product advisor, and a community area where you can share ideas, questions, and source code with vision developers around the world.

IMAQ Vision for Visual Basic Organization

IMAQ Vision for Visual Basic consists of four ActiveX controls contained in two files: `cwimaq.ocx` and `cwmv.ocx`.

cwimaq.ocx

`cwimaq.ocx` contains three ActiveX controls and a collection of ActiveX objects. The ActiveX controls are the CWIMAQ control, CWIMAQVision control, and the CWIMAQViewer control. See the *ActiveX Objects* section for more information about the ActiveX objects.

CWIMAQ Control

Use this control to configure and perform an acquisition from your IMAQ acquisition device. The CWIMAQ control has property pages that allow you to modify various properties to configure your acquisition and gather information about your IMAQ device. Most of the functionality available from the property pages during design time is also available through the properties of the CWIMAQ control during run-time. The control has methods that allow you to perform and control acquisitions, as well.

Note You must have the NI-IMAQ device driver installed on the target system for the CWIMAQ control to function.

CWIMAQVision Control

Use this control to analyze and process images and their related data. The CWIMAQVision control provides methods for reading and writing images to and from files, analyzing images, and performing a variety of image processing algorithms on images.

Note You must purchase IMAQ Vision for Measurement Studio to use this control.

CWIMAQViewer Control

Use this control to display images and provide the interface through which the user will interact with the displayed image. This control includes the ability to zoom and pan images and to draw regions of interest (ROIs) on an image. The CWIMAQViewer control has property pages that allow you to configure the viewer's appearance and behavior during design time as well as properties that you can configure during run-time. The control has methods that allow you to attach images to and detach images from the viewer for display purposes.

Note The CWIMAQViewer control is referred to as a *viewer* in the rest of this document.

cwmv.ocx

`cwmv.ocx` contains one ActiveX control and a collection of ActiveX objects. See the *ActiveX Objects* section for more information about ActiveX objects.

CWMachineVision control

Use this control to perform high-level machine vision tasks, such as measuring distances. This control is written entirely in Visual Basic using the methods on the CWIMAQVision and CWIMAQViewer controls. The source code for the CWMachineVision control is included in the product.

Note You must purchase IMAQ Vision for Measurement Studio to use this control.

ActiveX Objects

ActiveX objects are classified as input and output objects. Use the objects to group related input parameters and output parameters to certain methods, thus reducing the number of parameters that you actually need to pass to those methods.

Note ActiveX objects in `cwimaq.ocx` have a CWIMAQ prefix, and objects in `cwmv.ocx` have a CWMV prefix.

You must create an ActiveX object before you can use it. You can use the `New` keyword in Visual Basic to create these objects. For example, use the following syntax to create and store an image in a variable named `image`:

```
Dim image as New CWIMAQImage
```

Tip If you intend to develop your application in Visual C++, National Instruments recommends that you use IMAQ Vision for LabWindows/CVI. However, if you decide to use IMAQ Vision for Visual Basic to develop applications for Visual C++, you can create objects using the respective Create methods on the CWIMAQVision control or CWMachineVision control. For example, to create a CWIMAQImage object, use the `CWIMAQVision.CreateCWIMAQImage` method.

Buffer Management

The CWIMAQ control uses the CWIMAQImage object in a CWIMAQImages collection as the image buffer when performing an acquisition. You can access this collection using the Images property on the CWIMAQ control. Set the count property of the collection to the number of buffers that your application requires.

You can also set the number of buffers during design time by setting the value of **Image Count** on the **Acquisition** property page for the CWIMAQ control. When you start the acquisition, the CWIMAQ control automatically manages the image type and the size of the images in the collection.

Acquire an Image

Use the CWIMAQ control to acquire images with a National Instruments IMAQ device. You can use IMAQ Vision for Visual Basic to perform one-shot and continuous acquisitions. You can choose the acquisition type during design time by setting the value of the **Acquisition Type** combo box to **One-Shot** or **Continuous**. The **Acquisition Type** combo box is located on the **Acquisition** property page of the CWIMAQ control. You can set the value at run-time by setting the `CWIMAQ.AcquisitionType` property to `cwimaqAcquisitionOneShot` or `cwimaqAcquisitionContinuous`.

One-Shot Acquisition

Use a one-shot acquisition to start an acquisition, perform the acquisition, and stop the acquisition using a single method. The number of frames acquired is equal to the number of images in the images collection. Use the `CWIMAQ.AcquireImage` method to perform this operation synchronously. Use the `CWIMAQ.Start` method to perform this operation asynchronously.

If you want to acquire a single field or frame into a buffer, set the image count to 1. This operation is also referred to as a *snap*. Use a snap for low-speed or single capture applications. The following code illustrates a synchronous snap:

```
Private Sub Start_Click()
    CWIMAQ1.AcquisitionType = cwimaqAcquisitionOneShot
    CWIMAQ1.AcquireImage
End Sub
```

If you want to acquire multiple frames, then set the image count to the number of frames you wish to acquire. This operation is called a *sequence*. Use a sequence for applications that process multiple images. The following code illustrates an asynchronous sequence, when `numberOfImages` is the number of images that you want to process:

```
Private Sub Start_Click()
    CWIMAQ1.AcquisitionType = cwimaqAcquisitionOneShot
    CWIMAQ1.Images.RemoveAll
    CWIMAQ1.Images.Add numberOfImages
    CWIMAQ1.Start
End Sub
```

Continuous Acquisition

Use a continuous acquisition to start an acquisition and continuously acquire frames into the image buffers, and then explicitly stop the acquisition. Use the `CWIMAQ.Start` method to start the acquisition. Use the `CWIMAQ.Stop` method to stop the acquisition. If you use a single buffer for the acquisition, this operation is called a *grab*. The following code illustrates a grab:

```
Private Sub Start_Click()
    CWIMAQ1.AcquisitionType = cwimaqAcquisitionContinuous
    CWIMAQ1.Start
End Sub

Private Sub Stop_Click()
    CWIMAQ1.Stop
End Sub
```

A *ring* operation uses multiple buffers for the acquisition. Use a ring for high-speed applications that require processing on every image. The following code illustrates a ring, where `numberOfImages` is the number of images that you want to process:

```
Private Sub Start_Click()
    CWIMAQ1.AcquisitionType = cwimaqAcquisitionContinuous
    CWIMAQ1.Images.RemoveAll
    CWIMAQ1.Images.Add numberOfImages
    CWIMAQ1.Start
End Sub

Private Sub Stop_Click()
    CWIMAQ1.Stop
End Sub
```

StillColor Acquisition

Use either an IMAQ PCI/PXI-1408 or an IMAQ PCI/PXI-1409 to acquire color images from color composite or RGB cameras. Use a StillColor composite snap to acquire high-quality color images of still or very slowly moving objects. For more information on StillColor, refer to Appendix A, *Color Basics and StillColor*.

Composite Snap

A StillColor composite snap acquires a single color image from a color composite camera and stores it in a memory buffer.

Follow these steps to set up a StillColor composite snap acquisition at design time:

1. Set the value of the **StillColor Mode** radio buttons on the **Attribute** property page for the CWIMAQ control to **Composite**.
2. Select the **Image Rep** value from the combo-box on the **Attribute** property page for the CWIMAQ control.

Follow these steps to set up a StillColor composite snap acquisition at run-time:

1. Set the value of the `CWIMAQ.StillColorMode` property to `cwimaqStillColorModeComposite`.
2. Set the `CWIMAQ.ImageRep` property to the required value.

You can perform a snap operation using the procedures described for one-shot acquisitions. The following code illustrates a StillColor composite synchronous snap.

```
Private Sub Start_Click()
    CWIMAQ1.StillColorMode = cwimaqStillColorModeComposite
    CWIMAQ1.ImageRep = cwimaqRepRGB32
    CWIMAQ1.AcquisitionType = cwimaqAcquisitionOneShot
    CWIMAQ1.Images.RemoveAll
    CWIMAQ1.Images.Add 1
    CWIMAQ1.Start
End Sub
```

RGB Snap

An RGB snap acquires a single color image into a memory buffer from a color RGB camera. A StillColor RGB snap is identical to a composite snap, except that you must set the **StillColor Mode** radio button to **RGB** at design time, or set `CWIMAQ.StillColorMode` to `cwimaqStillColorModeRGB` at run-time. The following code illustrates a StillColor composite synchronous snap:

```
Private Sub Start_Click()

    CWIMAQ1.StillColorMode = cwimaqStillColorModeRGB

    CWIMAQ1.ImageRep = cwimaqRepLum8

    CWIMAQ1.AcquisitionType = cwimaqAcquisitionOneShot

    CWIMAQ1.Images.RemoveAll

    CWIMAQ1.Images.Add 1

    CWIMAQ1.Start

End Sub
```

Note To disable StillColor mode, at design time set the **StillColor Mode** radio buttons to None or at run-time set the `CWIMAQ.StillColorMode` to `cwimaqStillColorModeDisabled`.

Triggering

You may need to coordinate a vision event with events that are occurring outside the computer, such as receiving a strobe pulse for lighting or an infrared detector pulse indicating the position of an item on an assembly line. Any TTL-level signal can serves as a trigger for IMAQ image acquisition devices. All of the lines are fully bidirectional, allowing your IMAQ device to generate or receive the triggers on any line. The IMAQ PCI/PXI-1407 and IMAQ PCI/PXI-1411 feature one external trigger line. The IMAQ PCI/PXI-1408, IMAQ PCI/PXI-1409, IMAQ PCI/PXI-1422, IMAQ PCI-1424, and IMAQ PCI-1428 each feature four external trigger lines and seven Real-Time System Integration (RTSI) bus lines for general purpose use. Use these RTSI triggers to coordinate your IMAQ device with other National Instruments boards, such as data acquisition (DAQ) boards.

Note You can use only four of the seven RTSI triggers at one time.

You can configure triggers during both design time and run-time. At design time, use the **Signal I/O** property page of the CWIMAQ control to configure the triggers. At run-time, use the CWIMAQSignals object, which you can access through `CWIMAQ.Signals` to configure the triggers.

Each CWIMAQSignal object that is added to the CWIMAQSignals collection corresponds to a trigger line that you want to configure. The `CWIMAQSignal.Line` property specifies which external or RTSI trigger receives the incoming trigger signal. Each trigger line has a programmable polarity that is specified with `CWIMAQSignal.Polarity`. Use the `CWIMAQ.FrameTimeout` property to specify the amount of time to wait for a trigger. Once you have configured the triggers, you can perform any acquisition described above. The following code illustrates the use of a trigger to control your acquisition:

```
Private Sub Start_Click()

    'Capture an image on a trigger on RTSI line 0

    CWIMAQ1.Signals.Add.Initialize cwimaqRTSI,_
    cwimaqCaptureStart,

    cwimaqActiveHigh, 0

End Sub
```

Display an Image

Display an image using the CWIMAQViewer control. Use the `CWIMAQViewer.Attach` method to attach the image you want the viewer to display. Once you attach an image to a viewer, the image automatically updates the viewer whenever an operation modifies the contents of the image. You can access the image attached to the viewer using the `CWIMAQViewer.Image` property. Before you attach an image to the viewer, the viewer already has an image attached by default. Therefore, the viewer has an image attached to it at all times. You can use the attached image as either a source image, destination image, or both using the Image property.

```
Private Sub Start_Click()
    CWIMAQ1.AcquisitionType = cwimaqAcquisitionContinuous
    CWIMAQ1.Images.RemoveAll
    CWIMAQ1.Images.Add 1
    CWIMAQViewer1.Attach CWIMAQ1.Images(1)
    CWIMAQ1.Start
End Sub
Private Sub Stop_Click()
    CWIMAQ1.Stop
End Sub
```

You can use the `CWIMAQViewer.Palette` property to access the CWIMAQPalette object associated with the viewer. Use the CWIMAQPalette object to programmatically apply a color palette to the viewer. You can set the `CWIMAQPalette.Type` property to apply predefined color palettes. For example, if you need to display a binary image—an image containing particle regions with pixel values of 1 and a background region with pixel values of 0—set the `Type` property to `cwimaqPaletteBinary`. For more information about color palettes, see Chapter 2, *Display*, of the *IMAQ Vision Concepts Manual*.

You can also set a default palette during design time using the **Menu** property page for the CWIMAQViewer control. You also can change the color palette during run time by using the pop-up menu on the viewer.

Camera Attributes

Camera attributes allow you to control camera functions, such as integration time and pixel binning. These camera attributes are camera-specific. You can access the camera attributes using `CWIMAQ.CameraAttribute`. You can also set them within MAX on the **Advanced** tab in the Properties dialog box. You can find information about specific attributes for your camera in the `<my camera>.txt` file, which is in the camera info directory in your `ni-imaq` directory. For more information about your specific camera attributes and their uses, consult your camera documentation.

Note Currently only the IMAQ PCI/PXI-1409, IMAQ PCI/PXI-1422, IMAQ PCI-1424, and IMAQ PCI-1428 devices support camera attributes.

The camera attribute file lists all attributes for the camera where each attribute description contains four fields: **Attribute Name**, **Description**, **Data Type**, and **Possible Values**. The **Attribute Name** field contains the name of the attribute in quotation marks. Pass the value of this field to the **Attribute** parameter of the **CameraAttribute** property.

The **Data Type** field contains the data type of the attribute, which can either be String, Integer, or Float. String indicates that possible values are listed in quotes in the **Possible Values** section. To set the value of a String attribute, set the desired string value for the CameraAttribute.

Note The spelling and syntax of the **Attribute Name** and string values must match the camera attribute file exactly.

Error Handling

Errors generated by the CWIMAQ control have an Error Context and a Status Code associated with them. The CWIMAQErrorContext constants specify the Error Contexts. The Status Code for an error is a negative value that corresponds to the actual error that occurred.

There are three ways that errors are reported by the CWIMAQ control:

- Return value of the methods
- Exceptions
- IMAQError event

The `CWIMAQ.ExceptionOnError` property specifies whether the control throws an exception when an error occurs. If ExceptionOnError is set to False, no exception is generated. If the error was generated as a result of a method call, then the method returns the status code. If ExceptionOnError is set to True, then the CWIMAQ control throws an exception. If you choose to catch this exception, then you can use the Err object in Visual Basic to get information about the exception that the control generated. The CWIMAQ control generates an `IMAQError` event if there is an error. To select the contexts for which CWIMAQ generates error events, add the desired CWIMAQErrorContexts constants together and assign the sum of the constants to `CWIMAQ.ErrorEventMask`. The IMAQError event provides information about the status code and the context in which the error occurred.

You can handle errors using one of the following methods:

- Set `ExceptionOnError` to `True` and do not provide an exception-handling mechanism. This will direct your application to generate a run-time error and display a run-time error dialog.
- Set `ExceptionOnError` to `True` and provide an exception handling mechanism. You can then direct how your application handles the exception.
- Set `ExceptionOnError` to `False` to check the value of the return code. This method will require you to check the return values every time you make a method call and handle the return values appropriately.
- Use the IMAQError event handler in conjunction with setting the value of `ExceptionOnError` to do what is appropriate for your application. This method allows you to handle errors in certain contexts differently from errors in others. Also, this method allows you to identify errors that might occur during an asynchronous acquisition in order to handle them appropriately.

Warnings

Warnings are different from errors in the following respects:

- They have a positive Status Code.
- They do not generate exceptions, even if ExceptionOnError is True.
- They generate an `IMAQWarning` event instead of an `IMAQError` event.

You can handle warning by checking the return value of the methods or by providing an IMAQWarning event handler.

A

Color Basics and StillColor

This appendix explains basic color theories, describes the different color image output options, and describes the different methods you can use to acquire a color image.

Introduction to Color

Color is the wavelength of the light we receive in our eye when we look at an object. In theory, the color spectrum is infinite. Humans, however, can see only a small portion of this spectrum—the portion that goes from the red edge of infrared light (the longest wavelength) to the blue edge of ultraviolet light (the shortest wavelength). This continuous spectrum is called the visible spectrum.

White light is a combination of all colors at once. The spectrum of white light is continuous and goes from ultraviolet to infrared in a smooth transition. You can represent a good approximation of white light by selecting a few reference colors and weighting them appropriately. The most common way to represent white light is to use three reference components, such as red, green, and blue (R, G, and B primaries). You can simulate most colors of the visible spectrum using these primaries. For example, video projectors use red, green, and blue light generators, and an RGB camera uses red, green, and blue sensors.

The perception of a color depends on many factors, such as:

- *Hue*, which is the perceived dominant color. Hue depends directly on the wavelength of a color.
- *Saturation*, which is dependent on the amount of white light present in a color. Pastels typically have a low saturation while very rich colors have a high saturation. For example, pink typically has a red hue but has a low saturation.
- *Luminance*, which is the brightness information in the video picture. The luminance signal amplitude varies in proportion to the brightness of the video signal and corresponds exactly to the monochrome picture.

- *Intensity*, which is the brightness of a color and which is usually expressed as light or dark. For example, orange and brown may have the same hue and saturation; however, orange has a greater intensity than brown. Intensity is used only in StillColor images. For more information on StillColor, see the *StillColor Basics* section in this appendix.

Image Representations

Color images can be represented in several different formats. These formats can contain all color information from the image or they can consist of just one aspect of the color information, such as hue or luminance. The following image representations can be produced using NI-IMAQ and StillColor or color IMAQ devices.

RGB

The most common image representation is 32-bit RGB format. In this representation, the three 8-bit color planes—red, green and blue—are packed into an array of 32-bit integers. This representation is useful for displaying the image on your monitor. The 32-bit integer organized as:

0	RED	GREEN	BLUE

where the high-order byte is not used and blue is the low-order byte.

StillColor also supports a 24-bit and a 16-bit representation of the RGB image. The 24-bit representation is equivalent to the 32-bit representation; however, there is no unused byte. For the 16-bit representation, the image is packed into an array of 16-bit integers where each 16-bit pixel contains red, green, and blue, encoded with only five bits each. The most significant bit of the integer is always 0.

Color Planes

Each color plane can be returned individually. The red, green, or blue plane is extracted from the RGB image and represented as an array of 8-bit integers.

Hue, Saturation, Luminance, and Intensity Planes

The hue, saturation, luminance, and intensity planes can also be returned individually if you want to analyze the image. You can retrieve the data in 8-bit format to reduce the amount of data to be processed, or, if you are using StillColor, you can retrieve the data in 16-bit format to take advantage of the higher precision available when using averaging.

The 16-bit image representation available in StillColor is scaled so that the pixel values are always positive. The value range is 0 to +32,767, so it is compatible with both 16-bit signed and 16-bit unsigned integers. On average, the 16-bit representation of a plane is equal to 128 times the 8-bit representation of the plane from the same image. The 16-bit representation is generally only used if you are performing averaging on your image. For example, averaging an image 16 times requires four extra bits ($16 = 2^4$) to represent the increased dynamic range. In this case, using the 16-bit representation may increase the dynamic range of your image.

Luminance, Intensity, Hue, and Saturation are defined using the Red, Green, and Blue values in the following formulas:

$$\text{Luminance} = 0.299 \times \text{Red} + 0.587 \times \text{Green} + 0.114 \times \text{Blue}$$

$$\text{Hue} = \text{ATN2}(Y, X)$$

where

$$Y = (\text{Green} - \text{Blue}) / \sqrt{2} \text{ and}$$

$$X = (2 \times \text{Red} - \text{Green} - \text{Blue}) / \sqrt{6}$$

$$\text{Intensity} = (\text{Red} + \text{Green} + \text{Blue}) / 3$$

$$\text{Saturation} = \sqrt{X^2 + Y^2}$$

where

$$Y = (\text{Green} - \text{Blue}) / \sqrt{2} \text{ and}$$

$$X = (2 \times \text{Red} - \text{Green} - \text{Blue}) / \sqrt{6}$$

See Figure A-1 for an illustration of the HSL color space.

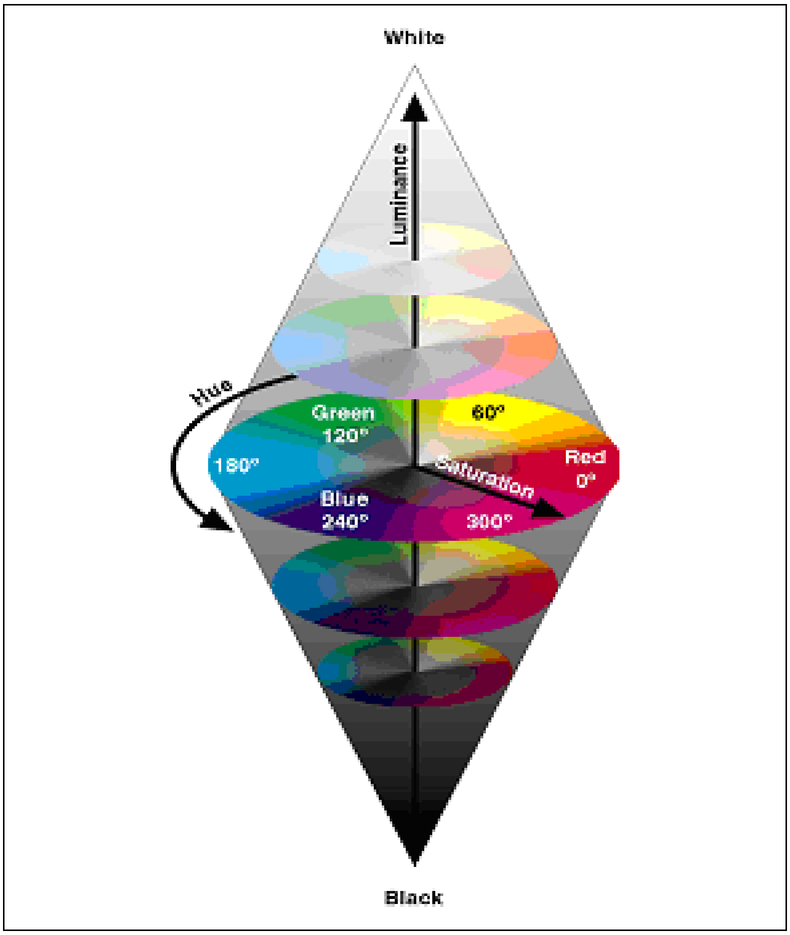

Figure A-1. HSL Color Space

Note You can find color versions of the illustrations in this appendix in the online version of this document included with your NI-IMAQ software kit.

32-Bit HSL and HSI

You can also pack the three 8-bit Hue, Saturation, and Luminance planes (HSL). In StillColor, you can pack the three Hue, Saturation, and Intensity planes (HSI) in one array of 32-bit integers, which is equivalent to the 32-bit RGB representation.

Camera Types

You can use two basic video camera types for color acquisition—RGB cameras and composite color video cameras.

An RGB camera delivers the three basic color components—red, green and blue—on three different wires. This type of camera often uses three independent CCD sensors to acquire the three color signals. RGB cameras are used for very accurate color acquisition.

A composite color camera transmits the video signal on a single wire. The signal is composed of two components that are added together. These components are:

- A monochrome video signal that contains the gray level information from the image and the composite synchronization signals. This signal is the same as a standard monochrome video signal, such as RS-170 or CCIR-601.
- A modulated signal that contains the color information from the image. The format of this signal depends on your camera. The three main color standards are as follows:
  - M-NTSC (also called NTSC), which is used mainly in the U.S. and Japan
  - B/G-PAL (also called PAL), which is used mainly in Europe, India, and Australia
  - SECAM, which is used mainly in France and the former Soviet Republics. SECAM is only used for broadcasting, so SECAM countries often use PAL as the local color image format.

StillColor Basics

StillColor is a technique you can use to acquire color images from composite color video or RGB cameras using the PCI/PXI-1408 or the PCI/PXI-1409 monochrome devices. Use StillColor Composite mode to acquire color images from a composite color video camera. Use StillColor RGB mode to acquire color images from an RGB camera. StillColor composite acquisition results in an image of much higher quality than the traditional color decoding that can be obtained with a color image acquisition board.

To acquire a color image, the PCI/PXI-1408 and PCI/PXI-1409 acquire multiple frames from the camera. Your computer then processes the frames using the StillColor algorithm and creates a single color image. Because StillColor uses your computer to process the image, the acquisition time for a single image depends on your system performance. You can acquire StillColor composite images at rates of up to 2 frames/s and StillColor RGB images at rates of up to 10 frames/s.

You can use StillColor in applications that require high-quality images of still or very slowly moving objects. StillColor supports many different image representations used in scientific or industrial applications, such as RGB bitmap and single plane hue, saturation, luminance, and intensity. StillColor also supports image averaging of up to 128 frames to increase the dynamic range of the StillColor image. See the *Introduction to Color* section in this appendix for more information on image representations.

StillColor Composite

In a composite color video signal, the color information (chroma) is modulated in phase and amplitude around a sub-carrier frequency of 3.58 MHz (NTSC) or 4.43 MHz (PAL). The modulated signal is then added to the luminance information and the entire signal, including synchronization pulses, is transmitted on a single line.

Traditional Color Decoding

On the receiver side or in your IMAQ board, the luminance and the chroma signals must be separated before the color image can be decoded and rebuilt. However, the modulated color information and some of the high-frequency luminance information share the same frequency range around the sub-carrier frequency. This sharing makes it impossible to separate the two signals perfectly and, therefore, perfect reconstruction of the original color image is not possible.

All of the traditional ways to separate the two signals result in visual artifacts on the final picture. Techniques such as frequency-band filtering or comb filtering can minimize some of these artifacts, but most techniques are optimized to obtain the best picture for visualization of a continuous acquisition. The composite color formats are designed so that artifacts resulting from one frame are almost cancelled by artifacts in following frames. This system takes advantage of the slow response time of the human eye to obscure most of these problems.

The situation is different in a single frame acquisition where a single image is needed. A single image usually clearly shows the result of a bad color/luminance separation. Typical weaknesses of traditional separation techniques are:

- Reduced luminance bandwidth, resulting in a blurry image
- Cross-color modulation where rapidly changing colors affect the luminance of the image, as shown on the edges of the parrot's head in Figure A-2
- Cross-luminance modulation where rapidly changing luminance (stripes) results in irritating random color patterns, as shown on the black and white stripes around the parrot's eye in Figure A-2

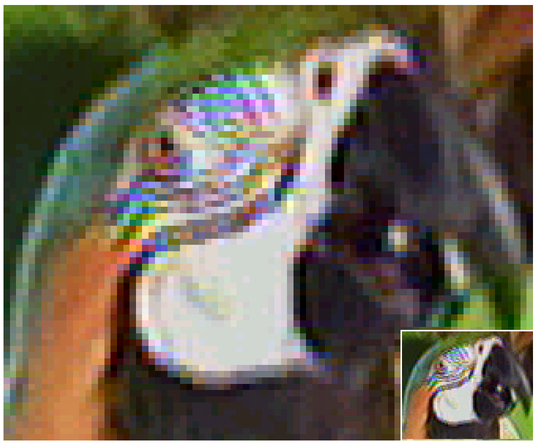

Figure A-2. Traditional Decoding

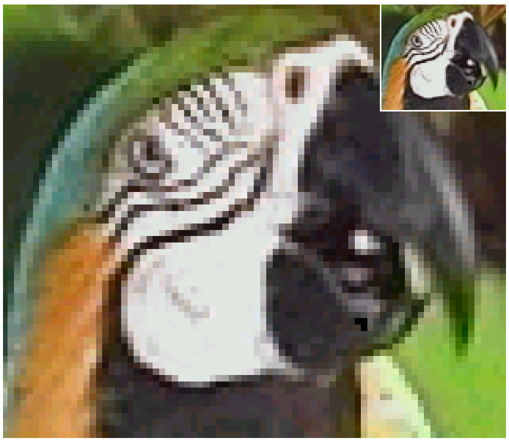

Figure A-3. StillColor Decoding

Note You can find color versions of the illustrations in this appendix in the online version of this document included with your NI-IMAQ software kit.

Both pictures are approximately 80 × 80 pixels in size and are acquired using an NTSC composite video signal. Figure A-1 uses classic decoding algorithms including bandpass/bandstop and comb filtering. Figure A-2 was acquired using the PCI/PXI-1408 and the StillColor algorithm.

Why StillColor?

StillColor is optimized for single-frame acquisition. A StillColor Composite acquisition acquires multiple consecutive frames. Assuming that all frames represent the same scene of still objects, the algorithm then uses knowledge about the composite color format to perfectly separate the color and the luminance information.

In an NTSC video signal, two consecutive frames representing the same object will contain the same luminance information but will have chroma signals that are opposite in phase. By adding the two frames together, the chroma information is cancelled, and by subtracting the two frames from each other, the luminance signal is cancelled. The resulting separation is now perfect, as shown in Figure A-2.

Color and luminance separation is more complex in a PAL video signal. The IMAQ device must acquire three consecutive frames, but the same perfect separation of the color and luminance information can be achieved after manipulation of these images.

After separating the color and luminance signals, the StillColor algorithm then decodes and rebuilds the color image. As shown in Figure A-2, the result does not show any of the artifacts encountered in traditional color decoding methods.

Composite Color Acquisition

The PCI/PXI-1408 and the PCI/PXI-1409, in conjunction with NI-IMAQ, supports acquisition of color images from an NTSC or PAL composite color video camera using the StillColor algorithm.

NI-IMAQ can acquire the multiple frames, decode the color information, and rebuild the image automatically. The output image can be a simple RGB color image or one of many image representations supported by NI-IMAQ. See the *Introduction to Color* section in this appendix for more information on image representations.

You can connect the composite video signal to any of the four input channels on the PCI/PXI-1408 or PCI/PXI-1409. Since StillColor is used for still scenes, you can perform only a snap (a single-image acquisition).

Note The IMAQ PCI/PXI-1411 can acquire continuous color images from NTSC or PAL at 30 frames per second.

StillColor RGB

RGB cameras output a color image using three lines. StillColor RGB will acquire the three signals and construct a color image. The three lines are connected to three channels on your IMAQ board. One frame is acquired from each of the three channels, which represent the red, green, and blue planes of the image. StillColor combines these frames to construct the color image.

RGB Color Acquisition

The PCI/PXI-1408 and PCI/PXI-1409, in conjunction with NI-IMAQ, support acquisition of color images from an RGB camera.

The NI-IMAQ driver can acquire the three frames and rebuild the image automatically. The output image can be a simple RGB color image or one of many image representations supported by the driver. See the *Introduction to Color* section in this appendix for more information on image representations.

For a StillColor RGB snap, connect the three camera channels—red, green, and blue—to Video 1, Video 2, and Video 3, respectively, on your IMAQ board. Specify a channel for the video synchronization signal by selecting that channel as the sync source using the **Operating Mode** tab in Measurement & Automation Explorer for IMAQ. A typical RGB camera includes the composite video synchronization signal in the green signal. You can also use other synchronization sources, such as an external composite video signal that can be connected to Video 0 or an external TTL composite synchronization signal that can be connected to the CSYNCIN pin of your IMAQ device. (See Chapter 4, *Programming with NI-IMAQ VIs*, of your hardware user manual for signal connection information.)

B
Variable Height Acquisition

Introduction

When you perform a line scan application, you may not know the exact size of the object you are imaging. A good example is a conveyer belt application in which you need to perform a continuous acquisition of objects of multiple sizes. For such applications, National Instruments has introduced a mode of triggered acquisition on its PCI/PXI-1409, PCI/PXI-1422, PCI/PXI-1424, and PCI-1428 IMAQ devices that allows you to acquire a variable number of lines in a multiple buffered application.

In this mode, you supply the IMAQ board a trigger which is asserted when you want to begin capture. The IMAQ board continues to acquire lines until the trigger is unasserted. Once the acquisition is complete, the driver returns the number of lines acquired. By using the variable line mode, you acquire only the amount of data that you need. This technology greatly enhances performance by minimizing the total amount of data to process.

Setup for Acquiring a Variable Number of Lines

To set up your system for acquiring a variable number of lines, first determine the size of the largest possible object to image. The NI-IMAQ driver software defines the dimensions of an image as width and height. Width is the number of pixels per line and height is the number of lines in the image. Once you determine the maximum possible object size, in lines, for your application, enter this number for the height parameter in Measurement & Automation Explorer, as follows:

1. Launch Measurement & Automation Explorer
2. Select your camera (**Devices and Interfaces»PCI-142x»your camera**) and click the **Properties** button on the **Basics** tab
3. From the **Properties** page, enter the maximum possible number of lines into the **Height** control

NI-IMAQ uses this line number information to allocate each image into a buffer. This allocation is done prior to the acquisition to ensure that memory is available on the system before the acquisition begins.

Setting Up the Trigger

The VHA trigger can come in on any unused external trigger line or RTSI line. The trigger can be either `High True` or `Low True`. Acquisition of a buffer begins on the assertion of this trigger line and terminates when the line is deasserted or the number of lines equals the **Height** parameter set in Measurement & Automation Explorer.

Enabling VHA in Your Code

The actual implementation varies depending on whether you use LabVIEW or C, but the concept is the same in either case. Perform the following general steps:

1. Enable the Variable Height Acquisition attribute or property.
2. Configure your buffers.
3. Set up your trigger to “Trigger each buffer.”
4. Begin acquisition.
5. Check actual height for each buffer during the acquisition.

Examples

If you install the IMAQ examples for your compiler, you can find a ready-to-run example showing how to acquire a variable number of lines with a line scan camera.

- In LabVIEW, see the `LL VHA Ring.vi` in the `IMAQ Signal IO.llb`
- In LabWindows/CVI, see the `VHA Ring.prj` in the `CVI Samples` folder
- In Microsoft Visual C version 6, see the `VHA Ring.mdp` in the `NI-IMAQ Samples` folder

C

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

A

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.
address	Value that identifies a specific location (or series of locations) in memory.
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.
aspect ratio	The ratio of a picture or image's width to its height.
asynchronous	Property of a function or operation that begins an operation and returns control to the program before the completion or termination of the operation.

B

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.

C

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.
chroma	The color information in a video signal.
chrominance	*See* chroma.
CSYNC	Composite synchronization signal. Signals in a color video system that multiplex all picture information into a single signal, such as NTSC, PAL, or SECAM.

D

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.
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

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.
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).
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.

H

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.

I

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.
interrupt	A computer signal indicating that the CPU should suspend its current task to service a designated activity.
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.

L

line count	The total number of horizontal lines in the picture.
LSB	Least significant bit.
luma	The brightness information in the video picture. The luma signal amplitude varies in proportion to the brightness of the video signal and corresponds exactly to the monochrome picture.
luminance	*See* luma.
LUT	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

memory buffer	*See* buffer.
memory window	Continuous blocks of memory that can be accessed quickly by changing addresses on the local processor.
method	In Visual Basic, 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.
MSB	Most significant bit.
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.

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.
PCLK	Pixel clock signal. Times the sampling of pixels on a video line.

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.

R

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).
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.
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

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.
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.
synchronous	Property or operation that begins an operation and returns control to the program only when the operation is complete.

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.
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