Image Acquisition Toolbox

For Use with MATLAB®

◼ Modeling

◼ Simulation

◼ Implementation

The MathWorks

User's Guide

Version 1

How to Contact The MathWorks:

	www.mathworks.com	Web
	comp.soft-sys.matlab	Newsgroup
@	support@mathworks.com	Technical support
	suggest@mathworks.com	Product enhancement suggestions
	bugs@mathworks.com	Bug reports
	doc@mathworks.com	Documentation error reports
	service@mathworks.com	Order status, license renewals, passcodes
	info@mathworks.com	Sales, pricing, and general information
☎	508-647-7000	Phone
	508-647-7001	Fax
✉	The MathWorks, Inc.	Mail
	3 Apple Hill Drive	
	Natick, MA 01760-2098	

For contact information about worldwide offices, see the MathWorks Web site.

Contents

Introduction

2

Connecting to Hardware

3

Acquiring Image Data

4

5

Working with Acquired Image Data

6

Using Events and Callbacks

Function Reference

7

<div align="right">

Property Reference

</div>

8

Index

Preface

This chapter introduces you to the Image Acquisition Toolbox and describes conventions used by the documentation.

What Is the Image Acquisition Toolbox?

The Image Acquisition Toolbox is a collection of functions that extend the capability of the MATLAB® numeric computing environment. The toolbox supports a wide range of image acquisition operations, including

- Acquiring images through many types of image acquisition devices, from professional grade frame grabbers to USB-based Webcams.
- Viewing a preview of the live video stream
- Triggering acquisitions (includes external hardware triggers)
- Configuring callback functions that execute when certain events occur
- Bringing the image data into the MATLAB workspace

Many of the toolbox functions are MATLAB M-files. You can view the MATLAB code for these functions using the statement

```
type function_name
```

You can extend the capabilities of the Image Acquisition Toolbox by writing your own M-files, or by using the toolbox in combination with other toolboxes, such as the Image Processing Toolbox and the Data Acquisition Toolbox.

Related Products

The MathWorks provides several products that are especially relevant to the kinds of tasks you can perform with the Image Acquisition Toolbox.

For more information about any of these products, see either

- The online documentation for that product if it is installed or if you are reading the documentation from the CD
- The MathWorks Web site, at `http://www.mathworks.com`; see the "products" section

The toolboxes listed below all include functions that extend the capabilities of MATLAB and are complementary to the Image Acquisition Toolbox.

Product	Description
Data Acquisition Toolbox	Acquire and send out data from plug-in data acquisition boards
Image Processing Toolbox	Perform image processing, analysis, and algorithm development
Instrument Control Toolbox	Control and communicate with test and measurement instruments

Configuration Notes

To determine if the Image Acquisition Toolbox is installed on your system, type this command at the MATLAB prompt.

```
ver
```

When you enter this command, MATLAB displays information about the version of MATLAB you are running, including a list of all toolboxes installed on your system and their version numbers.

For information about installing the toolbox, see the MATLAB Installation Guide for your platform.

For the most up-to-date information about system requirements, see the system requirements page, available in the products area at The MathWorks Web site (`www.mathworks.com`).

About the Documentation

This section describes the structure of the Image Acquisition Toolbox documentation.

Structure of the Documentation

The documentation is organized as follows:

- Chapter 1, "Getting Started," introduces the toolbox by stepping through a simple example.
- Chapter 2, "Introduction," describes the components of the Image Acquisition Toolbox and setting up image acquisition hardware.
- Chapter 3, "Connecting to Hardware," describes how you connect MATLAB to an image acquisition device by creating a video input object.
- Chapter 4, "Acquiring Image Data," describes how you trigger the acquisition of image frames. The toolbox supports several types of triggers.
- Chapter 5, "Working with Acquired Image Data," describes how you access logged image data and bring it into the MATLAB workspace where you can work with it.
- Chapter 6, "Using Events and Callbacks," describes how to use event callbacks to enhance the functioning of your application.
- Chapter 7, "Function Reference," provides reference information about the functions provided by the Image Acquisition Toolbox.
- Chapter 8, "Property Reference," provides reference information about the properties of the video input and video source objects.

Typographical Conventions

Item	Convention	Example
Example code	Monospace font	To assign the value 5 to A, enter `A = 5`
Function names, syntax, filenames, directory/folder names, user input, items in drop-down lists	Monospace font	The `cos` function finds the cosine of each array element. Syntax line example is `MLGetVar ML_var_name`
Buttons and keys	**Boldface** with book title caps	Press the **Enter** key.
Literal strings (in syntax descriptions in reference chapters)	**Monospace bold** for literals	`f = freqspace(n,'`**`whole`**`')`
Mathematical expressions	*Italics* for variables Standard text font for functions, operators, and constants	This vector represents the polynomial $p = x^2 + 2x + 3$.

MATLAB output	Monospace font	MATLAB responds with
		
		`A =`
		`    5`

Menu and dialog box titles	**Boldface** with book title caps	Choose the **File Options** menu.
New terms and for emphasis	*Italics*	An *array* is an ordered collection of information.
Omitted input arguments	(...) ellipsis denotes all of the input/output arguments from preceding syntaxes.	`[c,ia,ib] = union(...)`
String variables (from a finite list)	*Monospace italics*	`sysc = d2c(sysd,'`*`method`*`')`

Getting Started

The best way to learn about the capabilities of the Image Acquisition Toolbox is to look at a simple example. This chapter illustrates the basic steps required to create an image acquisition application by implementing a simple motion detection application. The example contains cross-references to other sections in the documentation that provide more in-depth discussions of the relevant concepts.

Example: Basic Image Acquisition Procedure (p. 1-2)

Presents a step-by-step approach to using the toolbox to create an image acquisition application

Example: Basic Image Acquisition Procedure

This section illustrates the basic steps required to create an image acquisition application by implementing a simple motion detection application. The application detects movement in a scene by performing a pixel-to-pixel comparison in pairs of incoming image frames. If nothing moves in the scene, pixel values remain the same in each frame. When something moves in the image, the application displays the pixels that have changed values.

The example highlights how you can use the Image Acquisition Toolbox to create a working image acquisition application with only a few lines of code.

Note To run the sample code in this example, you must have an image acquisition device connected to your system. The device can be a professional grade image acquisition device, such as a frame grabber, or a generic Windows image acquisition device, such as a Webcam. The code can be used with various types of devices with only minor changes.

Overview

To use the Image Acquisition Toolbox to acquire image data, you must perform the following basic steps:

Step 1:   Install and configure your image acquisition device

Step 2:   Retrieve information that uniquely identifies your image acquisition device to the Image Acquisition Toolbox

Step 3:   Create a video input object

Step 4:   Preview the video stream (Optional)

Step 5:   Configure image acquisition object properties (Optional)

Step 6:   Acquire image data

Step 7:   Clean up

The following sections implement each step.

Step 1: Install Your Image Acquisition Device

Follow the setup instructions that come with your image acquisition device. Setup typically involves

- Installing the frame grabber board in your computer.
- Installing any software drivers required by the device. These are supplied by the device vendor.
- Connecting a camera to a connector on the frame grabber board.
- Verifying that the camera is working properly by running the application software that came with the camera and viewing a live video stream.

Generic Windows image acquisition devices, such as Webcams and digital video camcorders, typically do not require the installation of a frame grabber board. You connect these devices directly to your computer via a USB or FireWire port.

After installing and configuring your image acquisition hardware, start MATLAB on your computer by double-clicking the icon on your desktop. You do not need to perform any special configuration of MATLAB to perform image acquisition.

Step 2: Retrieve Hardware Information

In this step, you get several pieces of information that the toolbox needs to uniquely identify the image acquisition device you want to access. You use this information when you create an image acquisition object, described in "Step 3: Create a Video Input Object" on page 1-6. The following table lists this information. You use the imaqhwinfo function to retrieve each item.

Adaptor name	An *adaptor* is the software that the toolbox uses to communicate with an image acquisition device via its device driver. The toolbox includes adaptors for certain vendors of image acquisition equipment and for particular classes of image acquisition devices. See "Adaptor Name" on page 1-4 for more information.
Device ID	The *device ID* is a number that the adaptor assigns to uniquely identify each image acquisition device with which it can communicate. See "Device ID" on page 1-5 for more information. **Note:** Specifying the device ID is optional; the toolbox uses the first available device ID as the default.
Video format	The *video format* specifies the image resolution (width and height) and other aspects of the video stream. Image acquisition devices typically support multiple video formats. See "Video Formats" on page 1-6 for more information. **Note:** Specifying the video format is optional; the toolbox uses one of the supported formats as the default.

Adaptor Name

To determine the name of the adaptor, enter the imaqhwinfo function at the MATLAB prompt without any arguments.

```
imaqhwinfo
ans =

    InstalledAdaptors: {'matrox'  'winvideo'}
        MATLABVersion: '6.5.1 (R13+)'
          ToolboxName: 'Image Acquisition Toolbox'
       ToolboxVersion: '1.0 (R13+)'
```

In the data returned by `imaqhwinfo`, the `InstalledAdaptors` field lists the adaptors that are available on your computer. In this example, `imaqhwinfo` found two adaptors available on the computer: `'matrox'` and `'winvideo'`. The listing on your computer might contain only one adaptor name. Select the adaptor name that provides access to your image acquisition device. For more information, see "Determining the Device Adaptor Name" on page 3-2.

Device ID

To find the device ID of a particular image acquisition device, enter the `imaqhwinfo` function at the MATLAB prompt, specifying the name of the adaptor as the only argument. (You found the adaptor name in the first call to `imaqhwinfo`, described in "Adaptor Name" on page 1-4.)

Note This example uses the adaptor for Matrox devices. You should substitute the name of the adaptor you would like to use.

```
info = imaqhwinfo('matrox')
info =

       AdaptorDllName: [1x73 char]
    AdaptorDllVersion: '1.0 (R13+)'
          AdaptorName: 'matrox'
            DeviceIDs: {[1]}
           DeviceInfo: [1x1 struct]
```

In the data returned, the `DeviceIDs` field is a cell array containing the device IDs of all the devices accessible through the specified adaptor.

Getting More Information About a Device. You can get more information about a particular device in the `DeviceInfo` field. This field is a structure array where each structure describes a particular device. This can be helpful if there is more than one device listed in `DeviceIDs`. To view the device information for a particular adaptor, reference the structure in the array or enter the `imaqhwinfo` function at the MATLAB prompt, specifying the adaptor name and device ID as arguments.

```
dev_info = imaqhwinfo('matrox',1)

dev_info =

            DefaultFormat: 'M_RS170'
     DeviceFileSupported: 1
              DeviceName: 'Orion'
                DeviceID: 1
       ObjectConstructor: 'videoinput('matrox', 1)'
        SupportedFormats: {1x10 cell}
```

Video Formats

You can optionally specify the video format of the video input object. To determine which video formats an image acquisition device supports, look in the SupportedFormats field of the DeviceInfo structure returned by the imaqhwinfo function. For more information, see "Determining Supported Video Formats" on page 3-6.

The example application uses the default video format.

Step 3: Create a Video Input Object

In this step you create the video input object that the toolbox uses to represent the connection between MATLAB and an image acquisition device. Using the properties of a video input object, you can control many aspects of the image acquisition process.

To create a video input object, enter the videoinput function at the MATLAB prompt. The videoinput function uses the adaptor name, device ID, and video format that you retrieved in step 2 to create the object. The adaptor name is the only required argument; the videoinput function can use defaults for the device ID and video format. For more information about image acquisition objects, see Chapter 3, "Connecting to Hardware."

```
vid = videoinput('matrox');
```

Note Substitute the adaptor name of the image acquisition device available on your system. To find the default `videoinput` function syntax for a device, look at the `ObjectConstructor` field of the `DeviceInfo` structure returned by the `imaqhwinfo` function. For more information, see "Getting More Information About a Device" on page 1-5.

To view a summary of the video input object you just created, enter `vid` at the MATLAB command prompt. The summary information shows many of the characteristics of the object, such as the number of frames that will be captured with each trigger, the trigger type, and the current state of the object. You can use video input object properties to control many of these characteristics. See "Step 5: Configure Object Properties (Optional)" on page 1-10 for more information.

```
vid

Summary of Video Input Object Using 'Orion'.

   Acquisition Source(s):  CH0, CH1, CH2, CH3, CH4, CH5, CH6, and
                           CH7 are available.

  Acquisition Parameters:  'CH0' is the current selected source.
                           10 frames per trigger using the selected source.
                           'M_RS170' video data to be logged upon START.
                           Grabbing first of every 1 frame(s).
                           Log data to 'memory' on trigger.

      Trigger Parameters:  1 'immediate' trigger(s) on START.

                  Status:  Waiting for START.
                           0 frames acquired since starting.
                           0 frames available for GETDATA.
```

Specifying the Video Format

This example does not specify the video format of the video input object; it uses the default video format selected by the toolbox. The default video format is listed in the `DefaultFormat` field of the `DeviceInfo` structure. If you want to use a different video format, you must specify the format when you create the object. See "Determining Supported Video Formats" on page 3-6 for more information.

Step 4: Preview the Video Stream (Optional)

After you create the video input object, MATLAB is able to access the image acquisition device and is ready to acquire data. However, before you begin, you might want to see a preview of the video stream to make sure that the image is satisfactory. For example, you might want to change the position of the camera, change the lighting, correct the focus, or make some other change to your image acquisition setup.

Note This step is optional at this point in the procedure because you can preview a video stream at any time after you create a video input object.

To preview the video stream in this example, enter the `preview` function at the MATLAB prompt, specifying the video input object created in step 3 as an argument.

```
preview(vid)
```

The `preview` function opens a window on your screen and displays a window containing the live video stream. While a preview window is open, the object sets the value of the `Previewing` property to `'on'`. If you change characteristics of the image by setting image acquisition object properties, the image displayed in the preview window reflects the change. The following figure shows the preview window for the example.

Name of image
acquisition device
adaptor and device ID

Live
video
image

Timestamp of
current frame
being displayed

Resolution

State of the video input object
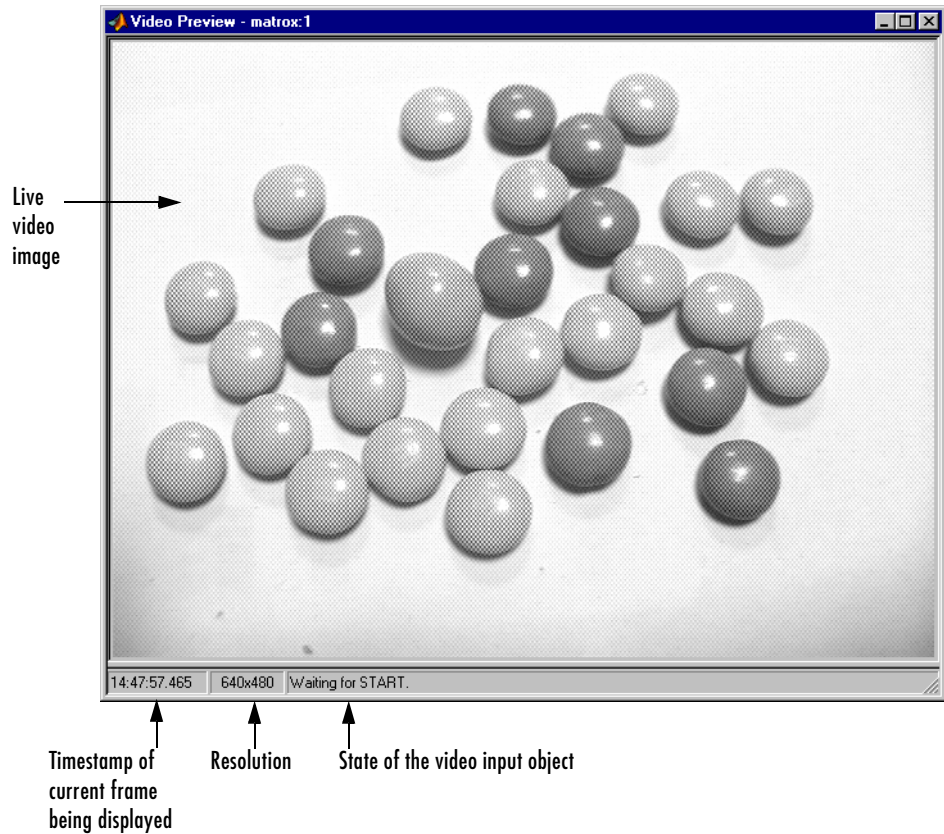
Preview Window

To close the preview window, click the **Close** button in the title bar or use the
closepreview function, specifying the video input object as an argument.

```
closepreview(vid)
```

Step 5: Configure Object Properties (Optional)

After creating the video input object and previewing the video stream, you might want to modify characteristics of the image or other aspects of the acquisition process. You accomplish this by setting the values of image acquisition object properties. This section

- Describes the types of image acquisition objects used by the toolbox
- Describes how to view all the properties supported by these objects, with their current values
- Describes how to set the values of object properties

Types of Image Acquisition Objects

The toolbox uses two types of objects to represent the connection with an image acquisition device:

- Video input objects
- Video source objects

A video input object represents the connection between MATLAB and a video acquisition device at a high level. The properties supported by the video input object are the same for every type of device. You created a video input object using the videoinput function in step 3.

When you create a video input object, the toolbox automatically creates one or more video source objects associated with the video input object. Each video source object represents a collection of one or more physical data sources that are treated as a single entity. The number of video source objects the toolbox creates depends on the device and the video format you specify. At any one time, only one of the video source objects, called the *selected* source, can be active. This is the source used for acquisition. For more information about these image acquisition objects, see "Creating Image Acquisition Objects" on page 3-9.

Viewing Object Properties

To view a complete list of all the properties supported by a video input object or a video source object, use the get function. To list the properties of the video input object created in step 3, enter this code at the MATLAB prompt.

```
get(vid)
```

The get function lists all the properties of the object with their current values.

```
General Settings:
    DeviceID = 1
    DiskLogger = []
    DiskLoggerFrameCount = O
    EventLog = [1xO struct]
    FrameGrabInterval = 1
    FramesAcquired = O
    FramesAvailable = O
    FramesPerTrigger = 10
    Logging = off
    LoggingMode = memory
    Name = M_RS17O-matrox-1
    NumberOfBands = 1
    Previewing = on
    ReturnedColorSpace = grayscale
    ROIPosition = [O O 640 480]
    Running = off
    Tag =
    Timeout = 10
    Type = videoinput
    .
    .
    .
```

To view the properties of the currently selected video source object associated with this video input object, use the getselectedsource function in conjunction with the get function. The getselectedsource function returns the currently active video source. To list the properties of the currently selected video source object associated with the video input object created in step 3, enter this code at the MATLAB prompt.

```
get(getselectedsource(vid))
```

The get function lists all the properties of the object with their current values.

> **Note** Video source object properties are device specific. The list of properties supported by the device connected to your system might differ from the list shown in this example.

```
General Settings:
  Parent = [1x1 videoinput]
  Selected = on
  SourceName = CHO
  Tag =
  Type = videosource

Device Specific Properties:
  InputFilter = none
  UserOutputBit3 = on
  UserOutputBit4 = on
  XScaleFactor = 1
  YScaleFactor = 1
```

Setting Object Properties

To set the value of a video input object property or a video source object property, you can use the set function or you can reference the object property as you would a field in a structure, using dot notation.

Some properties are read only; you cannot set their values. These properties typically provide information about the state of the object. Other properties become read only when the object is running. To view a list of all the properties you can set, use the set function, specifying the object as the only argument.

To implement continuous image acquisition, the example sets the TriggerRepeat property to Inf. To set this property using the set function, enter this code at the MATLAB prompt.

```
set(vid,'TriggerRepeat',Inf);
```

To help the application keep up with the incoming video stream while processing data, the example sets the FrameGrabInterval property to 5. This specifies that the object acquire every fifth frame in the video stream. (You might need to experiment with the value of the FrameGrabInterval property to find a value that provides the best response with your image acquisition

setup.) This example shows how you can set the value of an object property by referencing the property as you would reference a field in a MATLAB structure.

```
vid.FrameGrabInterval = 5;
```

To set the value of a video source object property, you must first use the `getselectedsource` function to retrieve the object. (You can also get the selected source by searching the video input object `Source` property for the video source object that has the `Selected` property set to `'on'`.)

To illustrate, the example assigns a value to the `Tag` property.

```
vid_src = getselectedsource(vid);

set(vid_src,'Tag','motion detection setup');
```

Step 6: Acquire Image Data

After you create the video input object and configure its properties, you can acquire data. This is typically the core of any image acquisition application, and it involves these steps:

- **Starting the video input object** — You start an object by calling the `start` function. Starting an object prepares the object for data acquisition. For example, starting an object locks the values of certain object properties (they become read only). Starting an object does not initiate the acquiring of image frames, however. The initiation of data logging depends on the execution of a trigger.

  The example calls the `start` function to start the video input object. Objects stop when they have acquired the requested number of frames. Because the example specifies a continuous acquisition, you must call the `stop` function to stop the object.

- **Triggering the acquisition** — To acquire data, a video input object must execute a trigger. Triggers can occur in several ways, depending on how the `TriggerType` property is configured. For example, if you specify an immediate trigger, the object executes a trigger automatically, immediately after it starts. If you specify a manual trigger, the object waits for a call to

the `trigger` function before it initiates data acquisition. For more information, see Chapter 4, "Acquiring Image Data."

In the example, because the `TriggerType` property is set to `'immediate'` (the default) and the `TriggerRepeat` property is set to `Inf`, the object automatically begins executing triggers and acquiring frames of data, continuously.

- **Bringing data into the MATLAB workspace** — The toolbox stores acquired data in a memory buffer, a disk file, or both, depending on the value of the video input object `LoggingMode` property. To work with this data, you must bring it into the MATLAB workspace. To bring multiple frames into the workspace, use the `getdata` function. Once the data is in the MATLAB workspace, you can manipulate it as you would any other data. For more information, see Chapter 5, "Working with Acquired Image Data."

Note The toolbox provides a convenient way to acquire a single frame of image data that doesn't require starting or triggering the object. See "Bringing a Single Frame into the Workspace" on page 5-10 for more information.

To run the example, enter this code at the MATLAB prompt.The example loops until a specified number of frames have been acquired. In each loop iteration, the example calls `getdata` to bring the two most recent frames into the MATLAB workspace. To detect motion, the example subtracts one frame from the other, creating a difference image, and then displays it. Pixels that have changed values in the acquired frames will have nonzero values in the difference image.

The `getdata` function removes frames from the memory buffer when it brings them into the MATLAB workspace. It is important to move frames from the memory buffer into the MATLAB workspace in a timely manner. If you do not move the acquired frames from memory, you can quickly exhaust all the memory available on your system.

The application creates a MATLAB figure and sets the `DoubleBuffer` property. This is not directly related to image acquisition but is included to ensure a smooth display.

Note The application uses functions in the Image Processing Toolbox.

```
figure; % Ensure smooth display
set(gcf,'doublebuffer','on');

start(vid)

while(vid.FramesAcquired<=100)
    data = getdata(vid,2);
    diff_im = imabsdiff(data(:,:,:,1),data(:,:,:,2));
    imshow(diff_im);
end

stop(vid)
```

The following figure shows how the example displays detected motion. In the
figure, areas representing movement are displayed.
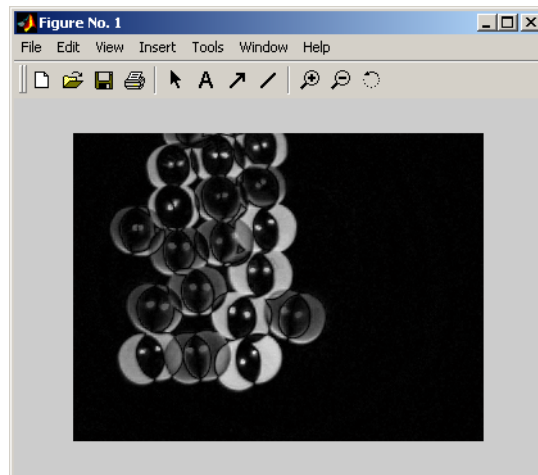
Figure Window Displayed by Application

Image Data in the MATLAB Workspace

In the example, the getdata function returns a 480-by-640-by-1-by-10 array of 8-bit data (uint8). The height and width of the array are primarily determined by the video resolution of the video format. On systems that support the definition of a region-of-interest (ROI), the values of the ROIPosition property can supersede the video resolution.

The third dimension represents the number of color bands in the image. Because the example data is a grayscale image, the third dimension is 1. For RGB formats, image frames have three bands: red is the first, green is the second, and blue is the third. The fourth dimension represents the number of frames that have been acquired from the video stream.

Note Devices typically express video resolution as column-by-row; MATLAB expresses matrix dimensions as row-by-column.

```
whos
  Name            Size                    Bytes  Class

  data            4-D                   3072000  uint8 array
  dev_info        1x1                      1601  struct array
  info            1x1                      2467  struct array
  vid             1x1                      1138  videoinput object
  vid_src         1x1                       726  videosource object
```

Step 7: Clean Up

When you finish using your image acquisition objects, you can remove them from memory and clear the MATLAB workspace of the variables associated with these objects.

```
delete(vid)
clear
close(gcf)
```

For more information, see "Deleting Image Acquisition Objects" on page 3-27.

2

Introduction

This chapter describes the Image Acquisition Toolbox and its components.

Overview

The Image Acquisition Toolbox is a collection of functions that extend the capability of the MATLAB® numeric computing environment.

The Image Acquisition Toolbox implements an object-oriented approach to image acquisition. Using toolbox functions, you create an object that represents the connection between MATLAB and specific image acquisition devices. Using properties of the object you can control various aspects of the acquisition process, such as the amount of video data you want to capture. Chapter 3, "Connecting to Hardware," describes how to create objects.

Once you establish a connection to a device, you can acquire image data by executing a trigger. In the toolbox, all image acquisition is initiated by a trigger. The toolbox supports several types of triggers that let you control when an acquisition takes place. For example, using hardware triggers you can synchronize an acquisition with an external device. Chapter 4, "Acquiring Image Data," describes how to trigger the acquisition of image data.

To work with the data you acquire, you must bring it into the MATLAB workspace. When the frames are acquired, the toolbox stores them in a memory buffer. The toolbox provides several ways to bring one or more frames of data into the workspace where you can manipulate it as you would any other multidimensional numeric array. Chapter 5, "Working with Acquired Image Data," describes this process.

Finally, you can enhance your image acquisition application by using event callbacks. The toolbox has defined certain occurrences, such as the triggering of an acquisition, as events. You can associate the execution of a particular function with a particular event. Chapter 6, "Using Events and Callbacks," describes this process.

Toolbox Components

The toolbox uses components called hardware device adaptors to connect to devices through their drivers. The toolbox includes adaptors that support devices produced by several vendors of image acquisition equipment. In addition, the toolbox includes an adaptor for generic Windows video acquisition devices.

The following figure shows these components and their relationship.
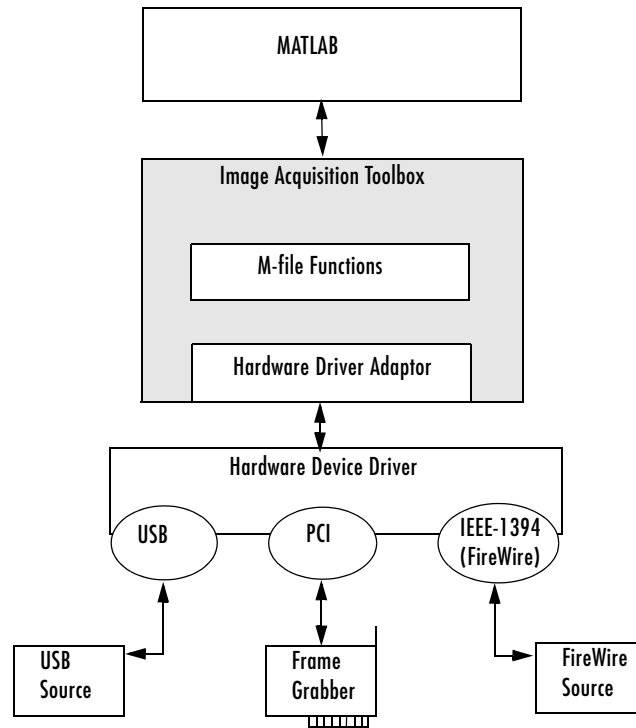
Image Acquisition Toolbox Components

System Requirements

The Image Acquisition Toolbox supports vendors of professional grade image acquisition equipment, such as Matrox and Data Translation. In addition, the toolbox supports many Windows video acquisition devices that provide WDM (Windows Driver Model) or VFW (Video for Windows) drivers, such as USB and FireWire (IEEE1394, i.LINK®) Web cameras, digital video (DV) camcorders, and TV tuner cards.

For the latest information about supported hardware, visit the Image Acquisition Toolbox product page at The MathWorks Web site (www.mathworks.com/products/imaq).

Setting Up Image Acquisition Hardware

In a typical image acquisition setup, an image acquisition device, such as a camera, is connected to a computer via an image acquisition board, such as a frame grabber, or via a Universal Serial Bus (USB) port or FireWire (IEEE-1394) port.

To acquire image data, you must perform the setup required by your particular image acquisition device.

For frame grabbers, also known as imaging boards, setup typically involves the following tasks:

- Installing an image acquisition board in your computer
- Installing any software drivers required by the device. These are supplied by the device vendor.
- Connecting the camera, or other image acquisition device, to the imaging board connector
- Verifying that the camera is working properly by running the application software that came with the camera and viewing a live video stream

Setup for generic Windows video acquisition devices is the same except these devices typically do not require the installation of an image acquisition board. These devices connect directly to the USB or FireWire port on your system.

After installing and configuring your image acquisition hardware, start MATLAB on your computer by double-clicking the icon on your desktop. You do not need to perform any special configuration of MATLAB to acquire data.

Resetting Your Image Acquisition Hardware

To return MATLAB and your image acquisition hardware to a known state, where no image acquisition objects exist and the hardware is not configured, use the `imaqreset` function.

If you connect another image acquisition device to your system after MATLAB is started, you can use `imaqreset` to make the toolbox aware of the new hardware.

A Note About Frame Rates and Processing Speed

The frame rate describes how fast an image acquisition device provides data, typically measured as frames per second.

Devices that support industry-standard video formats must provide frames at the rate specified by the standard. For RS170 and NTSC, the standard dictates a frame rate of 30 frames per second (30 Hz). The CCIR and PAL standards define a frame rate of 25 Hz. Nonstandard devices can be configured to operate at higher rates. Generic Windows image acquisition devices, such as Webcams, might support many different frame rates. Depending on the device being used, the frame rate might be configurable using a device-specific property of the image acquisition object.

The rate at which the Image Acquisition Toolbox can process images depends on the processor speed, the complexity of the processing algorithm, and the frame rate. Given a fast processor, a simple algorithm, and a frame rate tuned to the acquisition setup, the Image Acquisition Toolbox can process data as it comes in.

Previewing Data

After you connect MATLAB to the image acquisition device (see "Connecting to Hardware" on page 3-1), you can view the video stream using the Image Acquisition Toolbox preview window. Previewing the video stream can be helpful to make sure that the image being captured is satisfactory. For example, by looking at a preview, you can verify that the lighting and focus are correct. (To change aspects of the image, you must use properties of the video input object. See "Configuring Image Acquisition Object Properties" on page 3-17 for more information.)

This section describes how to

- Open a preview window
- Close a preview window

Opening a Preview Window

To open a preview window, use the preview function. This window displays a view of the live video stream.

You can open multiple preview windows at the same time to view images being returned by different image acquisition devices. If you change characteristics of the image, by setting image acquisition object properties, the image displayed in the preview window reflects the change.

The following example opens a preview window for the video input object vid.

```
preview(vid);
```

Name of image
acquisition device
adaptor and device ID

Live video image

Video Preview - matrox:1

14:47:57.465 | 640x480 | Waiting for START.

Timestamp of   Resolution   State of the video input object
last frame
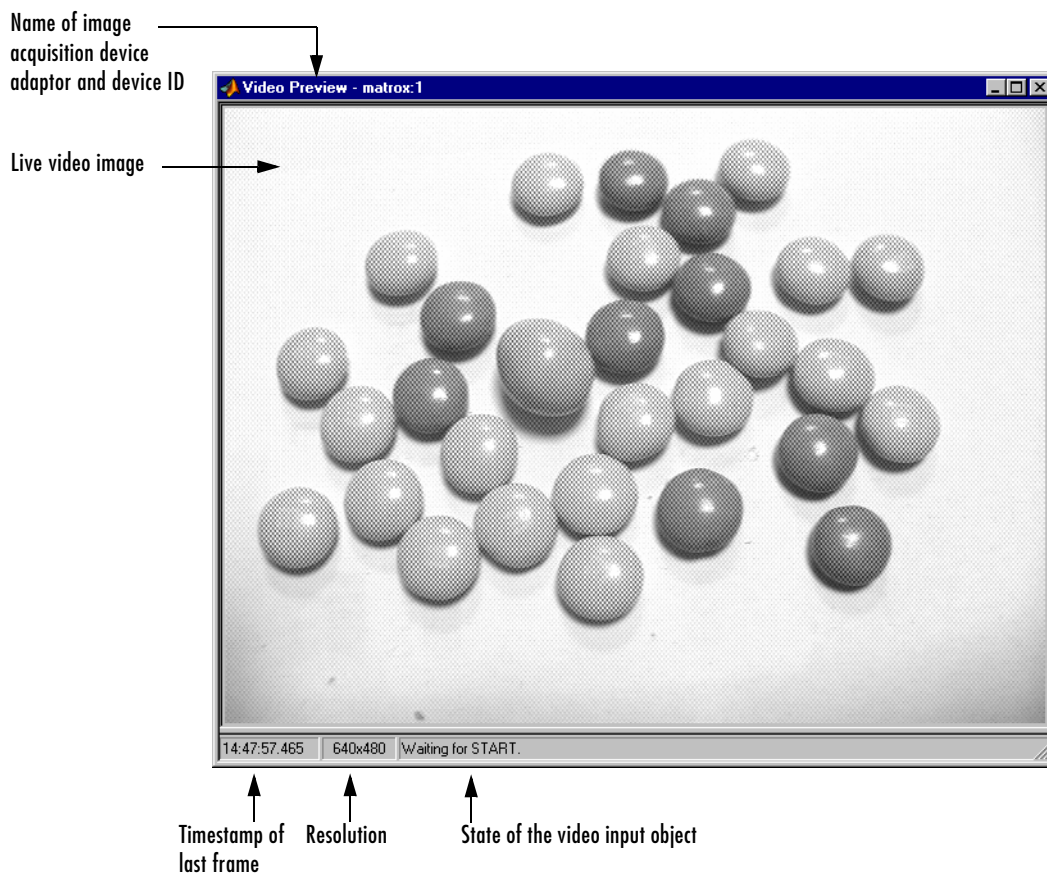
Closing a Preview Window

To close a particular preview window, use the closepreview function,
specifying the video input object as an argument.

```
closepreview(vid)
```

To close all currently open preview windows, use the closepreview function
without any arguments.

```
closepreview
```

Troubleshooting

If, after installing the Image Acquisition Toolbox and using it to establish a connection to your image acquisition device, you are unable to acquire data or encounter other problems, try these troubleshooting steps first. They might help fix the problem.

1 Verify that your image acquisition hardware is functioning properly.

2 If the hardware is functioning properly, verify that you are using a hardware device driver that is compatible with the Image Acquisition Toolbox.

The following sections describe how to perform these steps for Matrox devices, Data Translation devices, and generic Windows video acquisition devices.

If you are encountering problems with the preview window, see "Troubleshooting a Preview Window" on page 2-13.

Troubleshooting Matrox Hardware

If you are having trouble using the Image Acquisition Toolbox with a supported Matrox frame grabber, perform these troubleshooting steps, in the order specified.

1 Verify that your image acquisition hardware is functioning properly.

For Matrox devices, run the application that came with your hardware, Matrox Intellicam, and verify that you can receive live video.

2 If your hardware is functioning properly, verify that you are using a hardware device driver that is compatible with the toolbox.

The Image Acquisition Toolbox is only compatible with specific driver versions provided with the Matrox Imaging Library (MIL) or MIL-Lite software and is not guaranteed to work with any other versions. Find out which driver version you are using on your system. (See "Determining the Driver Version for Matrox Devices" on page 2-9 to learn how.) Verify that the version is compatible with the Image Acquisition Toolbox by checking the list of supported drivers on the Image Acquisition Toolbox product page at The MathWorks Web site (www.mathworks.com/products/imaq). If you

discover that you are using an unsupported driver, visit the Matrox Web site (www.matrox.com) to download the latest drivers.

Determining the Driver Version for Matrox Devices

To determine which Matrox Imaging Library version you are using, run the Matrox MIL Configuration utility. You can access this software through the Windows **Start** button.

Select **Start->Programs->Matrox Imaging Products->MIL Configuration**.

The software version is listed on the **Information** tab.

Device driver version
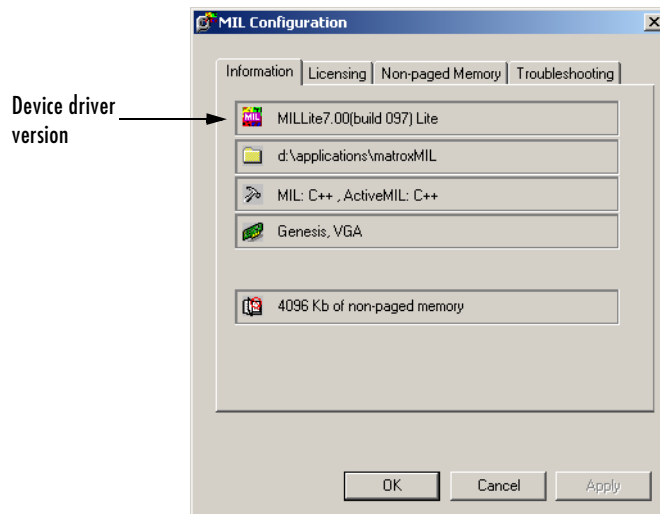
Matrox MIL Configuration Utility

Troubleshooting Data Translation Hardware

If you are having trouble using the Image Acquisition Toolbox with a supported Data Translation frame grabber, perform these troubleshooting steps, in the order specified.

1 Verify that your image acquisition hardware is functioning properly.

For Data Translation devices, run the application that came with your hardware and verify that you can receive live video.

2 If your hardware is functioning properly, verify that you are using a hardware device driver that is compatible with the toolbox.

The Image Acquisition Toolbox is only compatible with specific driver versions provided by Data Translation with the Imaging Omni CD and is not guaranteed to work with any other versions. Determine which driver version you are using on your system and verify that the version is compatible with the Image Acquisition Toolbox by checking the list of supported drivers on the Image Acquisition Toolbox product page at The MathWorks Web site (`www.mathworks.com/products/imaq`). If you discover that you are using an unsupported driver, visit the Data Translation Web site (`www.datatranslation.com`) to download the latest drivers.

Troubleshooting Windows Video Hardware

If you are having trouble using the Image Acquisition Toolbox with a supported Windows video acquisition device, perform these recommended troubleshooting steps, in the order specified.

1 Verify that your image acquisition hardware is functioning properly.

For Windows devices, run the application that came with your hardware and verify that you can receive live video.

You can also verify your hardware by running the Microsoft image capture application, `AMCap.exe`, which is included with the toolbox distribution. Go to the `$MATLAB/toolbox/imaq/imaq` directory, where `$MATLAB` is your top-level installation directory, and double-click on `AMCAP.exe`. If you can start the utility, run the utility, and close the utility without encountering any errors, the toolbox should be able to operate with your image acquisition

device. If you encounter errors, resolve them before attempting to use the toolbox with the device.

2 If your hardware is functioning properly, verify that you are using hardware device drivers that are compatible with the toolbox.

Note For Windows image acquisition devices, you must also have the correct version of Microsoft DirectX installed on your computer.

The Image Acquisition Toolbox is only compatible with WDM (Windows Driver Model) or VFW (Video for Windows) drivers. Contact the hardware manufacturer to determine if the driver provided with your hardware conforms to these driver classes. Determine which driver version you are using on your system and verify that the version is compatible with the Image Acquisition Toolbox by checking the list of supported drivers on the Image Acquisition Toolbox product page at The MathWorks Web site (www.mathworks.com/products/imaq). If you discover that you are using an unsupported driver, visit the hardware manufacturer's Web site for the latest drivers.

The Image Acquisition Toolbox is only compatible with specific versions of the Microsoft DirectX multimedia technology and is not guaranteed to work with any other versions. Find out which driver version you are using on your system. (See "Determining the Microsoft DirectX Version" to learn how.) For a list of the versions compatible with the Image Acquisition Toolbox, refer to the product page at The MathWorks Web site (www.mathworks.com/products/imaq). Visit the Microsoft DirectX Web site (www.microsoft.com/directx/) for the latest version of DirectX.

Determining the Microsoft DirectX Version

To determine which version of Microsoft DirectX you are using, run the DirectX Diagnostic Tool. You can access this software through the Windows **Start** button.

Select **Start->Run**.

In the **Run** dialog box, launch the **DirectX Diagnostic Tool** by opening the dxdiag program.
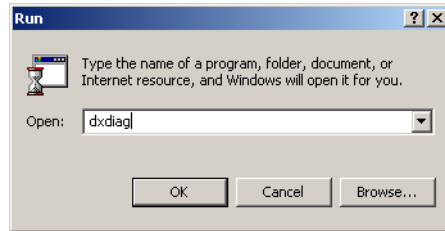
In the **DirectX Diagnostic Tool**, the Microsoft DirectX version is listed on the **System** tab under the **System Information** section.

DirectX version →
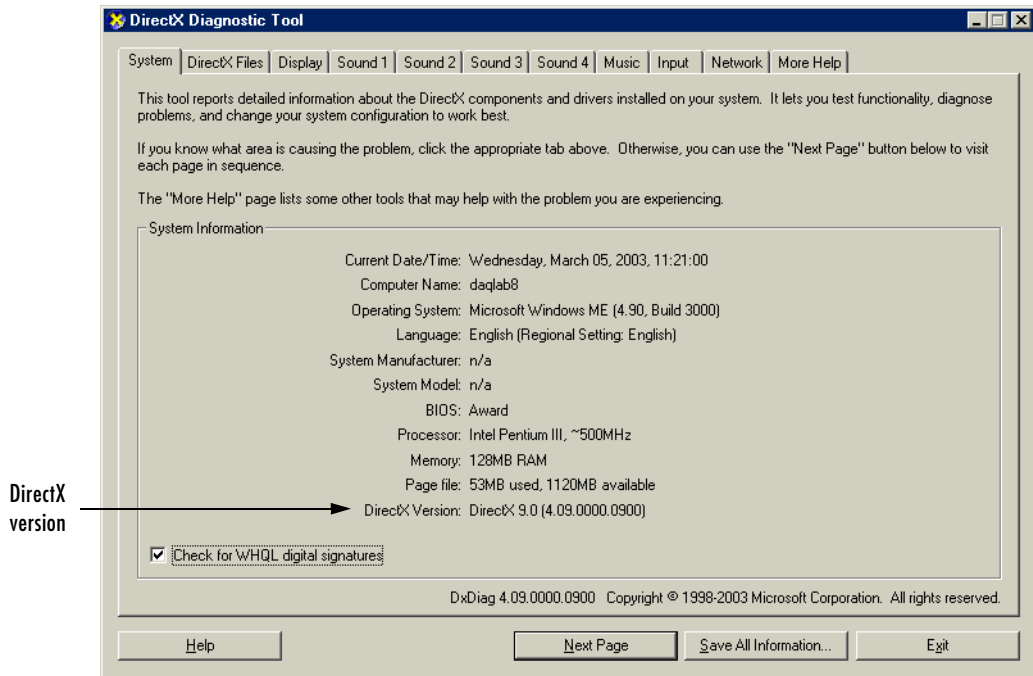
DirectX Diagnostic Tool

Troubleshooting a Preview Window

When previewing the video stream, if you encounter a problem, try one of the following solutions.

Problem	Possible Solutions
Preview window stops running.	When this occurs, try any of the following solutions: • Close the preview window and reopen it. • Verify that your image acquisition device is working properly. Close MATLAB and run the application that came with your device. • Make sure no other application is using the device.
Preview displays blank, gray window.	When this occurs, try any of the following solutions: • Close the preview window and reopen it. • Check memory usage. It is possible that there is not enough memory available for the incoming image data. To increase the memory allocation, use the imaqmem function and specify a higher value for the FrameMemoryLimit. • Make sure no other application is using the device.
Preview window displays dropped frames message.	When this occurs, try any of the following solutions: • Close the preview window and reopen it. • Check memory usage. It is possible that there is not enough memory available for the incoming image data. To increase the memory allocation, use the imaqmem function and specify a higher value for the FrameMemoryLimit.

3

Connecting to Hardware

To connect to an image acquisition device from within MATLAB, you must create a video input object. This object represents the connection between MATLAB and the device. You can use object properties to control various aspects of the acquisition. Before you can create the object, you need several pieces of information about the device that you want to connect to.

This chapter describes tasks related to establishing a connection between MATLAB and an image acquisition device.

Getting Hardware Information

To access an image acquisition device, the toolbox needs several pieces of information:

- The name of the *adaptor* the toolbox uses to connect to the image acquisition device
- The *device ID* of the device you want to access (optional)
- The *video format* of the video stream (optional)

You use the imaqhwinfo function to retrieve this information, as described in the following sections:

- "Determining the Device Adaptor Name" on page 3-2
- "Determining the Device ID" on page 3-4
- "Determining Supported Video Formats" on page 3-6

Note When using imaqhwinfo to get information about a device, especially devices that use a Video for Windows (VFW) driver, you might encounter dialog boxes reporting an assertion error. Make sure that the software drivers are installed correctly and that the acquisition device is connected to the computer.

Determining the Device Adaptor Name

An adaptor is the software the toolbox uses to communicate with an image acquisition device via its device driver. The toolbox includes adaptors for some vendors of image acquisition equipment and for particular classes of image acquisition devices. The following table lists the adaptors included with the Image Acquisition Toolbox. For more information, see "System Requirements" on page 2-3.

Adaptors Included with the Image Acquisition Toolbox

Adaptor Name	Description
`'dt'`	Adaptor for image acquisition devices produced by Data Translation, Inc.
`'matrox'`	Adaptor for image acquisition devices produced by Matrox Electronic Systems, Ltd.
`'winvideo'`	Adaptor for devices that provide a WDM (Windows Driver Model) or VFW (Video for Windows) driver including USB and FireWire (IEEE-1394) cameras

To determine which adaptors are available on your system, call the `imaqhwinfo` function. The `imaqhwinfo` function returns information about the toolbox software and lists the adaptors available on the system in the `InstalledAdaptors` field. In this example, there are two adaptors available on the system.

```
imaqhwinfo
ans =

    InstalledAdaptors: {'matrox'  'winvideo'}
        MATLABVersion: '6.5.1 (R13+)'
          ToolboxName: 'Image Acquisition Toolbox'
       ToolboxVersion: '1.0 (R13+)'
```

Note While every adaptor supported by the Image Acquisition Toolbox is installed with the toolbox, `imaqhwinfo` only lists adaptors in the `InstalledAdaptors` field that are loadable. That is, the device drivers required by the vendor are installed on the system. Note, however, that inclusion in the `InstalledAdaptors` field does not necessarily mean that an adaptor is connected to a device.

Determining the Device ID

The adaptor assigns a unique number to each device with which it can communicate. The adaptor assigns the first device it detects the device ID 1, the second it detects the device ID 2, and so on.

To find the device ID of a particular image acquisition device, call the `imaqhwinfo` function, specifying the name of the adaptor as the only argument. When called with this syntax, `imaqhwinfo` returns a structure containing information about all the devices available through the specified adaptor.

In this example, the `imaqhwinfo` function returns information about all the devices available through the Matrox adaptor.

```
info = imaqhwinfo('matrox');

info =

        AdaptorDllName: [1x74 char]
     AdaptorDllVersion: '6.5 (R13+)'
           AdaptorName: 'matrox'
              DeviceIDs: {[1]}
            DeviceInfo: [1x1 struct]
```

The fields in the structure returned by `imaqhwinfo` provide the following information.

Device Information Returned by imaqhwinfo

Field	Description
AdaptorDllName	Text string that identifies the name of the adaptor dynamic link library (DLL)
AdaptorDllVersion	Information about the version of the adaptor DLL
AdaptorName	Name of the adaptor
DeviceIDs	Cell array containing the device IDs of all the devices accessible through this adaptor
DeviceInfo	Array of device information structures. See "Getting More Information About a Particular Device" on page 3-5 for more information.

Getting More Information About a Particular Device

If an adaptor provides access to multiple devices, you might need to find out more information about the devices before you can select a device ID. The DeviceInfo field is an array of device information structures. Each device information structure contains detailed information about a particular device available through the adaptor.

To view the information for a particular device, call imaqhwinfo again, this time specifying a device ID as an argument.

```
dev_info = imaqhwinfo('matrox',1)

dev_info =

          DefaultFormat: 'M_RS170'
    DeviceFileSupported: 1
             DeviceName: 'Orion'
               DeviceID: 1
      ObjectConstructor: 'videoinput('matrox', 1)'
       SupportedFormats: {1x10 cell}
```

The fields in the device information structure provide the following information about a device.

Device Information Structure Returned by imaqhwinfo

Field	Description
DefaultFormat	Text string that identifies the video format used by the device if none is specified at object creation time
DeviceFileSupported	If set to 1, the device supports device configuration files; otherwise 0. See "Using Device Configuration Files (Camera Files)" on page 3-14 for more information.
DeviceName	Descriptive text string, assigned by the adaptor, that identifies the device
DeviceID	ID assigned to the device by the adaptor

Device Information Structure Returned by imaqhwinfo

Field	Description
ObjectConstructor	Default syntax you can use to create a video input object to represent this device. See "Creating Image Acquisition Objects" on page 3-9 for more information.
SupportedFormats	Cell array of strings that identify the video formats supported by the device. See "Determining Supported Video Formats" on page 3-6 for more information.

Determining Supported Video Formats

The video format specifies the characteristics of the images in the video stream, such as the image resolution (width and height), the industry standard used, and the size of the data type used to store pixel information.

Image acquisition devices typically support multiple video formats. You can specify the video format when you create the video input object to represent the connection to the device. See "Creating Image Acquisition Objects" on page 3-9 for more information.

Note Specifying the video format is optional; the toolbox uses one of the supported formats as the default.

To determine which video formats an image acquisition device supports, look in the SupportedFormats field of the DeviceInfo structure returned by the imaqhwinfo function. To view the information for a particular device, call imaqhwinfo, specifying the device ID as an argument.

```
dev_info = imaqhwinfo('matrox',1)

dev_info =

          DefaultFormat: 'M_RS170'
     DeviceFileSupported: 1
             DeviceName: 'Orion'
```

```
              DeviceID: 1
    ObjectConstructor: 'videoinput('matrox', 1)'
     SupportedFormats: {1x10 cell}
```

The DefaultFormat field lists the default format selected by the toolbox. The SupportedFormats field is a cell array containing text strings that identify all the supported video formats. The toolbox assigns names to the formats based on vendor-specific terminology. If you want to specify a video format when you create an image acquisition object, you must use one of the text strings in this cell array. See "Creating Image Acquisition Objects" on page 3-9 for more information.

```
celldisp(dev_info.SupportedFormats)

ans{1} =

M_RS170

ans{2} =

M_RS170_VIA_RGB

ans{3} =

M_CCIR

ans{4} =

M_CCIR_VIA_RGB

ans{5} =

M_NTSC

ans{6} =

M_NTSC_RGB

ans{7} =

M_NTSC_YC
```

```
ans{8} =

M_PAL

ans{9} =

M_PAL_RGB

ans{10} =

M_PAL_YC
```

Creating Image Acquisition Objects

After you get information about your image acquisition hardware, described in "Getting Hardware Information" on page 3-2, you can establish a connection to the device by creating an image acquisition object. The toolbox uses two types of image acquisition objects:

- Video input object
- Video source object

Video Input Objects

A video input object represents the connection between MATLAB and a video acquisition device at a high level. You must create the video input object using the videoinput function. See "Creating a Video Input Object" on page 3-10 for more information.

Video Source Objects

When you create a video input object, the toolbox automatically creates one or more video source objects associated with the video input object. Each video source object represents a collection of one or more physical data sources that are treated as a single entity. The number of video source objects the toolbox creates depends on the device and the video format you specify.

At any one time, only one of the video source objects, called the *selected* source, can be active. This is the source used for acquisition. The toolbox selects one of the video source objects by default, but you can change this selection. See "Specifying the Selected Video Source Object" on page 3-15 for more information. The following figure illustrates how a video input object acts as a container for one or more video source objects.
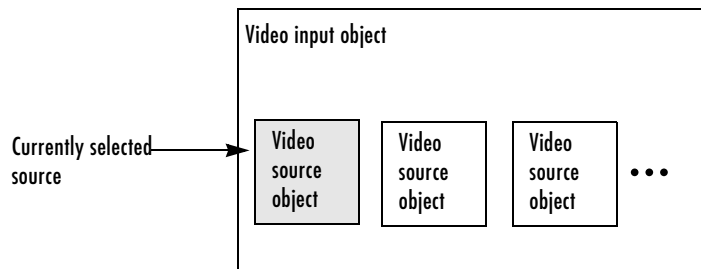
Relationship of Video Input Objects and Video Source Objects

For example, a Matrox frame grabber device can support eight physical connections, which Matrox calls channels. These channels can be configured in various ways, depending upon the video format. If you specify a monochrome video format, such as RS170, the toolbox creates eight video source objects, one object for each of the eight channels on the device. If you specify a color video format, such as NTSC RGB, the Matrox device uses three physical channels to represent one RGB connection, where each physical connection provides the red data, green data, and blue data separately. With this format, the toolbox only creates two video source objects for the same device.

Creating a Video Input Object

To create a video input object, call the videoinput function specifying the adaptor name, device ID, and video format. You retrieved this information using the imaqhwinfo function (described in "Getting Hardware Information" on page 3-2). The only required argument is the adaptor name. The toolbox can use default values for the device ID and video format.

This example creates a video input object to represent the connection to a Matrox image acquisition device. The imaqhwinfo function includes the default videoinput syntax in the ObjectConstructor field of the device information structure.

```
vid = videoinput('matrox');
```

This syntax uses the default video format listed in the DefaultFormat field of the data returned by imaqhwinfo. You can optionally specify the video format. See "Specifying the Video Format" on page 3-12 for more information.

Viewing a Summary of a Video Input Object

To view a summary of the characteristics of the video input object you created, enter the variable name you assigned to the object at the command prompt. For example, this is the summary for the object vid.

```
vid
```

① Summary of Video Input Object Using 'Orion'.

②   Acquisition Source(s): CH0, CH1, CH2, CH3, CH4, CH5, CH6, and
                                     CH7 are available.

③ Acquisition Parameters: 'CH0' is the current selected source.
                                     10 frames per trigger using the selected source.
                                     'M_RS170' video data to be logged upon START.
                                     Grabbing first of every 1 frame(s).
                                     Log data to 'memory' on trigger.

④   Trigger Parameters: 1 'immediate' trigger(s) on START.

⑤   Status: Waiting for START.
                                     0 frames acquired since starting.
                                     0 frames available for GETDATA.

The items in this list correspond to the numbered elements in the object summary:

1 The title of the summary includes the name of the image acquisition device this object represents. In the example, this is a Matrox Orion frame grabber.

2 The Acquisition Source section lists the name of all the video source objects associated with this video input object. For many objects, this list might only contain one video source object. In the example, the Matrox device supports eight physical input channels and, with the default video format, the toolbox creates a video source object for each connection. For an example showing the video source objects created with another video format, see "Specifying the Video Format" on page 3-12.

3 The Acquisition Parameters section lists the values of key video input object properties. These properties control various aspects of the acquisition, such as the number of frames to acquire and the location where acquired frames are stored. For information about these properties, see Chapter 4, "Acquiring Image Data."

4 The Trigger Parameters section lists the trigger type configured for the object and the number of times the trigger is to be executed. Trigger execution initiates data logging, and the toolbox supports several types of triggers. The example object is configured by default with an immediate trigger. For more information about configuring triggers, see Chapter 4, "Acquiring Image Data."

5 The Status section lists the current state of the object. A video input object can be in one of several states:

- Running or not running (stopped)
- Logging or not logging
- Previewing or not previewing

In the example, the object describes its state as `Waiting for START`. This indicates it is not running. For more information about the running state, see "Starting and Stopping a Video Input Object" on page 3-24. This section also reports how many frames of data have been acquired and how many frames are available in the buffer where the toolbox stores acquired frames. For more information about these parameters, see "Controlling Logging Parameters" on page 4-16.

Specifying the Video Format

You can optionally specify the format of the video stream when you create a video input object as a third argument to the `videoinput` function. This argument can take two forms:

- A text string specifying a video format
- A name of a device configuration file, also known as a camera file

The following sections describe these options. If you do not specify a video format, the `videoinput` function uses one of the video formats supported by the device. For Matrox and Data Translation devices, it chooses the RS170 video format. For Windows devices, it uses the first RGB format in the list of supported formats or, if no RGB formats are supported, the device's default format.

Using a Video Format String

To specify a video format as a text string, use the imaqhwinfo function to determine the list of supported formats. The imaqhwinfo function returns this information in the SupportedFormats field of the device information structure. See "Determining Supported Video Formats" on page 3-6 for more information.

In this example, each of the text strings is a video format supported by a Matrox device.

```
info = imaqhwinfo('matrox');

info.DeviceInfo.SupportedFormats

ans =
  Columns 1 through 4

   'M_RS170'    'M_RS170_VIA_RGB'    'M_CCIR'    'M_CCIR_VIA_RGB'

  Columns 5 through 8

'M_NTSC'    'M_NTSC_RGB'      'M_NTSC_YC'    'M_PAL'

  Columns 9 through 10

'M_PAL_RGB'    'M_PAL_YC'
```

For Matrox devices, the toolbox uses the RS170 format as the default. (To find out which is the default video format, look in the DefaultFormat field of the device information structure returned by the imaqhwinfo function.)

Note For Matrox devices, the M_NTSC_RGB format string represents a component video format.

This example creates a video input object, specifying a color video format.

```
vid2 = videoinput('matrox', 1,'M_NTSC_RGB');
```

3-13

Using Device Configuration Files (Camera Files)

For some devices, you can use a device configuration file, also known as a camera file, to specify the video format as well as other configuration settings. Image acquisition device vendors supply these device configuration files.

Note The toolbox ignores hardware trigger configurations included in a device configuration file. To configure a hardware trigger, you must use the toolbox `triggerconfig` function. See "Example: Using a Hardware Trigger" on page 4-12 for more information.

For example, with Matrox frame grabbers, you can download digitizer configuration format (DCF) files, in their terminology. These files configure their devices to support particular cameras.

Some image acquisition device vendors provide utility programs you can use to create a device configuration file or edit an existing one. See your hardware vendor's documentation for more information.

To determine if your image acquisition device supports device configuration files, check the value of the `DeviceFileSupported` field of the device information structure returned by `imaqhwinfo`. See "Getting More Information About a Particular Device" on page 3-5 for more information.

This example creates a video input object specifying a Matrox device configuration file as an argument.

```
vid = videoinput('matrox',1,'pulnix.dcf')

Summary of Video Input Object Using 'Orion'.

   Acquisition Source(s): CH0 and CH1 are available.

 Acquisition Parameters: 'CH0' is the current selected source.
                         10 frames per trigger using the selected source.
                         'C:\pulnix.dcf' video data to be logged upon START.
                         Grabbing first of every 1 frame(s).
                         Log data to 'memory' on trigger.

     Trigger Parameters: 1 'immediate' trigger(s) on START.

                 Status: Waiting for START.
                         0 frames acquired since starting.
                         0 frames available for GETDATA.
```

When you use a device configuration file, the value of the VideoFormat property of the video input object is the full path of the file, not a video format string.

Specifying the Selected Video Source Object

When you create a video input object, the toolbox creates one or more video source objects associated with the video input object. The number of video source objects created depends on the device and the video format. The Source property of the video input object lists these video source objects.

To illustrate, this example lists the video source objects associated with the video input object vid.

```
   get(vid,'Source')

       Display Summary for Video Source Object Array:

       Index:    SourceName:    Selected:
       1         'CH0'          'on'
       2         'CH1'          'off'
       3         'CH2'          'off'
       4         'CH3'          'off'
```

```
5          'CH4'          'off'
6          'CH5'          'off'
7          'CH6'          'off'
8          'CH7'          'off'
```

By default, the video input object makes the first video source object in the array the selected source. To use another video source, change the value of the SelectedSourceName property.

This example changes the currently selected video source object from CH0 to CH1 by setting the value of the SelectedSourceName property.

```
vid.SelectedSourceName = 'CH1';
```

Note The getselectedsource function returns the video source object that is currently selected at the time the function is called. If you change the value of the SelectedSourceName property, you must call the getselectedsource function again to retrieve the new selected video source object.

Configuring Image Acquisition Object Properties

The video input object and the video source object both support properties that enable you to control characteristics of the video image and how it is acquired.

The video input object properties control aspects of an acquisition that are common to all image acquisition devices. For example, you can use the `FramesPerTrigger` property to specify the amount of data you want to acquire.

The video source object properties control aspects of the acquisition associated with a particular source. The set of properties supported by a video source object varies with each device. For example, some image acquisition devices support properties that enable you to control the quality of the image being produced, such as `Brightness`, `Hue`, and `Saturation`.

With either type of object, you can use the same toolbox functions to

- View a list of all the properties supported by the object, with their current values
- View the value of a particular property
- Get information about a property
- Set the value of a property

Note Three video input object trigger properties require the use of a special configuration function. For more information, see "Setting Trigger Properties" on page 3-23.

Viewing the Values of Object Properties

To view all the properties of an image acquisition object, with their current values, use the `get` function. You can also use the `inspect` function to view a list of object properties in the **Property Inspector** window, where you can also edit their values.

This example uses the `get` function to display a list of all the properties of the video input object `vid`. "Viewing the Properties of a Video Source Object" on page 3-19 describes how to do this for video source objects.

If you do not specify a return value, the get function displays the object properties in four categories: General Settings, Callback Function Settings, Trigger Settings, and Acquisition Sources.

```
get(vid)
  General Settings:
    DeviceID = 1
    DiskLogger = []
    DiskLoggerFrameCount = 0
    EventLog = [1x0 struct]
    FrameGrabInterval = 1
    FramesAcquired = 0
    FramesAvailable = 0
    FramesPerTrigger = 10
    Logging = off
    LoggingMode = memory
    Name = M_RS170-matrox-1
    NumberOfBands = 1
    Previewing = off
    ReturnedColorSpace = grayscale
    ROIPosition = [0 0 640 480]
    Running = off
    Tag =
    Timeout = 10
    Type = videoinput
    UserData = []
    VideoFormat = M_RS170
    VideoResolution = [640 480]

  Callback Function Settings:
    ErrorFcn = @imaqcallback
    FramesAcquiredFcn = []
    FramesAcquiredFcnCount = 0
    StartFcn = []
    StopFcn = []
    TimerFcn = []
    TimerPeriod = 1
    TriggerFcn = []
```

```
Trigger Settings:
  InitialTriggerTime = [0 0 0 0 0 0]
  TriggerCondition = none
  TriggerFrameDelay = 0
  TriggerRepeat = 0
  TriggersExecuted = 0
  TriggerSource = none
  TriggerType = immediate

Acquisition Sources:
  SelectedSourceName = CH0
  Source = [1x8 videosource]
```

Viewing the Properties of a Video Source Object

To view the properties supported by the video source object (or objects) associated with a video input object, use the getselectedsource function to retrieve the currently selected video source object. This example lists the properties supported by the video source object associated with the video input object vid. Note the device-specific properties that are included.

Note The video source object for your device might not include device-specific properties. For example, devices accessed with the 'winvideo' adaptor, such as Webcams, that use a Video for Windows (VFW) driver, may not provide a way for the toolbox to programmatically query for device properties. Use the configuration tools provided by the manufacturer to configure these devices.

```
get(getselectedsource(vid))
  General Settings:
    Parent = [1x1 videoinput]
    Selected = on
    SourceName = CH0
    Tag =
    Type = videosource

  Device Specific Properties:
    InputFilter = lowpass
    UserOutputBit3 = off
    UserOutputBit4 = off
```

```
            XScaleFactor = 1
            YScaleFactor = 1
```

Viewing the Value of a Particular Property

To view the value of a particular property of an image acquisition object, use the get function, specifying the name of the property as an argument. You can also access the value of the property as you would a field in a MATLAB structure.

This example uses the get function to retrieve the value of the Previewing property.

```
get(vid,'Previewing')

ans =

off
```

This example illustrates how to access the same property by referencing the object as if it were a MATLAB structure.

```
vid.Previewing

ans =

off
```

Getting Information About Object Properties

To get information about a particular property, you can view the reference page for the property in the "Property Reference" section of this documentation. You can also get information about a particular property at the command line by using the propinfo or imaqhelp functions.

The propinfo function returns a structure that contains information about the property such as its data type, default value, and a list of all possible values, if the property supports such a list. This example uses propinfo to get information about the LoggingMode property.

```
propinfo(vid,'LoggingMode')

ans =

                Type: 'string'
          Constraint: 'enum'
     ConstraintValue: {'memory'   'disk'   'disk&memory'}
        DefaultValue: 'memory'
            ReadOnly: 'whileRunning'
       DeviceSpecific: O
```

The imaqhelp function returns reference information about the property with a complete description. This example uses imaqhelp to get information about the LoggingMode property.

```
imaqhelp(vid,'LoggingMode')
```

Setting the Value of an Object Property

To set the value of a particular property of an image acquisition object, use the set function, specifying the name of the property as an argument. You can also assign the value to the property as you would a field in a MATLAB structure.

Note Because some properties are read only, only a subset of all video input and video source properties can be set.

This example uses the set function to set the value of the LoggingMode property.

```
set(vid,'LoggingMode','disk&memory')
```

To verify the new value of the property, use the get function.

```
get(vid,'LoggingMode')

ans =

disk&memory
```

This example sets the value of a property by assigning the value to the object as if it were a MATLAB structure.

```
vid.LoggingMode = 'disk';

vid.LoggingMode

ans =

disk
```

Viewing a List of All Settable Object Properties

To view a list of all the properties of a video input object or video source object that can be set, use the set function.

```
set(vid)

  General Settings:
   DiskLogger
   FrameGrabInterval
   FramesPerTrigger
   LoggingMode: [ {memory} | disk | disk&memory ]
   Name
   ROIPosition
   Tag
   Timeout
   UserData

  Callback Function Settings:
    ErrorFcn: string -or- function handle -or- cell array
    FramesAcquiredFcn: string -or- function handle -or- cell array
    FramesAcquiredFcnCount
    StartFcn: string -or- function handle -or- cell array
    StopFcn: string -or- function handle -or- cell array
    TimerFcn: string -or- function handle -or- cell array
    TimerPeriod
    TriggerFcn: string -or- function handle -or- cell array

  Trigger Settings:
    TriggerFrameDelay
    TriggerRepeat

  Acquisition Sources:
    SelectedSourceName: [ {CH0} | CH1 | CH2 | CH3 | CH4 ]
```

Setting Trigger Properties

The values of certain trigger properties, `TriggerType`, `TriggerCondition`, and `TriggerSource`, are interrelated. For example, some `TriggerCondition` values are only valid with specific values of the `TriggerType` property.

To ensure that you specify only valid combinations for the values of these properties, you must use two functions:

- The `triggerinfo` function returns all the valid combinations of values for the specified video input object.
- The `triggerconfig` function sets the values of these properties.

For more information, see "Specifying Trigger Type, Source, and Condition" on page 4-5.

Starting and Stopping a Video Input Object

When you create a video input object, you establish a connection between MATLAB and an image acquisition device. However, before you can acquire data from the device, you must start the object, using the start function.

```
start(vid);
```

When you start an object, you reserve the device for your exclusive use and lock the configuration. Thus, certain properties become read only while running.

An image acquisition object stops running when any of the following conditions is met:

• The requested number of frames is acquired. This occurs when

```
FramesAcquired = FramesPerTrigger * (TriggerRepeat + 1)
```

where FramesAcquired, FramesPerTrigger, and TriggerRepeat are properties of the video input object. For information about these properties, see Chapter 4, "Acquiring Image Data."

• A run-time error occurs.
• The object's Timeout value is reached.
• You issue the stop function.

When an object is started, the toolbox sets the object's Running property to 'on'. When an object is not running, the toolbox sets the object's Running property to 'off'; this state is called stopped. The following figure illustrates how an object moves from a running to a stopped state.
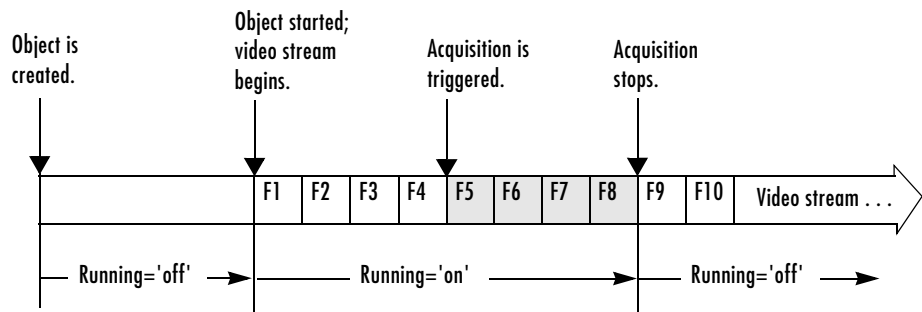
Transitions from Running to Stopped States

The following example illustrates starting and stopping an object:

1 Create an image acquisition object — This example creates a video input object for a Webcam image acquisition device. To run this example on your system, use the `imaqhwinfo` function to get the object constructor for your image acquisition device and substitute that syntax for the following code.

```
vid = videoinput('winvideo',1);
```

2 Configure properties — To illustrate object states, set the video input object's `TriggerType` property to `'Manual'`. To set the value of certain trigger properties, including the `TriggerType` property, you must use the `triggerconfig` function. See "Setting the Values of Trigger Properties" on page 4-5 for more information.

```
triggerconfig(vid, 'Manual')
```

Verify that the image is in a stopped state.

```
vid.Running
ans =
off
```

3 Start the image acquisition object — Call the `start` function to start the image acquisition object.

```
start(vid);
```

The video object is now running but not logging. With manual triggers, the video stream begins when the object starts but no frames are acquired until the trigger executes.

```
vid.Running
ans =
on
```

4 Execute the manual trigger — Call the `trigger` function to execute the manual trigger.

```
trigger(vid)
```

After it acquires the specified number of frames, the video input object stops running.

```
vid.Running
ans =
off
```

5 **Clean up** — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.

```
delete(vid)
clear vid
```

Deleting Image Acquisition Objects

When you finish using your image acquisition objects, use the `delete` function to remove them from memory. After deleting them, clear the variables that reference the objects from the MATLAB workspace by using the `clear` function.

Note When you delete a video input object, all the video source objects associated with the video input object are also deleted.

To illustrate, this example creates several video input objects and then deletes them.

1 **Create several image acquisition objects** — This example creates several video input objects for a single Webcam image acquisition device, specifying several different video formats. To run this example on your system, use the `imaqhwinfo` function to get the object constructor for your image acquisition device and substitute that syntax for the following code.

```
vid = videoinput('winvideo',1);
vid2 = videoinput('winvideo',1,'RGB24_176x144');
vid3 = videoinput('winvideo',1,'YV12_352x288');
```

2 **Clean up** — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.

You can delete image acquisition objects one at a time, using the delete function.

```
delete(vid)
```

You can also delete all the video input objects that currently exist in memory in one call to `delete` by using the `imaqfind` function. The `imaqfind` function returns an array of all the video input objects in memory.

```
imaqfind

   Video Input Object Array:

   Index:   Type:         Name:
```

```
      1          videoinput      RGB555_128x96-winvideo-1
      2          videoinput      RGB24_176x144-winvideo-1
      3          videoinput      YV12_352x288-winvideo-1
```

Nest a call to the `imaqfind` function within the `delete` function to delete all these objects from memory.

```
delete(imaqfind)
```

Note that the variables associated with the objects remain in the workspace.

```
whos
  Name       Size                     Bytes  Class

  vid        1x1                       1120  videoinput object
  vid2       1x1                       1120  videoinput object
  vid3       1x1                       1120  videoinput object
  vids       1x3                       1280  videoinput object
```

These variables are not valid image acquisition objects.

```
isvalid(vid)

ans =
    0
```

To remove these variables from the workspace, use the `clear` command.

Saving and Loading Image Acquisition Objects

You can save a video input object to a MAT-file just as you would any workspace variable by using the `save` command. This example saves the video input object `vid` to the MAT-file `myvid.mat`.

```
save myvid vid
```

When you save a video input object, all the video source objects associated with the video input object are also saved.

To load an image acquisition object that was saved to a MAT-file into the MATLAB workspace, use the `load` command. For example, to load `vid` from MAT-file `myvid.mat`, use

```
load myvid
```

Note The values of read-only properties are not saved. When you load an image acquisition object into the MATLAB workspace, read-only properties revert back to their default values. To determine if a property is read only, use the `propinfo` function or read the property reference page.

4

Acquiring Image Data

The core of any image acquisition application is the data acquired from the input device. A *trigger* is the event that initiates the acquisition of image frames, a process called *logging*. A trigger event occurs when a certain condition is met. For some types of triggers, the condition can be the execution of a toolbox function. For other types of triggers, the condition can be a signal from an external source that is monitored by the image acquisition hardware.

This chapter describes how to configure and use the various triggering options supported by the Image Acquisition Toolbox and control other acquisition parameters.

Overview

When a trigger occurs, the toolbox sets the object's Logging property to 'on' and starts storing the acquired frames in a buffer in memory, a disk file, or both. When the acquisition stops, the toolbox sets the object's Logging property to 'off'. The following figure illustrates when an object moves into a logging state and the relation between running and logging states.
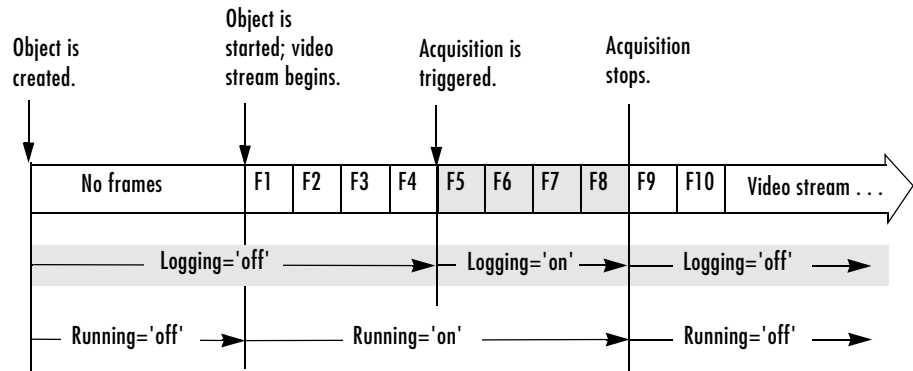
Logging State Transitions

Note After Logging is set to 'off', it is possible that the object might still be logging data to disk. To determine when disk logging is complete, check the value of the DiskLoggerFrameCount property. For more information, see "Logging Image Data to Disk" on page 4-32.

The following figure illustrates a group of frames being acquired from the video stream, logged to memory and disk, and brought into the MATLAB workspace as a multidimensional numeric array. Note that when frames are brought into the MATLAB workspace, they are removed from the memory buffer.
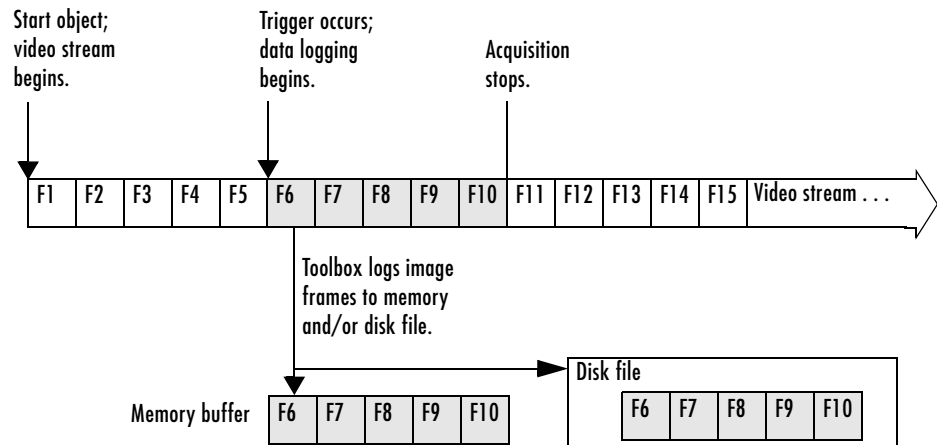
Overview of Data Logging

Trigger Properties

The video input object supports several properties that you can use to configure aspects of trigger execution. Some of these properties return information about triggers. For example, to find out when the first trigger occurred, look at the value of the `InitialTriggerTime` property. Other properties enable you to control trigger behavior. For example, you use the `TriggerRepeat` property to specify how many additional times an object should execute a trigger.

The following table provides a brief description of all the trigger-related properties supported by the video input object. For information about how to set these properties, see "Setting the Values of Trigger Properties" on page 4-5.

Trigger Properties of the Video Input Object

Property	Description
InitialTriggerTime	Reports the absolute time when the first trigger executed
TriggerCondition	Specifies the condition that must be met for a trigger to be executed. This property is always set to `'none'` for immediate and manual triggers.
TriggerFcn	Specifies the callback function to execute when a trigger occurs. For more information about callbacks, see Chapter 6, "Using Events and Callbacks."
TriggerFrameDelay	Specifies the number of frames to skip before logging data to memory, disk, or both. For more information, see "Delaying Data Logging After a Trigger" on page 4-24.
TriggerRepeat	Specifies the number of additional times to execute a trigger. If the value of `TriggerRepeat` is 0 (zero), the trigger executes but is not repeated any additional times. For more information, see "Specifying Multiple Triggers" on page 4-25.
TriggersExecuted	Reports the number of triggers that have been executed
TriggerSource	Specifies the source to monitor for a trigger condition to be met. This property is always set to `'none'` for immediate and manual triggers.
TriggerType	Specifies the type of trigger: `'immediate'`, `'manual'`, or `'hardware'`. Use the `triggerinfo` function to determine whether your image acquisition device supports hardware triggers.

Setting the Values of Trigger Properties

Most trigger properties can be set using the same methods you use to set any other image acquisition object property: using the set function or referencing the property as you would a field in a structure. For example, you can use the set function to specify the value of the TriggerRepeat property, where vid is a video input object created using the videoinput function.

```
set(vid,'TriggerRepeat',Inf)
```

For more information, see "Configuring Image Acquisition Object Properties" on page 3-17.

Some trigger properties, however, are interrelated and require the use of the triggerconfig function to set their values. These properties are the TriggerType, TriggerCondition, and TriggerSource properties. For example, some TriggerCondition values are only valid when the value of the TriggerType property is 'hardware'.

Specifying Trigger Type, Source, and Condition

Setting the values of the TriggerType, TriggerSource, and TriggerCondition properties can be a two-step process:

1 Determine valid configurations of these properties by calling the triggerinfo function.

2 Set the values of these properties by calling the triggerconfig function.

For an example of using these functions, see "Example: Using a Hardware Trigger" on page 4-12.

Determining Valid Configurations

To find all the valid configurations of the TriggerType, TriggerSource, and TriggerCondition properties, use the triggerinfo function, specifying a video input object as an argument.

```
config = triggerinfo(vid);
```

This function returns an array of structures, one structure for each valid combination of property values. Each structure in the array is made up of three

fields that contain the values of each of these trigger properties. For example, the structure returned for an immediate trigger always has these values:

```
      TriggerType: 'immediate'
 TriggerCondition: 'none'
    TriggerSource: 'none'
```

A device that supports hardware configurations might return the following structure.

```
      TriggerType: 'hardware'
 TriggerCondition: 'risingEdge'
    TriggerSource: 'TTL'
```

Note The text strings used as the values of the `TriggerCondition` and `TriggerSource` properties are device specific. Your device, if it supports hardware triggers, might support different condition and source values.

Configuring Trigger Type, Source, and Condition Properties

To set the values of the `TriggerType`, `TriggerSource`, and `TriggerCondition` properties, you must use the `triggerconfig` function. You specify the value of the property as an argument to the function.

For example, this code sets the values of these properties for a hardware trigger.

```
triggerconfig(vid,'hardware','risingEdge','TTL')
```

If you are specifying a manual trigger, you only need to specify the trigger type value as an argument.

```
triggerconfig(vid,'manual')
```

You can also pass one of the structures returned by the `triggerinfo` function to the `triggerconfig` function and set all three properties at once.

```
triggerconfig(config(1))
```

See the `triggerconfig` function documentation for more information.

Specifying the Trigger Type

To specify the type of trigger you want to execute, set the value of the TriggerType property of the video input object. You must use the triggerconfig function to set the value of this property. The following table lists all the trigger types supported by the toolbox, with information about when to use each type of trigger.

Comparison of Trigger Types

TriggerType Value	TriggerSource and TriggerCondition Values	Description
'immediate'	Always 'none'	The trigger occurs automatically, immediately after the start function is issued. This is the default trigger type. For more information, see "Example: Using an Immediate Trigger" on page 4-8.
'manual'	Always 'none'	The trigger occurs when you issue the trigger function. A manual trigger can provide more control over image acquisition. For example, you can monitor the video stream being acquired, using the preview function, and manually execute the trigger when you observe a particular condition in the scene. For more information, see "Example: Using a Manual Trigger" on page 4-10.
'hardware'	Device-specific	Hardware triggers are external signals that are processed directly by the hardware. This type of trigger is used when synchronization with another device is part of the image acquisition setup or when speed is required. A hardware device can process an input signal much faster than software. For more information, see "Example: Using a Hardware Trigger" on page 4-12. **Note**: Only a subset of image acquisition devices supports hardware triggers. To determine the trigger types supported by your device, see "Determining Valid Configurations" on page 4-5.

Example: Using an Immediate Trigger

To use an immediate trigger, simply create a video input object. Immediate triggering is the default trigger type for all video input objects. With an immediate trigger, the object executes the trigger immediately after you start the object running with the start command. The following figure illustrates an immediate trigger.
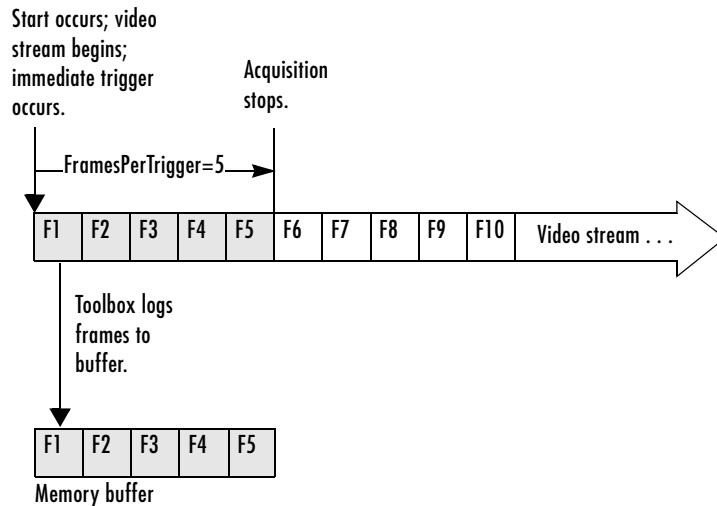
Immediate Trigger

The following example illustrates how to use an immediate trigger:

1 Create an image acquisition object — This example creates a video input object for a Matrox image acquisition device. To run this example on your system, use the imaqhwinfo function to get the object constructor for your image acquisition device and substitute that syntax for the following code.

```
vid = videoinput('matrox',1);
```

Verify that the object has not acquired any frames.

```
get(vid,'FramesAcquired')
ans =

    0
```

2 Configure properties — To use an immediate trigger, you do not have to configure the `TriggerType` property because `'immediate'` is the default trigger type. You can verify this by using the `triggerconfig` function to view the current trigger configuration or by viewing the video input object's properties.

```
triggerconfig(vid)
ans =

        TriggerType: 'immediate'
   TriggerCondition: 'none'
      TriggerSource: 'none'
```

This example sets the value of the `FramesPerTrigger` property to 5. (The default is 10 frames per trigger.)

```
set(vid,'FramesPerTrigger',5)
```

3 Start the image acquisition object — Call the `start` function to start the image acquisition object. By default, the object executes an immediate trigger and acquires five frames of data, logging the data to a memory buffer. After logging the specified number of frames, the object stops running.

```
start(vid)
```

To verify that the object acquired data, view the value of the `FramesAcquired` property. The object updates the value of this property as it acquires data.

```
vid.FramesAcquired
ans =

   5
```

To execute another immediate trigger, you must restart the object. Note, however, that this deletes the data acquired by the first trigger. To execute multiple immediate triggers, specify a value for the `TriggerRepeat` property. See "Specifying Multiple Triggers" on page 4-25 for more information.

4 **Clean up** — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.

```
delete(vid)
clear vid
```

Example: Using a Manual Trigger

To use a manual trigger, create a video input object and set the value of the TriggerType property to 'manual'. A video input object executes a manual trigger after you issue the trigger function. The following figure illustrates a manual trigger.
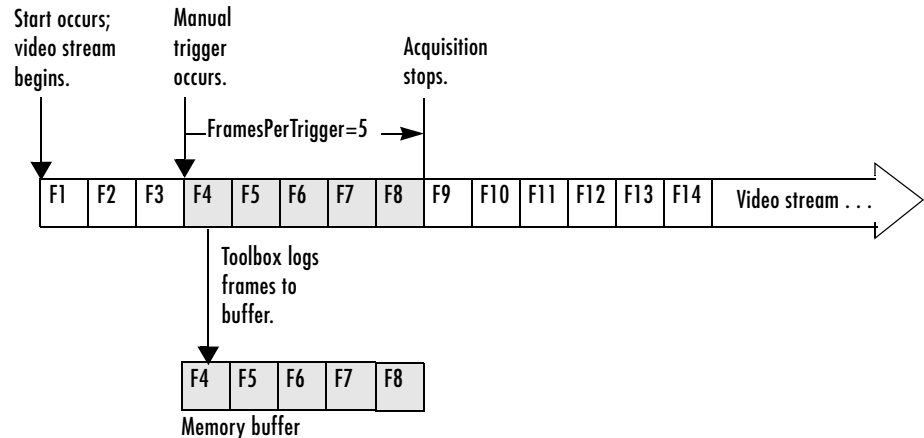
Manual Trigger

The following example illustrates how to use a manual trigger:

1 **Create an image acquisition object** — This example creates a video input object for a Webcam image acquisition device. To run this example on your system, use the imaqhwinfo function to get the object constructor for your image acquisition device and substitute that syntax for the following code.

```
vid = videoinput('winvideo',1);
```

Verify that the object has not acquired any frames.

```
get(vid,'FramesAcquired')
ans =
    0
```

2 Configure properties — Set the video input object's `TriggerType` property to `'Manual'`. To set the values of certain trigger properties, including the `TriggerType` property, you must use the `triggerconfig` function. See "Setting the Values of Trigger Properties" on page 4-5 for more information.

```
triggerconfig(vid, 'Manual')
```

This example also sets the value of the `FramesPerTrigger` property to 5. (The default is 10 frames per trigger.)

```
set(vid,'FramesPerTrigger',5)
```

3 Start the image acquisition object — Call the `start` function to start the image acquisition object.

```
start(vid);
```

The video object is now running but not logging. With manual triggers, the video stream begins when the object starts but no frames are acquired until the trigger executes.

```
vid.Running
ans =
on

vid.Logging
ans =
off
```

Verify that the object has still not acquired any frames.

```
get(vid,'FramesAcquired')
ans =
    0
```

4 Execute the manual trigger — Call the `trigger` function to execute the manual trigger.

```
trigger(vid)
```

The object initiates the acquisition of five frames. Check the
`FramesAcquired` property again to verify that five frames have been
acquired.

```
get(vid,'FramesAcquired')
ans =
    5
```

After it acquires the specified number of frames, the video input object stops
running. To execute another manual trigger, you must first restart the video
input object. Note that this deletes the frames acquired by the first trigger.
To execute multiple manual triggers, specify a value for the `TriggerRepeat`
property. See "Specifying Multiple Triggers" on page 4-25 for more
information.

5 Clean up — Always remove image acquisition objects from memory, and the
variables that reference them, when you no longer need them.

```
delete(vid)
clear vid
```

Example: Using a Hardware Trigger

To use a hardware trigger, create a video input object and set the value of the
`TriggerType` property to `'hardware'`. You must also specify the source of the
hardware trigger and the condition type. The hardware monitors the source
you specify for the condition you specify. The following figure illustrates a
hardware trigger. For hardware triggers, the video stream does not start until
the trigger occurs.

Note Trigger sources and the conditions that control hardware triggers are
device specific. Use the `triggerinfo` function to determine whether your
image acquisition device supports hardware triggers and, if it does, which
conditions you can configure. Refer to the documentation that came with your
device for more detailed information about its hardware triggering
capabilities.

Start object;
hardware
monitors trigger
source.

Trigger
condition met;
trigger occurs.

Acquisition
stops.

├─── FramesPerTrigger=5 ───→

| No frames | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | Video stream . . . |

Toolbox logs
frames to
buffer.

| F1 | F2 | F3 | F4 | F5 |

Memory buffer

Hardware Trigger

The following example illustrates how to use a hardware trigger:

1 **Create an image acquisition object** — This example creates a video input object for a Matrox image acquisition device. To run this example on your system, use the `imaqhwinfo` function to get the object constructor for your image acquisition device and substitute that syntax for the following code. The device must support hardware triggers.

```
vid = videoinput('matrox',1);
```

2 **Determine valid trigger property configurations** — Use the `triggerinfo` function to determine if your image acquisition device supports hardware triggers, and if it does, to find out valid configurations of the `TriggerSource` and `TriggerCondition` properties. See "Determining Valid Configurations" on page 4-5 for more information.

In this example, `triggerinfo` returns the following valid trigger configurations.

```
triggerinfo(vid)
Valid Trigger Configurations:

      TriggerType:    TriggerCondition:    TriggerSource:
      'immediate'     'none'               'none'
      'manual'        'none'               'none'
      'hardware'      'risingEdge'         'TTL'
      'hardware'      'fallingEdge'        'TTL'
```

3 **Configure properties** — Configure the video input object trigger properties to one of the valid combinations returned by `triggerinfo`. You can specify each property value as an argument to the `triggerconfig` function

```
triggerconfig(vid, 'hardware','risingEdge','TTL')
```

Alternatively, you can set these values by passing one of the structures returned by the `triggerinfo` function to the `triggerconfig` function.

```
configs = triggerinfo(vid);
triggerconfig(vid,configs(3));
```

This example also sets the value of the `FramesPerTrigger` property to 5. (The default is 10 frames per trigger.)

```
set(vid,'FramesPerTrigger',5)
```

4 **Start the image acquisition object** — Call the `start` function to start the image acquisition object.

```
start(vid)
```

The object is running but not logging any data. The hardware begins monitoring the trigger source for the specified condition. When the condition is met, the hardware executes a trigger and begins providing image frames to the object. The object acquires the number of frames specified by the `FramesPerTrigger` property. View the value of the `FramesAcquired` property to see how much data was acquired. The object updates the value of this property as it acquires data.

```
vid.FramesAcquired
ans =

    5
```

After it executes the trigger and acquires the specified number of frames, the video input object stops running. To execute another hardware trigger, you must first restart the video input object. Note that this deletes the frames acquired by the first trigger. To execute multiple triggers, specify a value for the `TriggerRepeat` property. See "Specifying Multiple Triggers" on page 4-25 for more information.
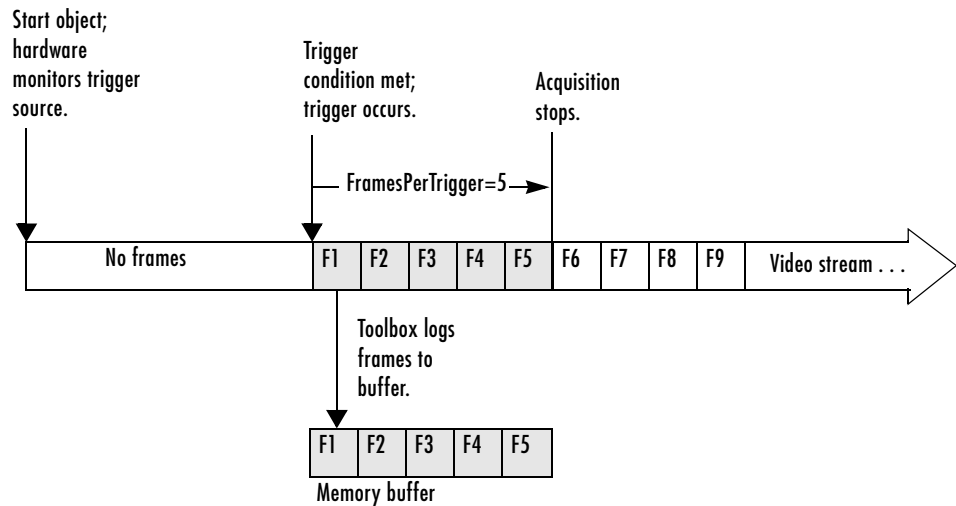
5 **Clean up** — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.

```
delete(vid)
clear vid
```

Controlling Logging Parameters

This section describes how to control various aspects of data logging.

- Specifying the logging mode
- Specifying the number of frames to log
- Determining how many frames have been logged since the object was started
- Determining how many frames are currently available in the memory buffer
- Delaying data logging after a trigger executes
- Specifying multiple trigger executions

Specifying Logging Mode

Using the video input object `LoggingMode` property, you can control where the toolbox logs acquired frames of data.

The default value for the `LoggingMode` property is `'memory'`. In this mode, the toolbox logs data to a buffer in memory. If you want to bring image data into the MATLAB workspace, you must log frames to memory. The functions provided by the toolbox to move data into the workspace all work with the memory buffer. For more information, see "Bringing Image Data into the MATLAB Workspace" on page 5-3.

You can also log data to a disk file by setting the `LoggingMode` property to `'disk'` or to `'disk&memory'`. By logging frames to a disk file, you create a permanent record of the frames you acquire. For example, this code sets the value of the `LoggingMode` property of the video input object `vid` to `'disk&memory'`.

```
set(vid,'LoggingMode','disk&memory');
```

Because the toolbox stores the image frames in Audio-Video Interleaved (AVI) format, you can view the logged frames in any standard media player. For more information, see "Logging Image Data to Disk" on page 4-32.

Specifying the Number of Frames to Log

In the Image Acquisition Toolbox, you specify the amount of data you want to acquire as the number of frames per trigger.

You specify the desired size of your acquisition as the value of the video input object `FramesPerTrigger` property. By default, the value of this property is 10 frames per trigger, but you can specify any value. The following figure illustrates an acquisition using the default value for the `FramesPerTrigger` property. To see an example of an acquisition, see "Example: Acquiring 100 Frames" on page 4-19.
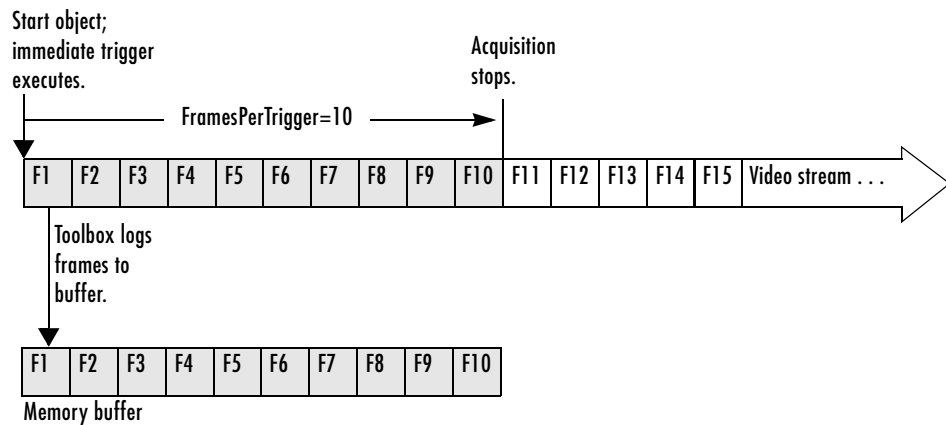
Specifying the Amount of Data to Log

Note While you can specify any size acquisition, the number of frames you can acquire is limited by the amount of memory you have available on your system for image storage. A large acquisition can potentially fill all available system memory. For large acquisitions, you might want to remove frames from the buffer as they are logged. For more information, see "Moving Multiple Frames into the Workspace" on page 5-3. To learn how to empty the memory buffer, see "Freeing Memory" on page 4-29.

Specifying a Noncontiguous Acquisition

Although `FramesPerTrigger` specifies the number of frames to acquire, these frames do not have to be captured contiguously from the video stream. You can specify that the toolbox skip a certain number of frames between frames it acquires. To do this, set the value of the `FrameGrabInterval` property.

Note The `FrameGrabInterval` property controls the interval at which the toolbox acquires frames from the video stream (measured in frames). This property does not control the rate at which frames are provided by the device, otherwise known as the frame rate.

The following figure illustrates how the `FrameGrabInterval` property affects an acquisition.
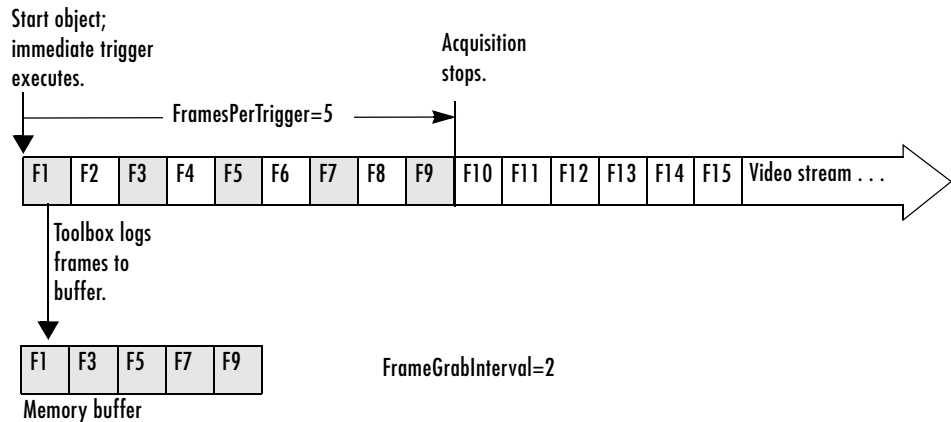
Impact of FrameGrabInterval on Data Logging

Determining How Much Data Has Been Logged

To determine how many frames have been acquired by a video input object, check the value of the FramesAcquired property. This property tells how many frames the object has acquired since it was started. To determine how many frames are currently available in the memory buffer, see "Determining How Many Frames Are Available" on page 4-20.

Example: Acquiring 100 Frames

This example illustrates how you can specify the amount of data to be acquired and determine how much data has been acquired. (For an example of configuring a time-based acquisition, see "Example: Acquiring 10 Seconds of Image Data" on page 5-5.)

1 **Create an image acquisition object** — This example creates a video input object for a Windows image acquisition device. To run this example on your system, use the imaqhwinfo function to get the object constructor for your image acquisition device and substitute that syntax for the following code.

```
vid = videoinput('winvideo',1);
```

2 **Configure properties** — Specify the amount of data you want to acquire as the number of frames per trigger. By default, a video input object acquires 10 frames per trigger. For this example, set the value of this property to 100.

```
set(vid,'FramesPerTrigger',100)
```

3 **Start the image acquisition object** — Call the start function to start the image acquisition object.

```
start(vid)
```

The object executes an immediate trigger and begins acquiring frames of data. The start function returns control to the command line immediately but the object continues logging the data to the memory buffer. After logging the specified number of frames, the object stops running.

4 **Check how many frames have been acquired** — To verify that the specified number of frames has been acquired, check the value of the FramesAcquired property. Note that the object continuously updates the value of the FramesAcquired property as the acquisition progresses. If you

view the value of this property several times during an acquisition, you can see the number of frames acquired increase until logging stops.

```
vid.FramesAcquired
ans =

    100
```

5 Clean up — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.

```
delete(vid)
clear vid
```

Determining How Many Frames Are Available

The FramesAcquired property tells how many frames the object has logged since it was started, described in "Determining How Much Data Has Been Logged" on page 4-19. Once you move frames from the memory buffer into the MATLAB workspace, the number of frames stored in the memory buffer will differ from the FramesAcquired value. To determine how many frames are currently available in the memory buffer, check the value of the FramesAvailable property.

Note The FramesAvailable property tells the number of frames in the memory buffer, *not* in the disk log, if LoggingMode is configured to 'disk' or 'disk&memory'. Because it takes longer to write frames to a disk file than to memory, the number of frames stored in the disk log might lag behind those stored in the memory buffer. To see how many frames are available in the disk log, look at the value of the DiskLoggerFrameCount property. See "Logging Image Data to Disk" on page 4-32 for more information.

This example illustrates the distinction between the FramesAcquired and the FramesAvailable properties:

1 Create an image acquisition object — This example creates a video input object for a Windows image acquisition device. To run this example on your system, use the imaqhwinfo function to get the object constructor for your image acquisition device and substitute that syntax for the following code.

```
vid = videoinput('winvideo',1);
```

2 **Configure properties** — For this example, configure an acquisition of 15 frames.

```
set(vid,'FramesPerTrigger',15)
```

3 **Start the image acquisition object** — Call the start function to start the image acquisition object.

```
start(vid)
```

The object executes an immediate trigger and begins acquiring frames of data. The start function returns control to the command line immediately but the object continues logging the data to the memory buffer. After logging the specified number of frames, the object stops running.

4 **Check how many frames have been acquired** — To determine how many frames the object has acquired and how many frames are available in the memory buffer, check the value of the FramesAcquired and FramesAvailable properties.

```
vid.FramesAcquired
ans =

    15

vid.FramesAvailable

ans =

    15
```

The object updates the value of these properties continuously as it acquires frames of data. The following figure illustrates how the object puts acquired frames in the memory buffer as the acquisition progresses.
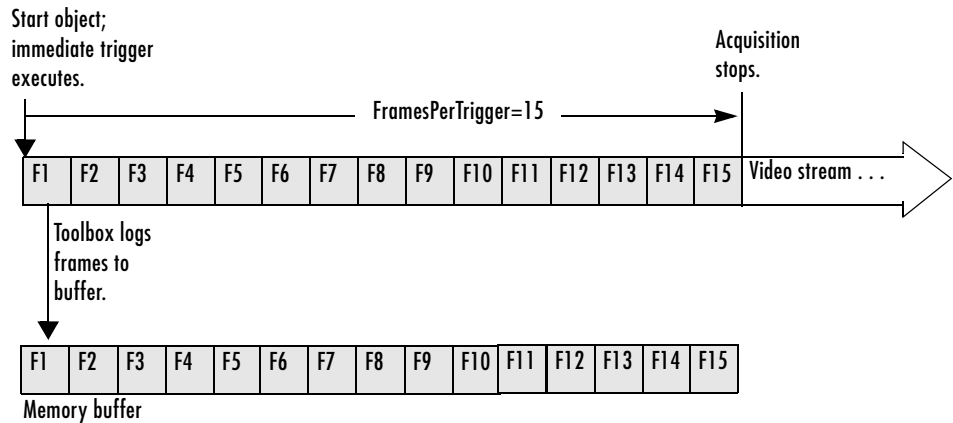
Frames Available After Initial Trigger Execution

5 **Remove frames from the memory buffer** — When you remove frames from the memory buffer, the object decrements the value of the FramesAvailable property by the number of frames removed.

To remove frames from the memory buffer, call the getdata function, specifying the number of frames to retrieve. For more information about using getdata, see "Bringing Image Data into the MATLAB Workspace" on page 5-3.

```
data = getdata(vid,5);
```

After you execute the getdata function, check the values of the FramesAcquired and FramesAvailable properties again. Notice that the FramesAcquired property remains unchanged but the object has decremented the value of the FramesAvailable property by the number of frames removed from the memory buffer.

```
vid.FramesAcquired

ans =

    15

vid.FramesAvailable
```

```
ans =

    10
```

The following figure illustrates the contents of the memory buffer after frames are removed.

T$_1$: After logging is complete

| F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | F10 | F11 | F12 | F13 | F14 | F15 |

Memory buffer

FramesAcquired=15
FramesAvailable=15

T$_2$: After getdata removes 5 frames

| F6 | F7 | F8 | F9 | F10 | F11 | F12 | F13 | F14 | F15 |

Memory buffer

FramesAcquired=15
FramesAvailable=10
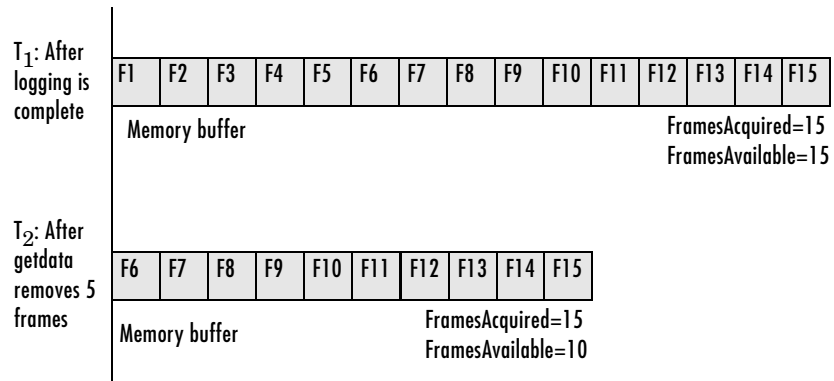
Contents of Memory Buffer Before and After Removing Frames

6 Clean up — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.

```
delete(vid)
clear vid
```

Delaying Data Logging After a Trigger

In some image acquisition setups, you might not want to log the first few frames returned from your camera or other imaging device. For example, some cameras require a short warmup time when activated. The quality of the first few images returned by these cameras might be too dark to be useful for your application.

To account for this characteristic of your setup, you can specify that the toolbox skip a specified number of frames after a trigger executes. You use the TriggerFrameDelay property to specify the number of frames you want to skip before logging begins.

For example, to specify a delay of five frames before data logging begins after a trigger executes, you would set the value of the TriggerFrameDelay property to 5. The number of frames captured is defined by the FramesPerTrigger property and is unaffected by the delay.

```
set(vid,'TriggerFrameDelay',5);
```

This figure illustrates this scenario.
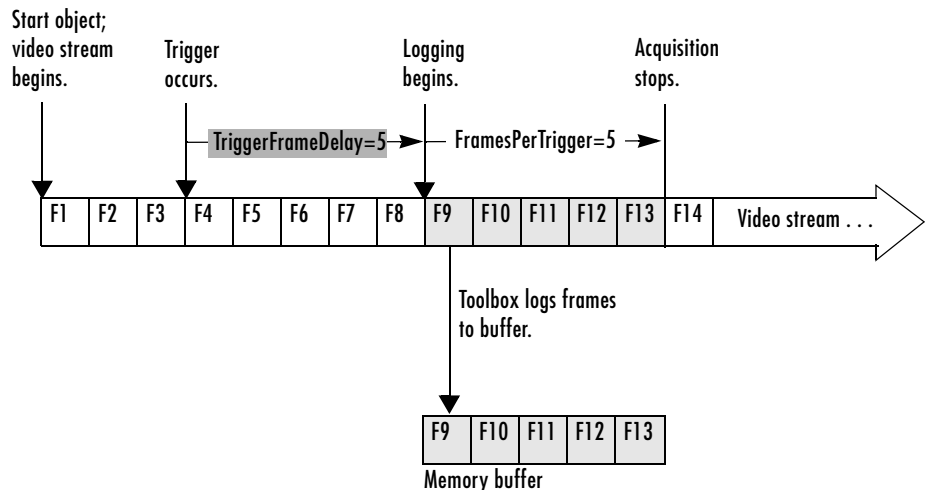
Specifying a Delay Before Data Logging Begins

Specifying Multiple Triggers

When a trigger occurs, a video input object acquires the number of frames specified by the FramesPerTrigger property and logs the data to a memory buffer, a disk file, or both.

When it acquires the specified number of frames, the video input object stops running. To execute another trigger, you must restart the video input object. Restarting an object causes it to delete all the data it has stored in the memory buffer from the previous trigger. To execute multiple triggers, retaining the data from each trigger, you must specify a value for the TriggerRepeat property.

Note that the TriggerRepeat property specifies the number of *additional* times a trigger executes. For example, to execute a trigger three times, you would set the value of the TriggerRepeat property to 2. In the following, vid is a video input object created with the videoinput function.

```
set(vid,'TriggerRepeat',2);
```

This figure illustrates an acquisition with three executions of a manual trigger. In the figure, the FramesPerTrigger property is set to 3.
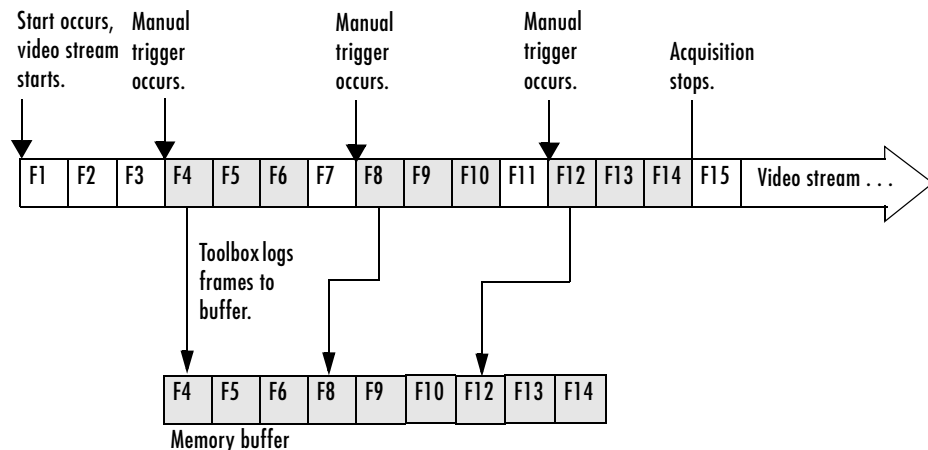
Executing Multiple Triggers

Waiting for an Acquisition to Finish

When you trigger an acquisition, control returns to the command line before data logging is complete. In some cases, you might want to delay further processing until the number of frames you requested has been logged. To block the command line until logging is finished, use the wait function.

Note The wait function can be very useful when you create a script or function in an M-file.

The following example illustrates the wait function.

1 **Create an image acquisition object** — This example creates a video input object for a Matrox image acquisition device. To run this example on your system, use the imaqhwinfo function to get the object constructor for your image acquisition device and substitute that syntax for the following code.

```
vid = videoinput('matrox',1);
```

2 **Configure properties** — To illustrate how wait blocks until all the specified frames become available, configure an acquisition large enough to produce a noticeable duration.

```
set(vid,'FramesPerTrigger',100)
```

3 **Start the image acquisition object** — Call the start function to start the image acquisition object.

```
start(vid)
```

The object executes an immediate trigger and begins acquiring frames of data. The start function returns control to the command line immediately but the object continues logging the data to the memory buffer.

4 **Block the command line until the acquisition finishes** — After the start function returns, call the wait function.

```
wait(vid)
```

The wait function returns control to the command line when the value of the video input object's Running property changes from 'on' to 'off'.

5 Clean up — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.

```
delete(vid)
clear vid
```

Managing Memory Usage

The first time it needs to allocate memory to store an image frame, the toolbox determines the total amount of memory it has available to store acquired image frames. The toolbox sets this value, called the *frame memory limit*, to be all the physical memory that is available when the toolbox is first accessed.

Image data can require a lot of memory. For example, even a relatively small (96-by-128) 24-bit color image requires almost 37 K bytes for each frame.

```
whos

  Name        Size                    Bytes  Class

  rgb_image   96x128x3                36864  uint8 array
```

This section describes how to

- Monitor the toolbox's memory usage
- Modify the toolbox's frame memory limit, if necessary
- Empty frames from the memory buffer

To see an example of monitoring memory usage in a callback function, see "Example: Monitoring Memory Usage" on page 6-16.

Monitoring Memory Usage

The toolbox includes a utility function, called imaqmem, that provides information about the toolbox's current memory usage.

The imaqmem function returns a structure that contains several memory usage statistics including the total amount of physical memory available, the amount of physical memory currently in use, and a value, called the memory load, that characterizes the current memory usage.

To illustrate, this example calls imaqmem and then uses the frame memory limit and the current frame memory usage statistics to calculate how much memory is left for image frame storage.

```
out = imaqmem;
mem_left = out.FrameMemoryLimit - out.FrameMemoryUsed;
```

Modifying the Frame Memory Limit

To enable your image acquisition application to work with more image frames, you might want to increase the frame memory limit. Using the `imaqmem` function you can determine the current frame memory limit and specify a new one.

To illustrate, this example calls `imaqmem` to get the current frame memory limit, which is approximately 28 MB, calls `imaqmem` to specify a new frame memory limit (36 MB), and then calls `imaqmem` again to verify the new setting.

```
out = imaqmem;

out.FrameMemoryLimit

ans =

    28389376

imaqmem(36000000)

out = imaqmem;

out.FrameMemoryLimit

ans =

    36000000
```

Freeing Memory

At times, while acquiring image data, you might want to delete some or all of the frames that are stored in memory. Using the `flushdata` function, you can delete all the frames currently stored in memory or only those frames associated with the execution of a trigger.

The following example illustrates how to use `flushdata` to delete all the frames in memory or one trigger's worth of frames.

1 **Create an image acquisition object** — This example creates a video input object for a Windows image acquisition device. To run this example on your

system, use the imaqhwinfo function to get the object constructor for your image acquisition device and substitute that syntax for the following code.

```
vid = videoinput('winvideo',1);
```

2 **Configure properties** — For this example, configure an acquisition of five frames per trigger and, to show the effect of flushdata, configure multiple triggers using the TriggerRepeat property.

```
vid.FramesPerTrigger = 5
vid.TriggerRepeat = 2;
```

3 **Start the image acquisition object** — Call the start function to start the image acquisition object.

```
start(vid)
```

The object executes an immediate trigger, acquires five frames of data, and repeats this trigger two more times. After logging the specified number of frames, the object stops running.

To verify that the object acquired data, view the value of the FramesAvailable property. This property reports how many frames are currently stored in the memory buffer.

```
vid.FramesAvailable
ans =

    15
```

4 **Delete a trigger's worth of image data** — Call the flushdata function, specifying the mode 'triggers'. This deletes the frames associated with the oldest trigger.

```
flushdata(vid,'triggers');
```

The following figure shows the frames acquired before and after the call to flushdata. Note how flushdata deletes the frames associated with the oldest trigger.
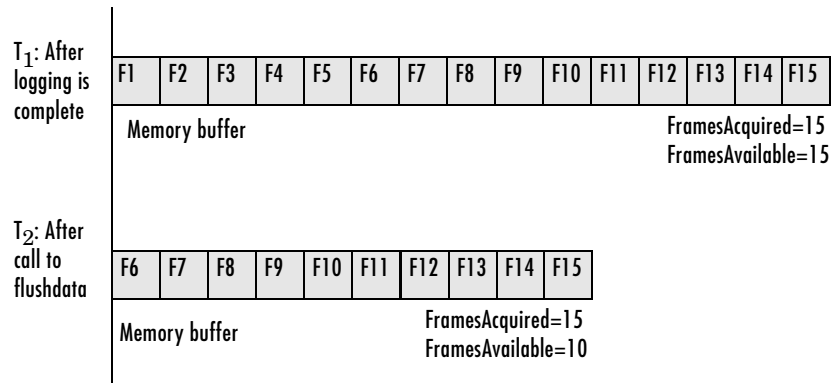
T_1: After logging is complete

F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | F10 | F11 | F12 | F13 | F14 | F15

Memory buffer

FramesAcquired=15
FramesAvailable=15

T_2: After call to flushdata

F6 | F7 | F8 | F9 | F10 | F11 | F12 | F13 | F14 | F15

Memory buffer

FramesAcquired=15
FramesAvailable=10

To verify that the object deleted the frames, view the value of the FramesAvailable property.

```
vid.FramesAvailable
ans =

    10
```

5 Empty the entire memory buffer — Calling flushdata without specifying the mode deletes all the frames stored in memory.

```
flushdata(vid);
```

To verify that the object deleted the frames, view the value of the FramesAvailable property.

```
vid.FramesAvailable
ans =

    0
```

6 Clean up — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.

```
delete(vid)
clear vid
```

Logging Image Data to Disk

While a video input object is running, you can log the image data being acquired to a disk file. Logging image data to disk provides a record of your data.

To set up data logging to disk, perform these steps:

1 Create a disk file to store the data. The toolbox logs the data to disk in Audio-Video Interleave (AVI) format because this format provides data compression capabilities that allow for efficient storage. You must use the MATLAB avifile function to create this log file. For more information, see "Creating an AVI File Object for Logging" on page 4-33.

2 Set the value of the video input object LoggingMode property to 'disk' or 'disk&memory'.

3 Set the value of the video input object DiskLogger property to the AVI file object created in step 1.

The following figure shows how the toolbox adds frames to the AVI file when a trigger occurs. With each subsequent trigger, the toolbox appends the acquired frames to the end of the AVI file. The frames must have the same dimensions. For an example of how to set up disk data logging, see "Example: Logging Data to Disk" on page 4-35.

Start occurs;
video stream
begins.

Trigger
occurs.

Acquisition
stops.

Trigger
occurs.

Acquisition
stops.

← FramesPerTrigger=5 →

← FramesPerTrigger=5 →

| F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F15 | F16 | F17 | F18 | F19 | F20 | F21 | Video stream . . . |

Toolbox logs
frames to disk
file.

Disk file

| F3 | F4 | F5 | F6 | F7 | F16 | F17 | F18 | F19 | F20 |
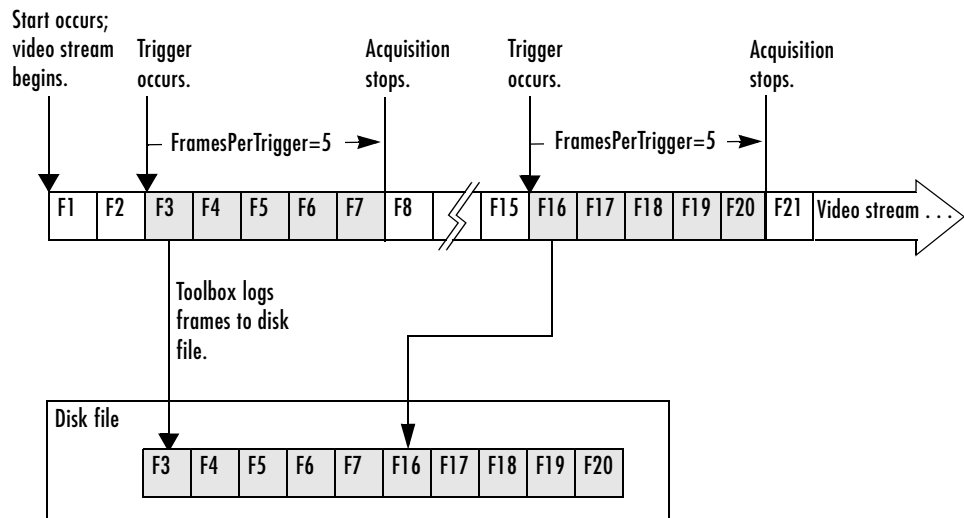
Logging Data to a Disk File

Creating an AVI File Object for Logging

To create an AVI file in the MATLAB environment, use the avifile function.
You specify the name of the AVI file to the avifile function. For example, to
create the AVI file named my_datalog.avi, enter this code at the MATLAB
command prompt.

```
aviobj = avifile('my_datalog.avi');
```

The avifile function returns an AVI file object. You can use the AVI file object
returned by the avifile function, aviobj, to modify characteristics of the AVI
file by setting the values of the object's properties. For example, you can specify
the codec used for data compression or specify the desired quality of the output.
For complete information about AVI file objects and their properties, see the
MATLAB avifile documentation.

Guidelines for Using an AVI File Object to Log Image Data

When you specify the AVI file object as the value of the DiskLogger property, you are creating a copy of the AVI file object. Do not access the AVI file object using the original variable name, aviobj, while the video input object is using the file for data logging. To avoid file access conflicts, keep in mind these guidelines when using an AVI file for data logging:

- Do not close an AVI file object while it is being used for data logging.
- Do not use the AVI file addframe function to add frames to the AVI file object while it is being used for data logging.
- Do not change the values of any AVI file object properties while it is being used for data logging.

Closing the DiskLogger AVI file

When data logging has ended, close the AVI file to make it accessible outside the MATLAB environment. Use the value of the video input object DiskLogger property to reference the AVI file, rather than the variable returned when you created the AVI file object (aviobj). See "Example: Logging Data to Disk" for an example.

Before you close the file, make sure that the video input object has finished logging frames to disk. Because logging to disk takes more time than logging to memory, the completion of disk logging can lag behind the completion of memory logging. To determine when logging to disk is complete, check the value of the DiskLoggerFrameCount property; this property tells how many frames have been logged to disk.

Note When you log frames to disk, the video input object queues the frames for writing but the operating system might not perform the write operation immediately. Closing an AVI file causes the data to be written to the disk.

Example: Logging Data to Disk

This example illustrates how to configure a video input object to log data to a disk file:

1 **Create a MATLAB AVI file object** – Create the MATLAB AVI file that you want to use for data logging, using the `avifile` function. You specify the name of the AVI file when you create it.

```
my_log = 'my_datalog.avi';
aviobj = avifile(my_log);

aviobj

Adjustable parameters:
                Fps: 15.0000
        Compression: 'Indeo3'
            Quality: 75
      KeyFramePerSec: 2.1429
           VideoName: 'my_datalog.avi'

Automatically updated parameters:
          Filename: 'my_datalog.avi'
        TotalFrames: 0
              Width: 0
             Height: 0
             Length: 0
          ImageType: 'Unknown'
       CurrentState: 'Open'
```

2 **Configure properties of the AVI file object** – You can optionally configure the properties of the AVI file object. The AVI file object supports properties that control the data compression used, image quality, and other characteristics of the file. The example sets the quality property to a midlevel value. By lowering the quality, the AVI file object creates smaller log files.

```
aviobj.Quality = 50;
```

4-35

3 Create a video input object — This example creates a video input object for a Matrox image acquisition device. To run this example on your system, use the imaqhwinfo function to get the video input object constructor for your image acquisition device and substitute that syntax for the following code.

```
vid = videoinput('matrox',1);
```

4 Configure video input object properties — Set up disk logging by setting the value of the DiskLogger property to be aviobj, the AVI file object created in step 1. Then, set the LoggingMode property to 'disk' (or 'disk&memory'). This example also sets the TriggerRepeat property.

```
vid.LoggingMode = 'disk&memory';
vid.DiskLogger = aviobj;
vid.TriggerRepeat = 3;
```

5 Start the video input object — Start logging data to disk.

```
start(vid)
```

The object executes an immediate trigger, acquires frames of data, repeats the trigger three additional times, and then stops.

To verify that all the frames have been logged to the AVI file, check the value of the DiskLoggerFrameCount property. This property tells the number of frames that have been logged to disk.

```
vid.DiskLoggerFrameCount

ans =

    40
```

> **Note** Because it takes longer to write frames to a disk file than to memory, the value of the `DiskLoggerFrameCount` property can lag behind the value of the `FramesAvailable` property, which specifies the number of frames logged to memory.

To verify that a disk file was created, go to the directory in which the log file resides and make sure it exists. The `exist` function returns 2 if the file exists.

```
if(exist(my_log)==2)
    disp('AVI file created.')
```

6 **Close the AVI file object** — Close the AVI file to make it available outside the MATLAB environment. Closing the AVI file object ensures that the logged data is written to the disk file. Be sure to use the value of the video input object `DiskLogger` property, `vid.DiskLogger`, to reference the AVI file object, not the original variable, `aviobj`, returned by the `avifile` function.

```
aviobj = close(vid.DiskLogger);
```

Use the original variable, `aviobj`, as the return value when closing an AVI file object.

7 **Clean up** — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.

```
delete(vid)
clear vid
```

5

Working with Acquired Image Data

When you trigger an acquisition, the toolbox stores the image data in a memory buffer, a disk file, or both. To work with this data, you must bring it into the MATLAB workspace.

This chapter describes how you use video input object properties and toolbox functions to bring the logged data into the MATLAB workspace.

Overview

When a trigger occurs, the toolbox acquires frames from the video stream and logs the frames to a buffer in memory, a disk file, or both, depending on the value of the LoggingMode property. To work with this logged image data, you must bring it into the MATLAB workspace.

The following figure illustrates a group of frames being acquired from the video stream, logged to memory and disk, and brought into the MATLAB workspace as a multidimensional numeric array. Note that when frames are brought into the MATLAB workspace, they are removed from the memory buffer.
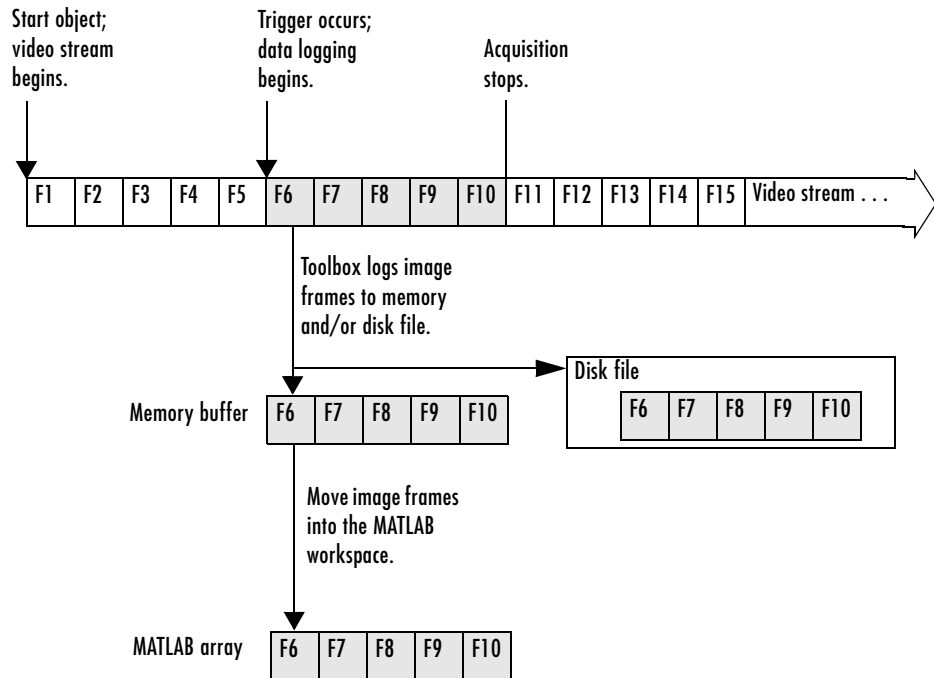
Overview of Image Acquisition

Bringing Image Data into the MATLAB Workspace

The toolbox provides three ways to move frames from the memory buffer into the MATLAB workspace:

- **Removing multiple frames from the buffer** — To move a specified number of frames from the memory buffer into the workspace, use the getdata function. The getdata function removes the frames from the memory buffer as it moves them into the workspace. The function blocks the MATLAB command line until all the requested frames are available, or until a timeout value expires. For more information, see "Moving Multiple Frames into the Workspace" on page 5-3.

- **Viewing the most recently acquired frames in the buffer** — To bring the most recently acquired frames in the memory buffer into the workspace *without* removing them from the buffer, use the peekdata function. When returning frames, peekdata starts with the most recently acquired frame and works backward in the memory buffer. In contrast, getdata starts at the beginning of the buffer, returning the oldest acquired frame first. peekdata does not block the command line and is not guaranteed to return all the frames you request. For more information, see "Viewing Frames in the Memory Buffer" on page 5-6.

- **Bring a single frame of data into the workspace** — As a convenience, the toolbox provides the getsnapshot function, which returns a single frame of data into the MATLAB workspace. Because the getsnapshot function does not require starting the object or triggering an acquisition, it is the easiest way to bring image data into the workspace. getsnapshot is independent of the memory buffer; it can return a frame even if the memory buffer is empty, and the frame returned does not affect the value of the FramesAvailable property. For more information, see "Bringing a Single Frame into the Workspace" on page 5-10.

Moving Multiple Frames into the Workspace

To move multiple frames of data from the memory buffer into the MATLAB workspace, use the getdata function. By default, getdata retrieves the number of frames specified in the FramesPerTrigger property but you can specify any number. See the getdata reference page for complete information about this function.

Note The getdata function removes the frames from the memory buffer that it moves into the workspace.

In this figure, getdata is called at T_1 with a request for 15 frames but only six frames are available in the memory buffer. getdata blocks until the specified number of frames becomes available, at T_2, at which point getdata moves the frames into the MATLAB workspace and returns control to the command prompt.
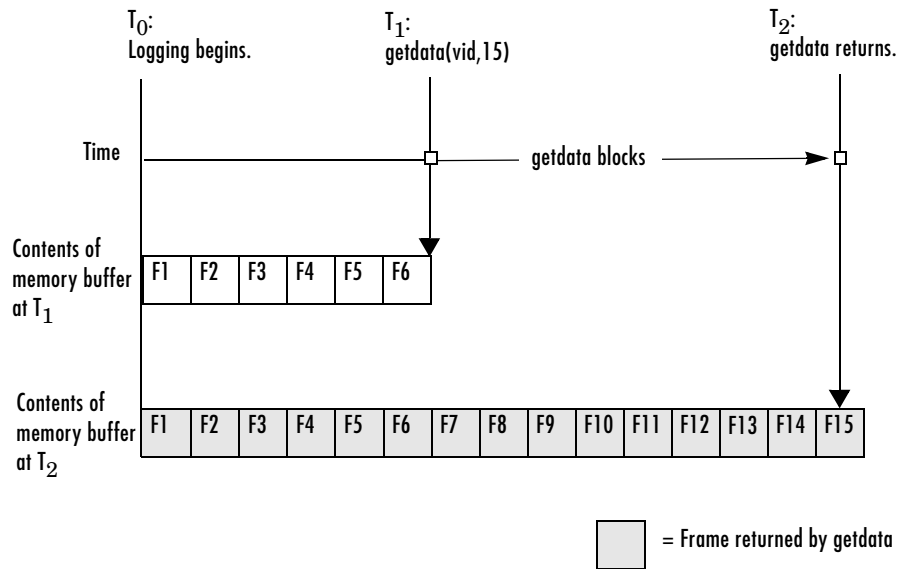
getdata Blocks Until Frames Become Available

Example: Acquiring 10 Seconds of Image Data

This example shows how you can configure an approximate time-based acquisition using the `FramesPerTrigger` property:

1 Create an image acquisition object — This example creates a video input object for a Windows image acquisition device. To run this example on your system, use the `imaqhwinfo` function to get the object constructor for your image acquisition device and substitute that syntax for the following code.

```
vid = videoinput('winvideo',1);
```

2 Configure properties — To acquire 10 seconds of data, determine the frame rate of your image acquisition device and then multiply the frame rate by the number of seconds of data you want to acquire. The product of this multiplication is the value of the `FramesPerTrigger` property.

For this example, assume a frame rate of 30 frames per second (fps). Multiplying 30 by 10, you need to set the `FramesPerTrigger` property to the value 300.

```
set(vid,'FramesPerTrigger',300)
```

3 Start the image acquisition object — Call the `start` function to start the image acquisition object.

```
start(vid)
```

The object executes an immediate trigger and begins acquiring frames of data. The `start` function returns control to the command line immediately but the object continues logging the data to the memory buffer. After logging the specified number of frames, the object stops running.

4 Bring the acquired data into the workspace — To verify that you acquired the amount of data you wanted, use the optional `getdata` syntax that returns the timestamp of every frame acquired. The difference between the first timestamp and the last timestamp should approximate the amount of data you expected.

```
[data time] = getdata(vid,300);

elapsed_time = time(3OO) - time(1)

    10.0467
```

5 **Clean up** — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.

```
delete(vid)
clear vid
```

Viewing Frames in the Memory Buffer

To view sample frames from the memory buffer without removing them, use the peekdata function.

The peekdata function always returns the most recently acquired frames in the memory buffer. For example, if you request three frames, peekdata returns the most recently acquired frame in the buffer at the time of the request and the two frames that immediately precede it.

The following figure illustrates this process. The command peekdata(vid,3) is called at three different times (T_1, T_2, and T_3). The shaded frames indicate the frames returned by peekdata at each call. (peekdata returns frames without removing them from the memory buffer.)

Note in the figure that, at T_3, only two frames have become available since the last call to peekdata. In this case, peekdata returns only the two frames, with a warning that it returned less data than was requested.

Figure showing:

T_0: Logging begins. T_1: peekdata(vid,3) T_2: peekdata(vid,3) T_3: peekdata(vid,3)

Time

Contents of memory buffer at T_1: F1 F2 F3 F4 F5 F6

Contents of memory buffer at T_2: F1 F2 F3 F4 F5 F6 F7 F8 F9 F10

Contents of memory buffer at T_3: F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12

☐ = Frame returned by peekdata

peekdata does not return F10 because it was already returned at T_2.

Frames Returned by peekdata

The following example illustrates how to use peekdata:

1 Create an image acquisition object — This example creates a video input object for a Data Translation image acquisition device. To run this example on your system, use the imaqhwinfo function to get the object constructor for your image acquisition device and substitute that syntax for the following code.

```
vid = videoinput('dt',1);
```

2 Configure properties — For this example, configure a manual trigger. You must use the triggerconfig function to specify the trigger type.

```
triggerconfig(vid,'manual')
```

In addition, configure a large enough acquisition to allow several calls to peekdata before it finishes.

```
set(vid,'FramesPerTrigger',300);
```

3 **Start the image acquisition object** — Call the start function to start the image acquisition object.

```
start(vid)
```

The video object is now running but not logging.

```
vid.Running
ans =
on

vid.Logging
ans =
off
```

Verify that the object has not acquired any frames.

```
get(vid,'FramesAcquired')
ans =
    0
```

4 **Use peekdata to view frames before a trigger** — If you call peekdata before you trigger the acquisition, peekdata can only return a single frame of data because data logging has not been initiated and the memory buffer is empty. If more than one frame is requested, peekdata issues a warning that it is returning fewer than the requested number of frames.

```
pdata = peekdata(vid,50);
Warning: PEEKDATA could not return all the frames requested.
```

Verify that peekdata returned a single frame. A single frame of data should have the same width and height as specified by the ROIPosition property and the same number of bands, as specified by the NumberOfBands property. In this example, the video format of the data is RGB so the value of the NumberOfBands property is 3.

```
whos
  Name          Size                    Bytes  Class

  pdata         96x128x3                36864  uint8 array
  vid           1x1                      1060  videoinput object
```

Verify that the object has not acquired any frames.

```
get(vid,'FramesAcquired')
ans =
   0
```

5 Trigger the acquisition — Call the `trigger` function to trigger an acquisition.

```
trigger(vid)
```

The object begins logging frames to the memory buffer.

6 View the most recently acquired frames — While the acquisition is in progress, call `peekdata` several times to view the latest frames in the memory buffer. Depending on the number of frames you request, and the timing of these requests, `peekdata` might return fewer than the number of frames you specify.

```
pdata = peekdata(vid,50);
```

To verify that `peekdata` returned the frames you requested, check the dimensions of `pdata`. `peekdata` returns a four-dimensional array of frames, where the last dimension indicates the number of frames returned.

```
whos
  Name          Size                    Bytes  Class

  pdata         4-D                   1843200  uint8 array
  vid           1x1                      1060  videoinput object

size(pdata)

ans =

    96   128    3    50
```

7 **Clean up** — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.
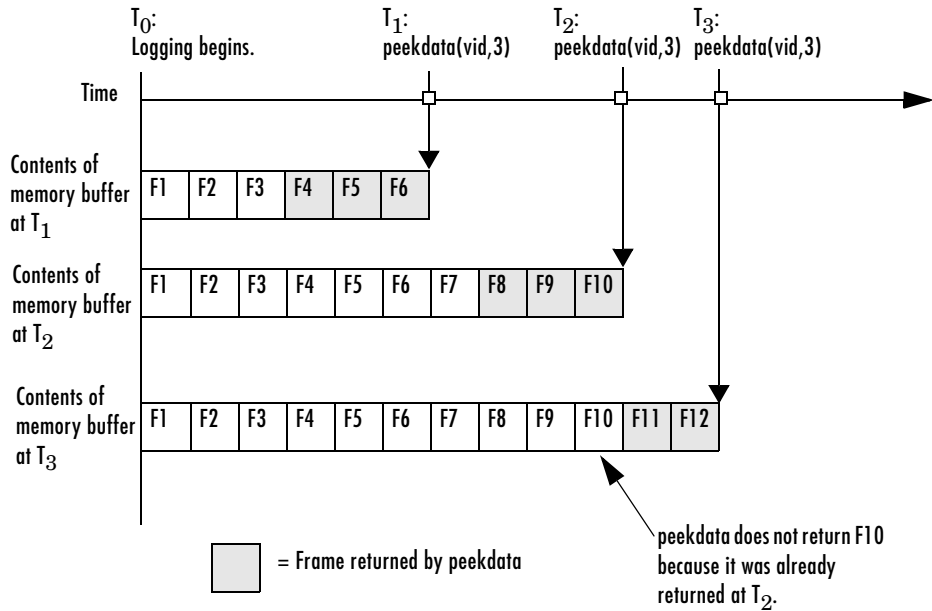
```
delete(vid)
clear vid
```

Bringing a Single Frame into the Workspace

To bring a single frame of image data into the MATLAB workspace, use the getsnapshot function. You can call the getsnapshot function at any time after object creation.

This example illustrates how simple it is to use the getsnapshot function.

1 **Create an image acquisition object** — This example creates a video input object for a Matrox device. To run this example on your system, use the imaqhwinfo function to get the object constructor for your image acquisition device and substitute that syntax for the following code.

```
vid = videoinput('matrox',1);
```

2 **Bring a frame into the workspace** — Call the getsnapshot function to bring a frame into the workspace. Note that you do not need to start the video input object before calling the getsnapshot function.

```
frame = getsnapshot(vid);
```

The getsnapshot function returns an image of the same width and height as specified by the ROIPosition property and the same number of bands as specified by the NumberOfBands property. In this example, the video format of the data is RGB so the value of the NumberOfBands property is 3.

```
whos
  Name        Size                    Bytes  Class

  frame       96x128x3                36864  uint8 array
  vid         1x1                      1060  videoinput object
```

Note that the frame returned by `getsnapshot` is not removed from the memory buffer, if frames are stored there, and does not affect the value of the `FramesAvailable` property.

3 Clean up — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.

```
delete(vid)
clear vid
```

Working with Image Data in the MATLAB Workspace

The illustrations in this documentation show the video stream and the contents of the memory buffer as a sequence of individual frames. In reality, each frame is a multidimensional array. The following figure illustrates the format of an individual frame.
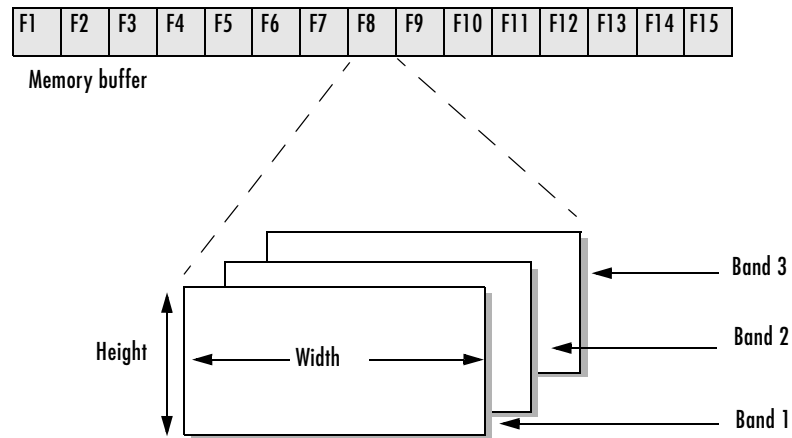
Format of an Individual Frame

This section describes how the toolbox

- Determines the dimensions of the data returned
- Determines the data type used for the data
- Determines the color space of the data

This section also describes several ways to view acquired image data.

Determining the Dimensions of Image Data

The video format used by the image acquisition device is the primary determinant of the width, height, and the number of bands in each image frame. Image acquisition devices typically support multiple video formats. You select the video format when you create the video input object (described in "Specifying the Video Format" on page 3-12). The video input object stores the video format in the `VideoFormat` property.

Industry-standard video formats, such as RS170 or PAL, include specifications of the image frame width and height, referred to as the *image resolution*. For example, the RS170 standard defines the width and height of the image frame as 640-by-480 pixels. Other devices, such as digital cameras, support the definition of many different, nonstandard image resolutions. The video input object stores the video resolution in the `VideoResolution` property.

Note Because devices typically express video resolution as width-by-height, the toolbox uses this convention for the `VideoResolution` property. However, when data is brought into the MATLAB workspace, the image frame dimensions are listed in reverse order, height-by-width, because MATLAB expresses matrix dimensions as row-by-column.

For devices that support the definition of a region of interest (ROI) in the image being captured, the dimensions of the ROI determine the dimensions of the image frames returned. The `VideoResolution` property specifies the dimensions of the image data being provided by the device; the `ROIPosition` property specifies the dimensions of the image frames being logged. See the `ROIPosition` property reference page for more information.

Each image frame is three dimensional; however, the video format determines the number of bands in the third dimension. For color video formats, such as RGB, each image frame has three bands: one each for the red, green, and blue data. Other video formats, such as the grayscale RS170 standard, have only a single band. The video input object stores the size of the third dimension in the `NumberOfBands` property.

The following example illustrates how video format affects the size of the image frames returned.

1 **Select a video format** — Use the imaqhwinfo function to view the list of video formats supported by your image acquisition device. This example shows the video formats supported by a Matrox Orion frame grabber. The formats are industry standard, such as RS170, NTSC, and PAL. These standards define the image resolution.

```
info = imaqhwinfo('matrox');

info.DeviceInfo.SupportedFormats

ans =
  Columns 1 through 4

   'M_RS170'   'M_RS170_VIA_RGB'   'M_CCIR'   'M_CCIR_VIA_RGB'

  Columns 5 through 8

'M_NTSC'     'M_NTSC_RGB'       'M_NTSC_YC'     'M_PAL'

  Columns 9 through 10

'M_PAL_RGB'     'M_PAL_YC'
```

2 **Create an image acquisition object** — This example creates a video input object for a Matrox image acquisition device using the default video format, RS170. To run this example on your system, use the imaqhwinfo function to get the object constructor for your image acquisition device and substitute that syntax for the following code.

```
vid = videoinput('matrox',1);
```

3 **View the video format and video resolution properties** — The toolbox creates the object with the default video format. This format defines the video resolution.

```
get(vid,'VideoFormat')

ans =

   M_RS170
```

```
get(vid,'VideoResolution')

ans =

   [640 480]
```

4 **Bring a single frame into the workspace** — Call the `getsnapshot` function to bring a frame into the workspace.

```
frame = getsnapshot(vid);
```

The dimensions of the returned data reflect the image resolution and the value of the `NumberOfBands` property.

```
vid.NumberOfBands
ans =

   1

size(frame)

ans =

   480 640
```

5 **Start the image acquisition object** — Call the `start` function to start the image acquisition object.

```
start(vid)
```

The object executes an immediate trigger and begins acquiring frames of data.

6 **Bring multiple frames into the workspace** — Call the `getdata` function to bring multiple image frames into the MATLAB workspace.

```
data = getdata(vid,10);
```

The `getdata` function brings 10 frames of data into the workspace. Note that the returned data is a four-dimensional array: each frame is three-dimensional and the nth frame is indicated by the fourth dimension.

```
size(data)

ans =

    480 640 1 10
```

7 **Clean up** — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.

```
delete(vid)
clear vid
```

Determining the Data Type of Image Frames

By default, the toolbox returns image frames in the data type used by the image acquisition device. If there is no MATLAB data type that matches the object's native data type, getdata chooses a MATLAB data type that preserves numerical accuracy. For example, in RGB 555 format, each color component is expressed in 5-bits. getdata returns each color as a uint8 value.

You can specify the data type you want getdata to use for the returned data. For example, if you want the image frame returned as an array of class double, use this syntax. To see a list of all the data types supported, see the getdata reference page.

1 **Create an image acquisition object** — This example creates a video input object for a Matrox image acquisition device. To run this example on your system, use the imaqhwinfo function to get the object constructor for your image acquisition device and substitute that syntax for the following code.

```
vid = videoinput('matrox',1);
```

2 **Bring a single frame into the workspace** — Call the getsnapshot function to bring a frame into the workspace.

```
frame = getsnapshot(vid);
```

3 **View the class of the returned data** — Use the class function to determine the data type used for the returned image data.

```
class(frame)

ans =

   uint8
```

4 Clean up — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.

```
delete(vid)
clear vid
```

Determining the Color Space

When you select the video format of the video stream, you can often also choose the color space, that is, the way color information is represented numerically. Many devices represent colors as RGB values. In this color space, colors are represented as a combination of various intensities of red, green, and blue. There are, however, other color spaces.

For example, the YCbCr color space (also known as YUV) is widely used for digital video. In this format, luminance (brightness or intensity) information is stored as a single component (Y). Chrominance (color) information is stored as two color-difference components (Cb and Cr). Cb represents the difference between the blue component and a reference value. Cr represents the difference between the red component and a reference value.

Note To display image frames using the image or imagesc functions, the data must use the RGB color space.

The toolbox can return image data in several color spaces, including grayscale, RGB, YCbCr, NTSC, and HSV. To determine the color representation used by your data, check the value of the ReturnedColorSpace property. For example, the color space used for the data in the previous example was RGB.

1 Create an image acquisition object — This example creates a video input object for a generic Windows image acquisition device. To run this example on your system, use the imaqhwinfo function to get the object constructor for your image acquisition device and substitute that syntax for the following code.

```
vid = videoinput('winvideo',1);
```

2 View the color space used for the data — View the value of the ReturnedColorSpace property.

```
vid.ReturnedColorSpace

ans =

rgb
```

3 Clean up — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.

```
delete(vid)
clear vid
```

Viewing Acquired Data

Once you bring the data into the MATLAB workspace, you can view it as you would any other image in MATLAB.

The Image Acquisition Toolbox includes a function, imaqmontage, that you can use to view all the frames of a multiframe image array in a single MATLAB image object. imaqmontage arranges the frames so that they roughly form a square. imaqmontage can be useful for visually comparing multiple frames.

MATLAB includes two functions, image and imagesc, that display images in a figure window. Both functions create a MATLAB image object to display the frame. You can use image object properties to control aspects of the display. The imagesc function automatically scales the input data.

The Image Processing Toolbox includes an additional display routine called imshow. Like image and imagesc, this function creates a MATLAB image object. However, imshow also automatically sets various image object properties to optimize the display.

Retrieving Timing Information

This section describes how the toolbox provides acquisition timing information, particularly,

- Determining when a trigger executed
- Determining when a particular frame was acquired

To see an example of retrieving timing information, see "Example: Determining the Frame Delay Duration" on page 5-21.

Determining When a Trigger Executed

To determine when the a trigger executed, check the information returned by a trigger event in the object's event log. You can also get access to this information in a callback function associated with a trigger event. For more information, see "Retrieving Event Information" on page 6-6.

As a convenience, the toolbox returns the time of the *first* trigger execution in the video input object's InitialTriggerTime property. This figure indicates which trigger is returned in this property when multiple triggers are configured.
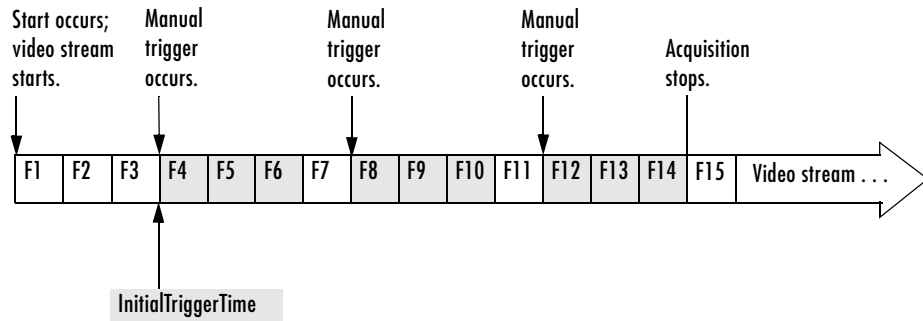
InitialTriggerTime Records First Trigger Execution

The trigger timing information is stored in MATLAB clock vector format. The following example displays the time of the first trigger for the video input object vid. The example uses the MATLAB datestr function to convert the information into a form that is more convenient to view.

```
datestr(vid.InitialTriggerTime)

ans =

02-Mar-2003 13:00:24
```

Determining When a Frame Was Acquired

The toolbox provides two ways to determine when a particular frame was acquired:

- By the absolute time of the acquisition
- By the elapsed time relative to the execution of the trigger

You can use the getdata function to retrieve both types of timing information.

Getting the Relative Acquisition Time

When you use the getdata function, you can optionally specify two return values. One return value contains the image data; the other return value contains a vector of timestamps that measure, in seconds, the time when the frame was acquired relative to the first trigger.

```
[data time] = getdata(vid);
```

To see an example, see "Example: Determining the Frame Delay Duration" on page 5-21.

Getting the Absolute Acquisition Time

When you use the getdata function, you can optionally specify three return values. The first contains the image data, the second contains a vector of relative acquisition times, and the third is an array of structures where each structure contains metadata associated with a particular frame.

```
[data time meta ] = getdata(vid);
```

Each structure in the array contains the following four fields. The AbsTime field contains the absolute time the frame was acquired. You can also retrieve this

metadata by using event callbacks. See "Retrieving Event Information" on page 6-6 for more information.

Frame Metadata

Field Name	
AbsTime	Absolute time the frame was acquired, returned in MATLAB clock format `[year month day hour minute seconds]`
FrameNumber	Frame number relative to when the object was started
RelativeFrame	Frame number relative to trigger execution
TriggerIndex	Trigger the event is associated with. For example, when the object starts, the associated trigger is 0. Upon stop, it is equivalent to the TriggersExecuted property.

Example: Determining the Frame Delay Duration

To illustrate, this example calculates the duration of the delay specified by the TriggerFrameDelay property.

1 **Create an image acquisition object** — This example creates a video input object for a Data Translation image acquisition device using the default video format. To run this example on your system, use the imaqhwinfo function to get the object constructor for your image acquisition device and substitute that syntax for the following code.

```
vid = videoinput('dt',1);
```

2 **Configure properties** — For this example, configure a trigger frame delay large enough to produce a noticeable duration.

```
set(vid,'TriggerFrameDelay',50)
```

3 **Start the image acquisition object** — Call the start function to start the image acquisition object.

```
start(vid)
```

The object executes an immediate trigger and begins acquiring frames of data. The start function returns control to the command line immediately but data logging does not begin until the trigger frame delay expires. After logging the specified number of frames, the object stops running.

4 **Bring the acquired data into the workspace** — Call the getdata function to bring frames into the workspace. Specify a return value to accept the timing information returned by getdata.

```
[data time ] = getdata(vid);
```

The variable time is a vector that contains the time each frame was logged, measured in seconds, relative to the execution of the first trigger. Check the first value in the time vector. It should reflect the duration of the delay before data logging started.

```
time

time =

    4.9987
    5.1587
    5.3188
    5.4465
    5.6065
    5.7665
    5.8945
    6.0544
    6.2143
    6.3424
```

5 **Clean up** — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.

```
delete(vid)
clear vid
```

Using Events and Callbacks

You can enhance the power and flexibility of your image acquisition application by using *event callbacks*. An event is a specific occurrence that can happen while an image acquisition object is running. The toolbox defines a set of events that include starting, stopping, or acquiring frames of data.

When a particular event occurs, the toolbox can execute a function that you specify. This is called a *callback*. Certain events can result in one or more callbacks. You can use callbacks to perform processing tasks while your image acquisition object is running. For example, you can display a message, analyze data, or perform other tasks. Callbacks are controlled through video input object properties. Each event type has an associated property. You specify the function that you want executed as the value of the property.

Example: Using the Default Callback Function

To illustrate how to use callbacks, this section presents a simple example that creates an image acquisition object and associates a callback function with the start event, trigger event, and stop event. For information about all the event callbacks supported by the toolbox, see "Event Types" on page 6-4.

The example uses the default callback function provided with the toolbox, `imaqcallback`. The default callback function displays the name of the object along with information about the type of event that occurred and when it occurred. To learn how to create your own callback functions, see "Creating and Executing Callback Functions" on page 6-11.

This example illustrates how to use the default callback function.

1 **Create an image acquisition object** — This example creates a video input object for a Matrox image acquisition device. To run this example on your system, use the `imaqhwinfo` function to get the object constructor for your image acquisition device and substitute that syntax for the following code.

```
vid = videoinput('matrox',1);
```

2 **Configure properties** — Set the values of three callback properties. The example uses the default callback function `imaqcallback`.

```
set(vid,'StartFcn',@imaqcallback)
set(vid,'TriggerFcn',@imaqcallback)
set(vid,'StopFcn',@imaqcallback)
```

For this example, specify the amount of data to log.

```
set(vid, 'FramesPerTrigger',100);
```

3 **Start the image acquisition object** — Start the image acquisition object. The object executes an immediate trigger, acquires 100 frames of data, and then stops. With the three callback functions enabled, the object outputs information about each event as it occurs.

```
start(vid)
Start event occurred at 14:38:46 for video input object: M_RS170-matrox-1.
Trigger event occurred at 14:38:46 for video input object: M_RS170-matrox-1.
Stop event occurred at 14:38:49 for video input object: M_RS170-matrox-1.
```

4 Clean up — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.

```
delete(vid)
clear vid
```

Event Types

The Image Acquisition Toolbox supports several different types of events. Each event type has an associated video input object property that you can use to specify the function that executes when the event occurs.

This table lists the supported event types, the name of the video input object property associated with the event, and a brief description of the event. For detailed information about these callback properties, see the property reference information for the property.

The toolbox generates a specific set of information for each event and stores it in an event structure. To learn more about the contents of these event structures and how to retrieve this information, see "Retrieving Event Information" on page 6-6.

Events and Callback Function Properties

Event	Callback Property	Description
Error	ErrorFcn	The toolbox generates an error event when a run-time error occurs, such as a hardware error or timeout. Run-time errors do not include configuration errors such as setting an invalid property value.
		When an error event occurs, the toolbox executes the function specified by the ErrorFcn property. By default, the toolbox executes the default callback function for this event, imaqcallback, which displays the error message at the MATLAB command line.
Frames Acquired	FramesAcquiredFcn	The toolbox generates a frames acquired event when a specified number of frames have been acquired. You use the FramesAcquiredFcnCount property to specify this number.
		When a frames acquired event occurs, the toolbox executes the function specified by the FramesAcquiredFcn property.

Events and Callback Function Properties (Continued)

Event	Callback Property	Description
Start	StartFcn	The toolbox generates a start event when an object is started. You use the start function to start an object. **Note**: If an error occurs in the start callback function, the object never starts. When a start event occurs, the toolbox executes the function specified by the StartFcn property.
Stop	StopFcn	The toolbox generates a stop event when the object stops running. An object stops running when the stop function is called, the specified number of frames is acquired, or a run-time error occurs. When a stop event occurs, the toolbox executes the function specified by the StopFcn property.
Timer	TimerFcn	The toolbox generates a timer event when a specified amount of time expires. Time is measured relative to when the object starts running. You use the TimerPeriod property to specify the amount of time. **Note**: Some timer events might not execute if your system is significantly slowed or if the TimerPeriod is set too small. When a timer event occurs, the toolbox executes the function specified by the TimerFcn property.
Trigger	TriggerFcn	The toolbox generates a trigger event when a trigger executes. The video input object executes immediate triggers. You execute manual triggers by calling the trigger function. The image acquisition device executes hardware triggers when a specified condition is met. When a trigger event occurs, the toolbox executes the function specified by the TriggerFcn property.

Retrieving Event Information

Each event has associated with it a set of information, generated by the toolbox and stored in an event structure. This information includes the event type, the time the event occurred, and other event-specific information. While a video input object is running, the toolbox records event information in the object's `EventLog` property. You can also access the event structure associated with an event in a callback function.

This section

- Defines the information in an event structure for all event types
- Describes how to retrieve information from the `EventLog` property

For information about accessing event information in a callback function, see "Creating and Executing Callback Functions" on page 6-11.

Event Structures

An event structure contains two fields: `Type` and `Data`. For example, this is an event structure for a trigger event:

```
Type: 'Trigger'
Data: [1x1 struct]
```

The `Type` field is a text string that specifies the event type. For a trigger event, this field contains the text string `'Trigger'`.

The `Data` field is a structure that contains information about the event. The composition of this structure varies depending on which type of event occurred. For information about the information associated with specific events, see the following sections:

- "Data Fields for Start, Stop, Frames Acquired, and Trigger Events" on page 6-7
- "Data Fields for Error Events" on page 6-7
- "Data Fields for Timer Events" on page 6-8

Data Fields for Start, Stop, Frames Acquired, and Trigger Events

For start, stop, frames acquired, and trigger events, the Data structure
contains these fields.

Field Name	Description
AbsTime	Absolute time the event occurred, returned in MATLAB clock format `[year month day hour minute seconds]`
FrameMemoryLimit	Amount of memory allotted for frame storage. You can specify this value using the imaqmem function.
FrameMemoryUsed	Amount of frame memory that is currently in use
FrameNumber	Frame number relative to when the object was started
RelativeFrame	Frame number relative to the execution of a trigger
TriggerIndex	Trigger the event is associated with. For example, upon start, the associated trigger is 0. Upon stop, it is equivalent to the TriggersExecuted property.

Data Fields for Error Events

For error events, the Data structure contains these fields.

Field Name	Description
AbsTime	Absolute time the event occurred, returned in MATLAB clock format `[year month day hour minute seconds]`
FrameMemoryLimit	Amount of memory allotted for frame storage. You can specify this value using the imaqmem function.

Field Name	Description
FrameMemoryUsed	Amount of frame memory that is currently in use
Message	Text message associated with the error
MessageID	MATLAB message identifier associated with the error

Data Fields for Timer Events

For timer events, the Data structure contains these fields.

Field Name	Description
AbsTime	Absolute time the event occurred, returned in MATLAB clock format [year month day hour minute seconds]
FrameMemoryLimit	Amount of memory allotted for frame storage. You can specify this value using the imaqmem function.
FrameMemoryUsed	Amount of frame memory that is currently in use

Example: Accessing Data in the Event Log

While a video input object is running, the toolbox stores event information in the object's EventLog property. The value of this property is an array of event structures. Each structure represents one event. For detailed information about the composition of an event structure for each type of event, see "Event Structures" on page 6-6.

The toolbox adds event structures to the EventLog array in the order in which the events occur. The first event structure reflects the first event recorded, the second event structure reflects the second event recorded, and so on.

Note Only start, stop, error, and trigger events are recorded in the `EventLog` property. Frames-acquired events and timer events are not included in the `EventLog`. Event structures for these events (and all the other events) are available to callback functions. For more information, see "Creating and Executing Callback Functions" on page 6-11.

To illustrate the event log, this example creates a video input object, runs it, and then examines the object's `EventLog` property:

1 **Create an image acquisition object** — This example creates a video input object for a Matrox image acquisition device. To run this example on your system, use the `imaqhwinfo` function to get the object constructor for your image acquisition device and substitute that syntax for the following code.

```
vid = videoinput('matrox',1);
```

2 **Start the image acquisition object** — Start the image acquisition object. By default, the object executes an immediate trigger, acquires 10 frames of data, and then stops.

```
start(vid)
```

3 **View the event log** — Access the `EventLog` property of the video input object. The execution of the video input object generated three events: start, trigger, and stop. Thus the value of the `EventLog` property is a 1x3 array of event structures.

```
events = vid.EventLog
events =

1x3 struct array with fields:
    Type
    Data
```

To list the events that are recorded in the `EventLog` property, examine the contents of the `Type` field.

```
{events.Type}
ans =
    'Start'    'Trigger'    'Stop'
```

To get information about a particular event, access the `Data` field in that event structure. The example retrieves information about the trigger event.

```
trigdata = events(2).Data

trigdata =

             AbsTime: [2002 12 13 15 23 45.9700]
    FrameMemoryLimit: 139427840
     FrameMemoryUsed: 0
         FrameNumber: 0
       RelativeFrame: 0
        TriggerIndex: 1
```

4 Clean up — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.

```
delete(vid)
clear vid
```

Creating and Executing Callback Functions

The power of using event callbacks is the processing that you can perform in response to events. You decide which events you want to associate callbacks with and the functions these callbacks execute.

This section

- Describes how to create a callback function
- Describes how to specify the function as the value of a callback property
- Provides two examples of using event callbacks:
  - Shows how to use callbacks to view a sample frame from the frames being acquired
  - Uses callback to implement a simple memory monitoring function

Note Callback function execution might be delayed if the callback involves a CPU-intensive task such as updating a figure.

Creating Callback Functions

M-file callback functions require at least two input arguments:

- The image acquisition object
- The event structure associated with the event

The function header for this callback function illustrates this basic syntax.

```
function mycallback(obj,event)
```

The first argument, obj, is the image acquisition object itself. Because the object is available, you can use in your callback function any of the toolbox functions, such as getdata, that require the object as an argument. You can also access all object properties.

The second argument, event, is the event structure associated with the event. This event information pertains only to the event that caused the callback function to execute. For a complete list of supported event types and their associated event structures, see "Event Structures" on page 6-6.

In addition to these two required input arguments, you can also specify additional, application-specific arguments for your callback function.

Note To receive the object and event arguments, and any additional arguments, you must use a cell array when specifying the name of the function as the value of a callback property. For more information, see "Specifying Callback Functions" on page 6-13.

Example: Writing a Callback Function

To illustrate, this example implements a callback function for a frames-acquired event. This callback function enables you to monitor the frames being acquired by viewing a sample frame periodically.

To implement this function, the callback function acquires a single frame of data and displays the acquired frame in a MATLAB figure window. The function also accesses the event structure passed as an argument to display the timestamp of the frame being displayed. The drawnow command in the callback function forces MATLAB to update the display.

```
function display_frame(obj,event)

sample_frame = peekdata(obj,1);

imagesc(sample_frame);

drawnow; % force an update of the figure window

abstime = event.Data.AbsTime;

t = fix(abstime);

sprintf('%s %d:%d:%d','timestamp', t(4),t(5),t(6))
```

To see how this function can be used as a callback, see "Example: Viewing a Sample Frame" on page 6-15.

Specifying Callback Functions

You associate a callback function with a specific event by setting the value of the event's callback property. The video input object supports callback properties for all types of events.

You can specify the callback function as the value of the property in any of three ways:

- Text string
- Cell array
- Function handle

The following sections provide more information about each of these options.

Note To access the object or event structure passed to the callback function, you must specify the function as a cell array or as a function handle.

Using a Text String to Specify Callback Functions

You can specify the callback function as a string. For example, this code specifies the callback function mycallback as the value of the start event callback property StartFcn for the video input object vid.

```
vid.StartFcn = 'mycallback';
```

In this case, the callback is evaluated in the MATLAB workspace.

Using a Cell Array to Specify Callback Functions

You can specify the callback function as a text string inside a cell array.

For example, this code specifies the callback function mycallback as the value of the start event callback property StartFcn for the video input object vid.

```
vid.StartFcn = {'mycallback'};
```

To specify additional parameters, include them as additional elements in the cell array.

```
time = datestr(now,0);
vid.StartFcn = {'mycallback',time};
```

6-13

The first two arguments passed to the callback function are still the video input object (obj) and the event structure (event). Additional arguments follow these two arguments.

Using Function Handles to Specify Callback Functions

You can specify the callback function as a function handle.

For example, this code specifies the callback function mycallback as the value of the start event callback property StartFcn for the video input object vid.

```
vid.StartFcn = @mycallback;
```

To specify additional parameters, include the function handle and the parameters as elements in the cell array.

```
time = datestr(now,0);
vid.StartFcn = {@mycallback,time};
```

If you are executing a local callback function from within an M-file, you must specify the callback as a function handle.

Specifying a Toolbox Function as a Callback

In addition to specifying callback functions of your own creation, you can also specify the start, stop, or trigger toolbox functions as callbacks. For example, this code sets the value of the stop event callback to the Image Acquisition Toolbox start function.

```
vid.StopFcn = @start;
```

Disabling Callbacks

If an error occurs in the execution of the callback function, the toolbox disables the callback and displays a message similar to the following.

```
start(vid)
??? Error using ==> frames_cb
Too many input arguments.

Warning: The FramesAcquiredFcn callback is being disabled.
```

To enable a callback that has been disabled, set the value of the property associated with the callback or restart the object.

Example: Viewing a Sample Frame

This example creates a video input object and sets the frames acquired event callback function property to the display_frame function, created in "Example: Writing a Callback Function" on page 6-12.

The example sets the TriggerRepeat property of the object to 4 so that 50 frames are acquired. When run, the example displays a sample frame from the acquired data every time five frames have been acquired.

1 **Create an image acquisition object** — This example creates a video input object for a Matrox image acquisition device. To run this example on your system, use the imaqhwinfo function to get the object constructor for your image acquisition device and substitute that syntax for the following code.

```
vid = videoinput('matrox',1);
```

2 **Configure property values** — This example sets the FramesPerTrigger value to 30 and the TriggerRepeat property to 4. The example also specifies as the value of the FramesAcquiredFcn callback the event callback function display_frame, created in "Example: Writing a Callback Function" on page 6-12. The object will execute the FramesAcquiredFcn every five frames, as specified by the value of the FramesAcquiredFcnCount property.

```
set(vid,'FramesPerTrigger',30);
set(vid,'TriggerRepeat',4);
set(vid,'FramesAcquiredFcnCount',5);
set(vid,'FramesAcquiredFcn',{'display_frame'});
```

3 **Acquire data** — Start the video input object. Every time five frames are acquired, the object executes the display_frame callback function. This callback function displays the most recently acquired frame logged to the memory buffer.

```
start(vid)
```

4 **Clean up** — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.

```
delete(vid)
clear vid
```

Example: Monitoring Memory Usage

This example creates a callback function for a timer event that displays the
toolbox's current memory usage and stops the acquisition when the available
memory for frame storage falls below a specified amount.

Creating the Memory Monitor Callback Function

This callback function implements a simple memory usage monitoring
function. The callback function uses the imaqmem function to retrieve two
memory usage statistics, FrameMemoryLimit and FrameMemoryUsed, and then
calculates the amount of memory that is currently left for allocating frames.
When the amount of memory available falls below a specified value, the
function outputs a message and stops the object.

```
function mem_mon(obj,event)

out = imaqmem;

mem_left = out.FrameMemoryLimit - out.FrameMemoryUsed;

msg = 'Memory left for frames';
msg2 = 'Memory load';
low_limit = 2000000;

if(mem_left > low_limit)
    sprintf('%s: %d \n%s: %d',msg, mem_left,msg2, out.MemoryLoad)
else
    disp('Memory available for frames getting low.');
    disp('Stopping acquisition.')
    stop(obj);
end
```

Running the Example

The example acquires frames until the amount of memory left for frame
storage reaches a lower limit specified in the callback function.

1 **Create an image acquisition object** — This example creates a video input
object for a Matrox image acquisition device. To run this example on your
system, use the imaqhwinfo function to get the object constructor for your
image acquisition device and substitute that syntax for the following code.

```
vid = videoinput('matrox',1);
```

2 Configure property values — This example sets up a continuous acquisition by setting the FramesPerTrigger value to Inf. The example also specifies the timer event callback function mem_mon, created in "Creating the Memory Monitor Callback Function" on page 6-16, as the value of the TimerFcn callback. The object will execute the TimerFcn every five seconds, as specified by the value of the TimerPeriod property.

```
set(vid,'FramesPerTrigger',Inf);
set(vid,'TimerPeriod',5);
set(vid,'TimerFcn',{'mem_mon'});
```

3 Acquire data — Start the video input object. Every 5 seconds, the object executes the callback function associated with the timer event. This function outputs the current memory available for frame storage and the memory load statistic. When the amount of memory reaches the specified lower limit, the callback function stops the acquisition.

```
start(vid)
ans =

ans =

Memory left for frames: 27791360
Memory load: 88

ans =

Memory left for frames: 26316800
Memory load: 88

ans =

Memory left for frames: 24842240
Memory load: 89

.
.
.
```

```
Memory left for frames: 2969600
Memory load: 97

Memory available for frames getting low.
Stopping acquisition.
```

4 **Clean up** — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.

```
delete(vid)
clear vid
```

Function Reference

This chapter provides descriptions of all toolbox functions that you can use directly.

Getting Command Line Function Help (p. 7-2)	Describes how to get help for toolbox functions at the command line
Functions — By Category (p. 7-3)	Contains a series of tables that provide brief descriptions of Image Acquisition Toolbox functions, arranged by category, followed by individual reference pages for each function

Getting Command Line Function Help

To get command-line function help, you can use the MATLAB `help` function. For example, to get help for the `getsnapshot` function, type

```
help getsnapshot
```

However, the Image Acquisition Toolbox provides "overloaded" versions of several MATLAB functions. That is, it provides toolbox-specific implementations of these functions using the same function name.

For example, the Image Acquisition Toolbox provides an overloaded version of the `delete` function. If you type

```
help delete
```

you get help for the MATLAB version of this function. You can determine if a function is overloaded by examining the last section of the help. For `delete`, the help contains the following overloaded versions (not all are shown).

```
Overloaded methods
help char/delete.m
help scribehandle/delete.m
help scribehgobj/delete.m
.
.
.
help imaqdevice/delete.m
```

To obtain help on the Image Acquisition Toolbox version of this function, type

```
help imaqdevice/delete
```

To avoid having to specify which overloaded version you want to view, use the `imaqhelp` function.

```
imaqhelp delete
```

You can also use this function to get help on image acquisition object properties. For more information on overloaded functions and class directories, refer to "MATLAB Classes and Objects" in the Help browser.

Functions — By Category

This section provides brief descriptions of all the functions in the Image Acquisition Toolbox. The functions are listed in tables in the following broad categories.

- General functions
- Trigger functions
- Data functions
- Tools

General Object Functions

Video input objects have one or more video source objects associated with them. In this table, functions that work on both types of object use the phrase "image acquisition object" to refer to both types of object.

clear	Clear image acquisition object from the workspace
delete	Remove video input object from memory
disp	Display summary information about an image acquisition object
get	Retrieve current settings of all image acquisition object properties
getselectedsource	Retrieve the currently selected video source object, associated with the video input object
imaqfind	Find all image acquisition objects in memory
isvalid	Determine if image acquisition object is a valid object
load	Load image acquisition objects into the workspace
save	Save image acquisition objects to a MAT-file
set	Retrieve a list of all settable properties or set the values of specific image acquisition object properties
start	Start a video input object

stop	Stop a video input object
videoinput	Create a video input object
wait	Block until a video input object stops running

Trigger Functions

trigger	Initiate logging of image data
triggerconfig	Set the values of TriggerType, TriggerCondition, and TriggerSource properties
triggerinfo	Retrieve all valid combinations of values for the TriggerType, TriggerCondition, and TriggerSource properties

Data Functions

flushdata	Delete data stored in the memory buffer
getdata	Bring data from the memory buffer into the MATLAB workspace, deleting data from the memory buffer as it is moved
getsnapshot	Bring a single frame of data into the MATLAB workspace
peekdata	Bring data from the memory buffer into the MATLAB workspace without deleting it from the memory buffer

Tools

closepreview	Close a preview window or all open preview windows
imaqhelp	Return image acquisition object function and property help
imaqhwinfo	Return information about image acquisition hardware on the system

imaqmem	Limit or display memory already in use by the Image Acquisition Toolbox
imaqmontage	Display a sequence of image frames as a montage
imaqreset	Return toolbox to its original state
preview	Open a live display of the current image acquisition device's scene
propinfo	Return image acquisition object property information

clear

Purpose   Clear an image acquisition object from the workspace

Syntax   `clear obj`

Description   `clear obj` removes the image acquisition object `obj` from the MATLAB
workspace. `obj` can be either a video input object or a video source object.

Note If you clear a video input object that is running (the `Running` property is
set to `'on'`), the object continues executing.

You can restore cleared objects to the MATLAB workspace with the `imaqfind`
function.

To remove an image acquisition object from memory, use the `delete` function.

See Also   `delete`, `imaqfind`, `isvalid`

Purpose   Close image preview windows

Syntax   closepreview(obj)
                     closepreview

Description   closepreview(obj) closes the image preview window associated with image
                     acquisition object obj.

                     closepreview closes all image preview windows for all image acquisition
                     objects.

See Also   preview, videoinput

delete

Purpose   Remove image acquisition object from memory

Syntax   delete(obj)

Description   delete(obj) removes obj, an image acquisition object or array of image
                  acquisition objects, from memory. Use delete to free memory at the end of an
                  image acquisition session.

                  If obj is an array of image acquisition objects and one of the objects cannot be
                  deleted, the delete function deletes the objects that can be deleted and returns
                  a warning.

                  When obj is deleted, it becomes invalid and cannot be reused. Use the clear
                  command to remove invalid image acquisition objects from the MATLAB
                  workspace.

                  If multiple references to an image acquisition object exist in the workspace,
                  deleting the image acquisition object invalidates the remaining references. Use
                  the clear command to delete the remaining references to the object from the
                  workspace.

                  If the image acquisition object obj is running or being previewed, the delete
                  function stops the object and closes the preview window before deleting it.

Example   vid = videoinput('winvideo', 1);
                      preview(vid);
                      delete(vid);

See Also   imaqfind, isvalid, videoinput

Purpose   Display method for image acquisition objects

Syntax   ```
obj
disp(obj)
```

**Description**  `obj` or `disp(obj)` displays summary information for image acquisition object `obj`.

If `obj` is an array of image acquisition objects, `disp` outputs a table of summary information about the image acquisition objects in the array.

In addition to the syntax shown above, you can display summary information for `obj` by excluding the semicolon when

- Creating a image acquisition object, using the `videoinput` function
- Configuring property values using the dot notation

**Example**     This example illustrates the summary display of a video input object.

```
vid = videoinput('winvideo')

Summary of Video Input Object Using 'IBM PC Camera'.

   Acquisition Source(s):  input1 is available.

  Acquisition Parameters:  'input1' is the current selected source.
                           10 frames per trigger using the selected source.
                           'RGB555_128x96' video data to be logged upon START.
                           Grabbing first of every 1 frame(s).
                           Log data to 'memory' on trigger.

      Trigger Parameters:  1 'immediate' trigger(s) on START.

                  Status:  Waiting for START.
                           0 frames acquired since starting.
                           0 frames available for GETDATA.
```

This example shows the summary information displayed for an array of video input objects.

```
vid2 = videoinput('winvideo');

[vid vid2]

   Video Input Object Array:

   Index:    Type:          Name:
   1         videoinput     RGB555_128x96-winvideo-1
   2         videoinput     RGB555_128x96-winvideo-1
```

See Also   videoinput

Purpose   Remove data from the memory buffer used to store acquired image frames

Syntax   ```
flushdata(obj)
flushdata(obj,mode)
```

Description `flushdata(obj)` removes all the data from the memory buffer used to store acquired image frames. `obj` can be a single video input object or an array of video input objects.

`flushdata(obj,mode)` removes all the data from the memory buffer used to store acquired image frames, where `mode` can have either of the following values:

mode	Description
`'all'`	Removes all the data from the memory buffer and sets the `FramesAvailable` property to 0 for the video input object `obj`. This is the default mode when none is specified, `flushdata(obj)`.
`'triggers'`	Removes the data acquired during the oldest trigger executed from the memory buffer. `TriggerRepeat` must be greater than 0 and `FramesPerTrigger` must not be set to `inf`.

See Also   `imaqhelp`, `getdata`, `peekdata`, `propinfo`, `videoinput`

Purpose	Display or get image acquisition object properties
Syntax	get(obj) V = get(obj) V = get(obj,'PropertyName')
Description	get(obj) displays all property names and their current values for image acquisition object obj. V = get(obj) returns a structure, V, in which each field name is the name of a property of obj and each field contains the value of that property. V = get(obj,'PropertyName') returns the value, V, of the property specified in PropertyName for image acquisition object obj. If PropertyName is a 1-by-N or N-by-1 cell array of strings containing property names, V is a 1-by-N cell array of values. If obj is a vector of image acquisition objects, V is an M-by-N cell array of property values where M is equal to the length of obj and N is equal to the number of properties specified.

Example	vid = videoinput('matrox', 1);
	
	get(vid, {'FramesPerTrigger','FramesAcquired'})
	
	out = get(vid, 'LoggingMode')
	
	get(vid);

See Also	set, videoinput

Purpose   Return acquired image frames to MATLAB workspace

Syntax
```
data = getdata(obj)
data = getdata(obj, n)
data = getdata(obj, n, type)
data = getdata(obj, n, type, format)
[data, time] = getdata(...)
[data, time, metadata] = getdata(...)
```

Description   `data = getdata(obj)` returns `data`, which contains the number of frames specified in the `FramesPerTrigger` property of the video input object `obj`. `obj` must be a 1-by-1 video input object.

`data` is returned as an H-by-W-by-B-by-F matrix where

H   Image height, as specified in the object's `ROIPosition` property

W   Image width, as specified in the object's `ROIPosition` property

B   Number of bands, as specified in the `NumberOfBands` property.

F   The number of frames returned

`data` is returned to the MATLAB workspace in its native data type using the color space specified by the `ReturnedColorSpace` property.

You can use the MATLAB `image` or `imagesc` functions to view the returned data. Use `imaqmontage` to view multiple frames at once.

`data = getdata(obj,n)` returns `n` frames of data associated with the video input object `obj`.

`data = getdata(obj,n,type)` returns `n` frames of data associated with the video input object `obj`, where `type` specifies the data type used to store the returned data. `type` can be any of these strings:

Type String	Data Type
`'uint8'`	Unsigned 8-bit integer
`'uint16'`	Unsigned 16-bit integer

getdata

Type String	Data Type
'uint32'	Unsigned 32-bit integer
'double'	Double precision
'native'	Uses native data type. This is the default.

If type is not specified, 'native' is used as the default. If there is no MATLAB data type that matches the object's native data type, getdata chooses a MATLAB data type that preserves numerical accuracy. For example, the components of 12-bit RGB color data would each be returned as uint8 data.

data = getdata(obj,n,type,format) returns n frames of data associated with the video input object obj, where format specifies the MATLAB format of data. format can be either of these strings:

Format String	Description
'numeric'	Returns data as an H-by-W-by-B-by-F array
'cell'	Returns data as an F-by-1 cell array of H-by-W-by-B matrices

[data,time] = getdata(...) returns time, an F-by-1 matrix, where F is the number of frames returned in data. Each element of time indicates the relative time, in seconds, of the corresponding frame in data, relative to the first trigger.

time = 0 is defined as the point at which data logging begins. When data logging begins, the object's Logging property is set to 'On'. time is measured continuously with respect to 0 until the acquisition stops. When the acquisition stops, the object's Running property is set to 'Off'.

[data, time, metadata] = getdata(...) returns metadata, an F-by-1 array of structures, where F is the number of frames returned in data. Each

structure contains information about the corresponding frame in `data`. The `metadata` structure contains these fields:

Metadata Field	Description
`'AbsTime'`	Absolute time the frame was acquired, expressed as a time vector
`'FrameNumber'`	Number identifying the nth frame since the `start` command was issued
`'RelativeFrame'`	Number identifying the nth frame relative to the start of a trigger
`'TriggerIndex'`	Number of the trigger in which this frame was acquired

`getdata` is a blocking function that returns execution control to the MATLAB workspace after the requested number of frames becomes available within the time period specified by the object's `Timeout` property. The object's `FramesAvailable` property is automatically reduced by the number of frames returned by `getdata`. If the requested number of frames is greater than the frames to be acquired, `getdata` returns an error.

It is possible to issue a Ctrl-C while `getdata` is blocking. This does not stop the acquisition but does return control to MATLAB.

Example   Construct a video input object associated with a Matrox device at ID 1.

```
obj = videoinput('matrox', 1);
```

Initiate an acquisition and access the logged data.

```
start(obj);
data = getdata(obj);
```

Display each image frame acquired.

```
imaqmontage(data);
```

Remove the video input object from memory.

```
delete(obj);
```

See Also   getsnapshot, imaqhelp, imaqmontage, peekdata, propinfo

Purpose   Return the currently selected video source object

Syntax   src = getselectedsource(obj)

Description   src = getselectedsource(obj) searches all the video source objects
                   associated with the video input object obj and returns the video source object,
                   src, that has the Selected property value set to 'on'.

                   To select a source for acquisition, use the SelectedSourceName property of the
                   video input object.

                   obj must be a 1-by-1 video input object.

See Also   imaqhelp, get, videoinput

getsnapshot

Purpose	Immediately return a single image frame
Syntax	frame = getsnapshot(obj)

Description   frame = getsnapshot(obj) immediately returns one single image frame, frame, from the video input object obj. The frame of data returned is independent of the video input object FramesPerTrigger property and has no effect on the value of the FramesAvailable or FramesAcquired property.

The object obj must be a 1-by-1 video input object.

frame is returned as an H-by-W-by-B matrix where

H	Image height, as specified in the ROIPosition property
W	Image width, as specified in the ROIPosition property
B	Number of bands associated with obj, as specified in the NumberOfBands property

frame is returned to the MATLAB workspace in its native data type using the color space specified by the ReturnedColorSpace property.

You can use the MATLAB image or imagesc function to view the returned data.

Note If obj is running but not logging, and has been configured with a hardware trigger, a timeout error will occur.

To interrupt the getsnapshot function and return control to the MATLAB command line, issue the ^C (Control-C) command.

Example   Create a video input object.

```
obj = videoinput('matrox', 1);
```

Acquire and display a single frame of data.

```
frame = getsnapshot(obj);
image(frame);
```

Remove the video input object from memory.

```
delete(obj);
```

See Also   getdata, imaqhelp, peekdata

imaqfind

Purpose   Find image acquisition objects

Syntax   imaqfind
                   out = imaqfind
                   out = imaqfind('P1', V1, 'P2', V2,...)
                   out = imaqfind(S)
                   out = imaqfind(obj, 'P1', V1, 'P2', V2,...)

Description   imaqfind returns an array containing all the video input objects that exist in
                   memory. If only a single video input object exists in memory, imaqfind displays
                   a detailed summary of that object.

                   out = imaqfind returns an array, out, of all the video input objects that exist
                   in memory.

                   out = imaqfind('P1', V1, 'P2', V2,...) returns a cell array, out, of
                   image acquisition objects whose property names and property values match
                   those passed as parameter/value pairs P1, V1, P2, V2. Parameter/value pairs
                   can be specified as a cell array.

                   out = imaqfind(S) returns a cell array, out, of image acquisition objects
                   whose property values match those defined in the structure S. The field names
                   of S are image acquisition object property names and the field values are the
                   requested property values.

                   out = imaqfind(obj, 'P1', V1, 'P2', V2,...) restricts the search for
                   matching parameter/value pairs to the image acquisition objects listed in obj.
                   obj can be an array of image acquisition objects.

                   Note Parameter name/parameter value string pairs, structures, and
                   parameter name/parameter value cell array pairs can be used in the same call
                   to imaqfind.

                   When searching for properties with specific values, imaqfind performs
                   case-sensitive searches. For example, if the value of an object's Name property
                   is 'MyObject', imaqfind does not find a match if you specify 'myobject'. Use
                   the get function to determine the exact spelling of a property value. Note,

however, that searches for properties that have an enumerated list of possible values are not case sensitive. For example, imaqfind will find an object with a Running property value of 'Off' or 'off'.

Example

This example uses imaqfind to find image acquisition objects with the specified property values.

```
obj1 = videoinput('matrox', 1, 'Tag', 'FrameGrabber');

obj2 = videoinput('winvideo', 1, 'Tag', 'Webcam');

out1 = imaqfind('Type', 'videoinput')

out2 = imaqfind('Tag', 'FrameGrabber')

out3 = imaqfind({'Type', 'Tag'}, {'videoinput', 'Webcam'})
```

See Also

get, videoinput

imaqhelp

Purpose   Return image acquisition object function and property help

Syntax
```
imaqhelp
imaqhelp('name')
imaqhelp(obj)
imaqhelp(obj,'name')
out = imaqhelp(...)
```

Description   `imaqhelp` provides a complete listing of image acquisition object functions.

`imaqhelp('name')` provides online help for the function or property `name`.

`imaqhelp(obj)` displays a listing of functions and properties for the image acquisition object `obj` along with the online help for the object's constructor. `obj` must be a 1-by-1 image acquisition object.

`imaqhelp(obj,'name')` displays the help for function or property `name` for the image acquisition object `obj`.

If `name` is a device-specific property name, `obj` must be provided.

`out = imaqhelp(...)` returns the help text in string `out`.

When property help is displayed, the names in the "See also" section that contain all uppercase letters are function names. The names that contain a mixture of upper- and lowercase letters are property names.

When function help is displayed, the "See also" section contains only function names.

Example   Getting general function and property help.

```
imaqhelp('videoinput')

out = imaqhelp('videoinput.m');

imaqhelp set

imaqhelp LoggingMode
```

Getting property help with device-specific information.

```
vid = videoinput('dt', 1);

src = getselectedsource(vid);

imaqhelp(vid, 'TriggerType')

imaqhelp(src, 'FrameRate')
```

See Also   propinfo

imaqhwinfo

Purpose   Return information about available image acquisition hardware

Syntax
```
out = imaqhwinfo
out = imaqhwinfo('adaptorname')
out = imaqhwinfo('adaptorname','field')
out = imaqhwinfo('adaptorname',deviceID)
out = imaqhwinfo(obj)
out = imaqhwinfo(obj,'field')
```

Description   `out = imaqhwinfo` returns a structure, `out`, that contains information about the image acquisition adaptors available on the system. An adaptor is the interface between MATLAB and the image acquisition devices connected to the system. The adaptor's main purpose is to pass information between MATLAB and an image acquisition device via its driver.

`out = imaqhwinfo('adaptorname')` returns a structure `out` that contains information about the specified adaptor. The information returned includes adaptor version and available hardware for the specified adaptor. To get a list of valid adaptor names, use the `imaqhwinfo` syntax.

`out = imaqhwinfo('adaptorname','field')` returns the value of the specified field, `field`, for the specified adaptor. The argument `field` can be a single string or a cell array of strings. If `field` is a cell array, `out` is a 1-by-N cell array where N is the length of `field`. To get a list of valid field names, use the `imaqhwinfo('adaptorname')` syntax.

`out = imaqhwinfo('adaptorname', deviceID)` returns information for the device identified by the numerical device ID `deviceID`. The `deviceID` can be a scalar or a vector. If `deviceID` is a vector, `out` is a 1-by-N structure array where N is the length of `deviceID`.

`out = imaqhwinfo(obj)` returns a structure `out` that contains information about the specified image acquisition object `obj`. If `obj` is an array of device objects, then `out` is a 1-by-n cell array of structures where n is the length of `obj`.

out = imaqhwinfo(obj,'field') returns the hardware information for the specified field, field, for the device object obj. field can be a single field name or a cell array of field names. out is an m-by-n cell array where m is the length of obj and n is the length of field. You can return a list of valid field names with the imaqhwinfo(obj) syntax.

Note After you call imaqhwinfo once, hardware information is cached by the toolbox. To force the toolbox to search for new hardware that might have been installed while MATLAB was running, use imaqreset.

Example

This example returns information about all the adaptors available on the system.

```
imaqhwinfo

ans =

InstalledAdaptors: {'matrox'  'winvideo'}
      MATLABVersion: '6.5.1 (R13+)'
        ToolboxName: 'Image Acquisition Toolbox'
     ToolboxVersion: '1.0 (R13+)
```

This example returns information about all the devices accessible through a particular adaptor.

```
info = imaqhwinfo('winvideo')
info =

       AdaptorDllName: [1x73 char]
    AdaptorDllVersion: '1.0 (R13+)'
          AdaptorName: 'winvideo'
            DeviceIDs: {[1]}
           DeviceInfo: [1x1 struct]
```

This example returns information about a specific device accessible through a particular adaptor. You identify the device by its device ID.

```
dev_info = imaqhwinfo('winvideo', 1)

dev_info =

            DefaultFormat: 'RGB555_128x96'
      DeviceFileSupported: 0
               DeviceName: 'IBM PC Camera'
                 DeviceID: 1
         ObjectConstructor: 'videoinput('winvideo', 1)'
          SupportedFormats: {1x34 cell}
```

This example gets information about the device associated with a particular video input object.

```
obj = videoinput('winvideo', 1);

obj_info = imaqhwinfo(obj)

obj_info =

                      AdaptorName: 'winvideo'
                       DeviceName: 'IBM PC Camera'
                        MaxHeight: 96
                         MaxWidth: 128
                     TotalSources: 1
         VendorDriverDescription: 'Windows WDM Compatible Driver'
             VendorDriverVersion: 'DirectX 9.0'
```

This example returns the value of a particular field in the device information associated with a particular video input object.

```
field_info = imaqhwinfo(vid,'adaptorname')

field_info =

winvideo
```

See Also   imaqhelp, imaqreset

Purpose   Limit memory or display memory usage for the Image Acquisition Toolbox

Syntax   mem = imaqmem
                 imaqmem(limit)

Description mem = imaqmem returns a structure out containing the following fields:

Field	Description
MemoryLoad	Number between 0 and 100 that gives a general idea of current memory utilization
TotalPhys	Total number of bytes of physical memory
AvailPhys	Number of bytes of physical memory currently available
TotalPageFile	Total number of bytes that can be stored in the paging file
AvailPageFile	Number of bytes available in the paging file
TotalVirtual	Total number of bytes that can be described in the user mode portion of the virtual address space
AvailVirtual	Number of bytes of unreserved and uncommitted memory in the user mode portion of the virtual address space
FrameMemoryLimit	Total number of bytes image acquisition frames can occupy in memory By default, the toolbox sets this limit to equal all available physical memory at the time imaqmem is first used or queried.
FrameMemoryUsed	Number of bytes currently allocated by the Image Acquisition Toolbox

imaqmem

imaqmem(limit) configures the frame memory limit, in bytes, for the Image Acquisition Toolbox. value is used to determine the maximum amount of memory the toolbox can use for logging image frames.

Note Configuring this limit does not remove any logged frames from the image acquisition memory buffer. To remove frames from the buffer, you can bring them into the MATLAB workspace, using the getdata function, or remove them from memory, using the flushdata function.

See Also   flushdata, getdata, videoinput

Purpose   Display a sequence of image frames as a montage

Syntax   imaqmontage(frames)
                   imaqmontage(obj)
                   imaqmontage(...,CLIM)

Description   imaqmontage(frames) displays a montage of image frames in a MATLAB
                   figure window using the imagesc function.

                   frames can be any data set returned by getdata, peekdata, or getsnapshot.

                   imaqmontage(obj) calls the getsnapshot function on video input object obj
                   and displays a single image frame in a MATLAB figure window using the
                   imagesc function. obj must be a 1-by-1 video input object.

                   imaqmontage(...,CLIM) displays a montage of image frames, where CLIM is a
                   two-element vector, [CLOW CHIGH], specifying the image scaling. Use CLIM to
                   specify a scaling value when overscaling the image data is a risk, for example,
                   when you are working with devices that provide data in a 12-bit format.

See Also   getdata, getsnapshot, imaqhelp, peekdata

imaqreset

Purpose   Disconnect and delete all image acquisition objects

Syntax   `imaqreset`

Description `imaqreset` deletes any image acquisition objects that exist in memory and unloads all adaptors loaded by the toolbox. As a result, the image acquisition hardware is reset.

`imaqreset` is the image acquisition command that returns MATLAB to the known state of having no image acquisition objects and no loaded image acquisition adaptors.

`imaqreset` can be used to force the toolbox to search for new hardware that might have been installed while MATLAB was running.

See Also   `delete`, `videoinput`

Purpose   Determine if image acquisition object is associated with an image acquisition device

Syntax   `out = isvalid(obj)`

Description   `out = isvalid(obj)` returns a logical array `out` that contains a 1 where the elements of `obj` are image acquisition objects associated with hardware and a 0 where the elements of `obj` are image acquisition objects not associated with hardware.

An object is an invalid image acquisition object when it is no longer associated with any hardware. If this is the case, `obj` should be cleared from the workspace.

See Also   `delete`, `imaqfind`, `videoinput`

load

Purpose   Load an image acquisition object into the MATLAB workspace

Syntax
```
load filename
load filename obj1 obj2 ...
S = load('filename','obj1','obj2',...)
```

Description   `load filename` returns all variables from the MAT-file `filename` to the MATLAB workspace.

`load filename obj1 obj2 ...` returns the specified image acquisition objects (`obj1`, `obj2`, etc.) from the MAT-file `filename` to the MATLAB workspace.

`S = load('filename','obj1','obj2',...)` returns the structure `S` with the specified image acquisition objects (`obj1`, `obj2`, etc.) from the MAT-file `filename` instead of directly loading the image acquisition objects into the workspace. The field names in `S` match the names of the image acquisition objects that were retrieved. If no objects are specified, then all variables existing in the MAT-file are loaded.

Values for read-only properties are restored to their default values upon loading. For example, the Running property is restored to `'off'`. Use `propinfo` to determine if a property is read only.

Examples
```
obj = videoinput('winvideo', 1);
set(obj, 'SelectedSourceName', 'input1')
save fname obj
load fname
load('fname', 'obj');
```

See Also   `imaqhelp`, `propinfo`, `save`

Purpose　　　　Return most recently acquired image data

Syntax　　　　`data = peekdata(obj,frames)`

Description　　`data = peekdata(obj,frames)` returns `data` containing the latest number of frames specified by `frames`. If `frames` is greater than the number of frames currently acquired, all available frames are returned with a warning message stating that the requested number of frames was not available. `obj` must be a 1-by-1 video input object.

`data` is returned as an H-by-W-by-B-by-F matrix where

H	Image height, as specified in the object's `ROIPosition` property
W	Image width, as specified in the object's `ROIPosition` property
B	Number of color bands, as specified in the `NumberOfBands` property
F	Number of frames returned

`data` is returned to the MATLAB workspace in its native data type using the color space specified by the `ReturnedColorSpace` property.

You can use the MATLAB `image` or `imagesc` functions to view the returned data. Use `imaqmontage` to view multiple frames at once.

`peekdata` is a nonblocking function that immediately returns image frames and execution control to the MATLAB workspace. Not all requested data might be returned.

Note `peekdata` provides a look at the data; it does not remove data from the memory buffer. The object's `FramesAvailable` property value is not affected by the number of frames returned by `peekdata`.

The behavior of `peekdata` depends on the settings of the `Running` and the `Logging` properties.

peekdata

Running	Logging	Object State	Result
On	Off	The object has been started but is waiting for a trigger. (TriggerType is set to 'manual' or 'hardware'). No data has been acquired so none is available.	peekdata returns a single frame of data and issues a warning, if you requested more than one frame.
On	On	The object has been started, a trigger has executed, and the object is actively acquiring data.	peekdata returns the n most recently acquired frames of data. The frames are not removed from the buffer.
Off	Off	The object has stopped running because it acquired the requested number of frames or you called the stop function.	peekdata can be called once to return the n most recently acquired frames of data, assuming FramesAvailable is greater than 0. Otherwise, peekdata returns an error. The frames returned are not removed from the memory buffer.

The number of frames available to peekdata is determined by recalling the last frame returned by a previous peekdata call, and the number of frames that were acquired since then.

peekdata can be used only after the start command is issued and while the object is running. peekdata can also be called once after obj has stopped running.

See Also   getdata, getsnapshot, imaqhelp, imaqmontage, propinfo, start

Purpose   Activate a live image preview window

Syntax   `preview(obj)`

Description   `preview(obj)` immediately activates a live image preview window for the video input object `obj`. The size of the previewed images reflects the `ROIPosition` property configuration. The preview window remains active until it is closed with the `closepreview` function.

The behavior of the preview window depends on the object's current state and trigger configuration.

While an object is stopped (`Running ='off'`), the preview window shows a live view of the image being acquired from the device, for all trigger types. The image is updated to reflect changes made to configurations of object properties. (The `FrameGrabInterval` property is ignored until a trigger occurs.)

When an object is started (`Running = 'on'`), the behavior of the preview window depends on the trigger type. If the trigger type is set to immediate or manual, the preview window continues to update the image displayed. If the trigger type is set to hardware, the preview window stops updating the image displayed until a trigger occurs.

While an object is logging (`Logging = 'on'`), the preview window might drop some data frames, but this does not affect the frames logged to memory or disk.

If you delete `obj` with the `delete` function, any associated preview windows are closed.

See Also   `closepreview`, `videoinput`

propinfo

Purpose   Return property characteristics for image acquisition objects

Syntax   ```
out = propinfo(obj)
out = propinfo(obj,'PropertyName')
```

**Description**    out = propinfo(obj) returns the structure out whose field names are the
names of all the properties supported by obj. obj must be a 1-by-1 image
acquisition object. The value of each field is a structure containing the fields
shown below.

| Field Name | Description |
|---|---|
| Type | Data type of the property. Possible values are 'any', 'callback','double', 'string', and 'struct' |
| Constraint | Type of constraint on the property value. Possible values are 'bounded', 'callback', 'enum', and 'none' |
| ConstraintValue | List of valid string values or a range of valid values |
| DefaultValue | Default value for the property |
| ReadOnly | Condition under which a property is read only:<br><br>• 'always'— Property cannot be configured.<br>• 'whileRunning'— Property cannot be configured while Running is set to on.<br>• 'never'— Property can be configured at any time. |
| DeviceSpecific | 1 if the property is device specific; otherwise, 0 (zero) |

out = propinfo(obj,'PropertyName') returns the structure out for the
property specified by PropertyName. If PropertyName is a cell array of strings,
propinfo returns a structure for each property, stored in a cell array.

**Example**        Create the video input object vid.

```
vid = videoinput('winvideo',1);
```

Capture all property information for all properties.

```
out = propinfo(vid);
```

Access property information for a particular property.

```
out1 = propinfo(vid,'LoggingMode');
```

See Also   imaqhelp

save

Purpose

Save image acquisition objects to a MAT-file

Syntax

```
save filename
save filename obj1 obj2 ...
save('filename','obj1','obj2',...)
```

Description

save filename saves all variables in the MATLAB workspace to the MAT-file filename. If filename does not include a file extension, save appends the .MAT extension to the filename.

save filename obj1 obj2 ... saves the specified image acquisition objects (obj1, obj2, etc.) to the MAT-file filename.

save('filename','obj1','obj2',...) is the functional form of the command, where the file name and image acquisition objects must be specified as strings. If no objects are specified, then all variables existing in the MATLAB workspace are saved.

Note Any data associated with the image acquisition object is not stored in the MAT-file. To save the data, bring it into the MATLAB workspace (using the getdata function), and then save the variable to the MAT-file.

To return variables from the MAT-file to the MATLAB workspace, use the load command. Values for read-only properties are restored to their default values upon loading. For example, the Running property is restored to 'off'. Use the propinfo function to determine if a property is read only.

Examples

```
obj = videoinput('winvideo', 1);

set(obj, 'SelectedSourceName', 'input1')

save fname obj

set(obj, 'TriggerFcn', {'mycallback', 5});

save('fname1', 'obj')
```

See Also

imaqhelp, load, propinfo

Purpose   Configure or display image acquisition object properties

Syntax   set(obj)
                  prop_struct = set(obj)
                  set(obj,'PropertyName')
                  prop_cell = set(obj,'PropertyName')
                  set(obj,'PropertyName',PropertyValue,...)
                  set(obj,S)
                  set(obj,PN,PV)

Description   set(obj) displays property names and any enumerated values for all
                 configurable properties of image acquisition object obj. obj must be a single
                 image acquisition object.

                 prop_struct = set(obj) returns the property names and any enumerated
                 values for all configurable properties of image acquisition object obj. obj must
                 be a single image acquisition object. The return value prop_struct is a
                 structure whose field names are the property names of obj, and whose values
                 are cell arrays of possible property values or empty cell arrays if the property
                 does not have a finite set of possible string values.

                 set(obj,'PropertyName') displays the possible values for the specified
                 property, PropertyName, of image acquisition object obj. obj must be a single
                 image acquisition object.

                 prop_cell = set(obj,'PropertyName') returns the possible values for the
                 specified property, PropertyName, of image acquisition object obj. obj must be
                 a single image acquisition object. The returned array prop_cell is a cell array
                 of possible value strings or an empty cell array if the property does not have a
                 finite set of possible string values.

                 set(obj,'PropertyName',PropertyValue,...) configures the property
                 PropertyName to the specified value PropertyValue for image acquisition
                 object obj. You can specify multiple property name/property value pairs in a
                 single statement. obj can be a single image acquisition object or a vector of
                 image acquisition objects, in which case set configures the property values for
                 all the image acquisition objects specified.

set(obj,S) configures the properties of obj with the values specified in S, where S is a structure whose field names are object property names.

set(obj,PN,PV) configures the properties specified in the cell array of strings, PN, to the corresponding values in the cell array PV, for the image acquisition object obj. PN must be a vector. If obj is an array of image acquisition objects, PV can be an M-by-N cell array, where M is equal to the length of the image acquisition object array and N is equal to the length of PN. In this case, each image acquisition object is updated with a different set of values for the list of property names contained in PN.

Note Parameter/value string pairs, structures, and parameter/value cell array pairs can be used in the same call to set.

Example
These examples illustrate the various ways to use the set function to set the values of image acquisition object properties.

```
set(obj, 'FramesPerTrigger', 15, 'LoggingMode', 'disk');

set(obj, {'TimerFcn', 'TimerPeriod'}, {@imaqcallback, 25});

set(obj, 'Name', 'MyObject');

set(obj, 'SelectedSourceName')
```

See Also
get, imaqfind, videoinput

Purpose   Obtain exclusive use of an image acquisition device

Syntax   start(obj)

Description start(obj) obtains exclusive use of the image acquisition device associated
with the video input object obj and locks the device's configuration. Starting an
object is a necessary first step to acquire image data, but it does not control
when data is logged.

obj can either be a 1-by-1 video input object or an array of video input objects.

Data logging is controlled with the TriggerType property.

Trigger Type	Logging Behavior
'hardware'	Data logging occurs when the condition specified in the object's TriggerCondition property is met via the TriggerSource.
'immediate'	Data logging occurs immediately.
'manual'	Data logging occurs when the trigger function is called.

Use the triggerconfig function to configure the object's trigger settings.

When an acquisition is started, obj performs the following operations:

1 Transfers the object's configuration to the associated hardware
2 Executes the object's StartFcn callback
3 Sets the object's Running property to 'On'

If the object's StartFcn errors, the hardware is never started and the object's
Running property remains 'Off'.

The start event is recorded in the object's EventLog property.

An image acquisition object stops running when one of the following conditions is met:

- The stop function is issued.
- The requested number of frames is acquired. This occurs when

  FramesAcquired = FramesPerTrigger * (TriggerRepeat + 1)

  where FramesAcquired, FramesPerTrigger, and TriggerRepeat are properties of the video input object.
- A run-time error occurs.
- The object's Timeout value is reached.

Example   The start function can be called by a video input object's event callback.

```
obj.StopFcn = {'start'};
```

See Also   imaqfind, imaqhelp, propinfo, stop, trigger, triggerconfig

Purpose   Stop video input object running and logging

Syntax   stop(obj)

Description stop(obj) halts an acquisition associated with the video input object obj. obj
                 can be either a single video input object or an array of video input objects.

                 The stop function

                 • Sets the object's Running property to 'Off'
                 • Sets the object's Logging property to 'Off', if needed
                 • Executes the object's StopFcn callback

                 An image acquisition object can also stop running under one of the following
                 conditions:

                 • The requested number of frames is acquired. This occurs when

                   FramesAcquired = FramesPerTrigger * (TriggerRepeat + 1)

                   where FramesAcquired, FramesPerTrigger, and TriggerRepeat are
                   properties of the video input object.
                 • A run-time error occurs.
                 • The object's Timeout value is reached.

                 The stop event is recorded in the object's EventLog property.

Example   The stop function can be called by a video input object's event callback.

                     obj.TimerFcn = {'stop'};

See Also   imaqfind, start, trigger, propinfo, videoinput

trigger

Purpose   Initiate data logging

Syntax   `trigger(obj)`

Description   `trigger(obj)` initiates data logging for the video input object `obj`. `obj` can be either a single video input object or an array of video input objects.

The `trigger` function

- Executes the object's `TriggerFcn` callback
- Records the absolute time of the first trigger event in the object's `InitialTriggerTime` property
- Configures the object's `Logging` property to `'On'`

`obj` must be running and its `TriggerType` property must be set to `'manual'`. To start an object running, use the `start` function.

The trigger event is recorded in the object's `EventLog` property.

Example   The `trigger` function can be called by a video input object's event callback.

```
obj.StartFcn = @trigger;
```

See Also   `imaqfind`, `start`, `stop`, `videoinput`

Purpose   Configure video input object trigger properties

Syntax   triggerconfig(obj,'type')
                   triggerconfig(obj,'type','condition')
                   triggerconfig(obj,'type','condition','source')
                   config = triggerconfig(obj)
                   triggerconfig(obj,config)

Description   triggerconfig(obj,'type')
                   triggerconfig(obj,'type','condition')
                   triggerconfig(obj,'type','condition','source') configures the values of
                   the TriggerType, TriggerCondition, and TriggerSource properties of video
                   input object obj to the values specified by type, condition, and source,
                   respectively.

                   obj can be either a single video input object or an array of video input objects.
                   If an error occurs, any video input objects in the array that have already been
                   configured are returned to their original configurations.

                   type, condition, and source are text strings. For a list of valid trigger
                   configurations, use triggerinfo(obj). condition and source are optional
                   parameters as long as a unique trigger configuration can be determined from
                   the parameters provided.

                   config = triggerconfig(obj) returns a MATLAB structure config
                   containing the object's current trigger configuration. obj must be a 1-by-1 video
                   input object. The field names of config are TriggerType, TriggerCondition,
                   and TriggerSource. Each field contains the current value of the object's
                   property.

                   triggerconfig(obj,config) configures the TriggerType, TriggerCondition,
                   and TriggerSource property values for video input object obj using config, a
                   MATLAB structure with field names TriggerType, TriggerCondition, and
                   TriggerSource, each containing the desired property value.

Examples   **Example 1**
                   Construct a video input object.

                       vid = videoinput('winvideo', 1);

triggerconfig

Configure trigger properties for the object.

```
triggerconfig(vid, 'manual')
```

Trigger the acquisition.

```
start(obj)
trigger(obj)
```

Remove video input object from memory.

```
delete(vid);
```

Example 2

This example uses a structure returned from `triggerinfo` to configure trigger parameters.

Create a video input object.

```
vid = videoinput('winvideo', 1);
```

Use `triggerinfo` to get all valid configurations for the trigger properties for the object.

```
config = triggerinfo(vid);
```

Pass one of the configurations to the `triggerconfig` function.

```
triggerconfig(vid,config(2));
```

Remove video input object from memory.

```
delete(vid);
```

See Also   imaqhelp, trigger, triggerinfo, videoinput

Purpose   Provide information on available trigger configurations

Syntax   ```
triggerinfo(obj)
triggerinfo(obj,type)
config = triggerinfo(...)
```

Description   `triggerinfo(obj)` displays all available trigger configurations for video input object `obj`. `obj` can only be a 1-by-1 video input object.

triggerinfo(obj,type) displays the available trigger configurations for the specified `TriggerType`, `type`, for video input object `obj`.

`config = triggerinfo(...)` returns `config`, an array of MATLAB structures, containing all the valid trigger configurations for the video input object `obj`. Each structure in the array contains these fields:

Field	Description
`TriggerType`	Name of the trigger type
`TriggerCondition`	Condition that must be met before executing a trigger
`TriggerSource`	Hardware source used for triggering

You can pass one of the structures in `config` to the `triggerconfig` function to specify the trigger configuration.

Example   This example illustrates how to use the `triggerinfo` function to retrieve valid configurations of the `TriggerType`, `TriggerSource`, and `TriggerCondition` properties.

```
vid = videoinput('winvideo');

config = triggerinfo(vid)

config =
```

triggerinfo

```
1x2 struct array with fields:
    TriggerType
    TriggerCondition
    TriggerSource

config(1)

ans =

        TriggerType: 'immediate'
   TriggerCondition: 'none'
      TriggerSource: 'none'
```

See Also   imaqhelp, triggerconfig

Purpose   Create a video input object

Syntax
```
obj = videoinput('adaptorname')
obj = videoinput('adaptorname', deviceID)
obj = videoinput('adaptorname', deviceID, 'format')
obj = videoinput('adaptorname', deviceID, 'devicefilename')
obj = videoinput('adaptorname', deviceID, 'format','P1',
  V1,'P2',V2,...)
obj = videoinput('adaptorname', deviceID, 'devicefilename','P1',
  V1,'P2',V2,...)
```

Description
```
obj = videoinput('adaptorname')
obj = videoinput('adaptorname',deviceID)
obj = videoinput('adaptorname',deviceID,'format') constructs a video
```
input object `obj`, where

`'adaptorname'`	String specifying the device adaptor `obj` is associated with. Valid values can be determined by using the `imaqhwinfo` function.
`'deviceID'`	Numerical device identifier. If `deviceID` is not specified, the first available device ID is used.
`'format'`	String specifying the video format for `obj`. If `format` is not specified, the device's default format is used.

Valid `deviceID` and `format` values, as well as the default format, can be determined by using the `imaqhwinfo('adaptorname')` syntax.

Upon creation, the object's `VideoFormat` property reflects the specified `format`. The toolbox chooses the first available video source object as the selected source and specifies this video source object's name in the object's `SelectedSourceName` property. Use `getselectedsource(obj)` to access the video source object that is used for acquisition.

As a convenience, a device's name can be used in place of the `deviceID`. If multiple devices have the same name, the first available device is used.

`obj = videoinput('adaptorname',deviceID,'devicefilename')` constructs a video input object using a device configuration file, `devicefilename`, as the

videoinput

video format. The object's `VideoFormat` property reflects the `devicefilename` and its path.

```
obj = videoinput('adaptorname',deviceID,'format','P1',V1,'P2',
V2,...)
obj = videoinput('adaptorname',deviceID,'devicefilename','P1',V1,
'P2',V2,...)
```
creates a video input object `obj` with the specified property values. If an invalid property name or property value is specified the object is not created.

The property value pairs can be in any format supported by the `set` function, i.e., parameter/value string pairs, structures, or parameter/value cell array pairs.

To view a complete listing of video input object functions and properties, use the `imaqhelp` function.

```
imaqhelp videoinput
```

Example   Construct a video input object.

```
obj = videoinput('matrox', 1);
```

Select the source to use for acquisition.

```
set(obj, 'SelectedSourceName', 'input1')
```

View the properties for the selected video source object.

```
src_obj = getselectedsource(obj);
get(src_obj)
```

Preview a stream of image frames.

```
preview(obj);
```

Acquire and display a single image frame.

```
frame = getsnapshot(obj);
image(frame);
```

Remove video input object from memory.

```
delete(obj);
```

See Also   delete, imaqfind, isvalid, preview

wait

Purpose   Wait for the image acquisition object to stop running

Syntax   wait(obj)
                 wait(obj,waittime)

Description wait(obj) blocks the MATLAB command line until the video input object obj
                 stops running (Running = 'off'). obj can be either a single video input object
                 or an array of video input objects. When obj is an array of objects, the wait
                 function waits until all objects in the array stop running. If the object is not
                 running or is an invalid object, wait returns immediately.

                 wait(obj,waittime) blocks the MATLAB command line until the video input
                 object obj stops running or until waittime seconds have expired, whichever
                 comes first.

                 Note The video input object's stop event callback function (StopFcn) might
                 not be called before this function returns.

                 An image acquisition object stops running when one of the following conditions
                 is met:

                 • The stop function is issued.
                 • The requested number of frames is acquired. This occurs when

                   FramesAcquired = FramesPerTrigger * (TriggerRepeat + 1)

                   where FramesAcquired, FramesPerTrigger, and TriggerRepeat are
                   properties of the video input object.
                 • A run-time error occurs.
                 • The object's Timeout value is reached.

Example   Create a video input object.

                   vid = videoinput('winvideo');

                 Specify an acquisition that should take several seconds. The example sets the
                 FramesPerTrigger property to 300.

                   vid.FramesPerTrigger = 300;

Start the object. Because it is configured with an immediate trigger (the default), acquisition begins when the object is started. The example calls the wait function after calling the start function. Notice how wait blocks the MATLAB command line until the acquisition is complete.

```
start(vid), wait(vid);
```

See Also   imaqhelp, start, stop, trigger, propinfo

wait

Property Reference

This chapter describes all the properties of video input objects and the properties of the video source object that are common to all video source objects. Video source objects can also support device-specific properties that vary depending on the image acquisition hardware. To get help on these device-specific properties, use the `imaqhelp` function, specifying the video source object as an argument.

Properties – By Category

This section contains brief descriptions of all properties supported by the image acquisition objects. The descriptions are organized by image acquisition object type:

- Video input object properties
- Video source object properties

Video Input Object Properties

The following tables list all the properties of the video input object in several categories

- General properties
- Callback function properties
- Trigger properties
- Acquisition source properties

General Properties

DeviceID	Identify the image acquisition device represented by the video input object
DiskLogger	Specify the MATLAB AVIFILE object for logging data to disk
DiskLoggerFrameCount	Indicate the number of frames written to disk
EventLog	Specify a structure array storing information related to specific events
FrameGrabInterval	Specify how often to acquire a frame from the video stream
FramesAcquired	Indicate the total number of frames acquired
FramesAvailable	Indicate the number of frames available in the memory buffer

General Properties (Continued)

FramesPerTrigger	Specify the number of frames to acquire using the selected video source
Logging	Indicate whether data is currently being logged to memory, disk, or both
LoggingMode	Specify the destination for acquired data
Name	Specify a descriptive name for the device object
NumberOfBands	Specify the number of color bands in the data to be acquired
Previewing	Indicate whether data is being previewed in a window
ReturnedColorSpace	Specify the color space used to return acquired data to MATLAB
ROIPosition	Specify the region-of-interest acquisition window
Running	Indicate whether the video input object is ready for acquisition
Tag	Specify a label to associate with an image acquisition object
Timeout	Specify an additional waiting time to extract data
Type	Indicate the class type of the image acquisition object
UserData	Specify data to associate with an image acquisition object

General Properties (Continued)

VideoFormat	Indicate the video input format for the image acquisition device
VideoResolution	Indicate the width and height of the incoming video stream

Callback Properties	
ErrorFcn	Specify the M-file executed when a run-time error occurs
FramesAcquiredFcn	Specify the M-file executed when a frames acquired event occurs
FramesAcquiredFcnCount	Specify the number of frames to acquire from the selected video source before a frames acquired event is generated
StartFcn	Specify the M-file executed before a video input object starts running
StopFcn	Specify the M-file executed when a video input object stops running
TimerFcn	Specify the M-file executed when a time period expires
TimerPeriod	Specify the period of time between timer events
TriggerFcn	Specify the M-file executed when a trigger occurs

Trigger Properties	
`InitialTriggerTime`	Indicate the absolute time of the first trigger
`TriggerCondition`	Specify the condition on which to execute a hardware trigger
`TriggerFrameDelay`	Specify the number of frames to delay before logging image frames
`TriggerRepeat`	Specify the number of additional times to execute the trigger
`TriggersExecuted`	Indicate the number of triggers that have been executed
`TriggerSource`	Specify the source to use for executing a hardware trigger
`TriggerType`	Specify the type of trigger to execute

Acquisition Source Properties	
`SelectedSourceName`	Name of video source object used for acquisition
`Source`	Array of video source objects associated with a video input object

Video Source Object Properties

Video input objects create one or more video source objects that represent the image acquisition data sources. The following table lists the properties common to all video source objects.

Note A video source object can support additional, device-specific properties. These properties vary, depending on the image acquisition hardware. To get information about these properties, use the `imaqhelp` function, specifying the video source object as an argument.

Video Source Object Properties

Parent	Indicate the parent of the video source object
Selected	Indicate whether the video source object is currently selected
SourceName	Indicate the name of a video source object
Tag	Specify a label to associate with an image acquisition object
Type	Indicate the class type of the image acquisition object

Purpose   Identify the image acquisition device represented by the video input object

Description The DeviceID property identifies the device represented by the video input object.

A device ID is a number, assigned by an adaptor, that uniquely identifies an image acquisition device. The adaptor assigns the first device it detects the identifier 1, the second device it detects the identifier 2, and so on.

You must specify the device ID as an argument to the videoinput function when you create a video input object. The object stores the value in the DeviceID property and also uses the value when constructing the default value of the Name property.

To get a list of the IDs of the devices connected to your system, use the imaqhwinfo function, specifying the name of a particular adaptor as an argument.

Characteristics

Access	Read only
Data type	double
Values	Any nonnegative integer

Example   Use the imaqhwinfo function to determine which adaptors are connected to devices on your system.

```
imaqhwinfo

ans =

    InstalledAdaptors: {'matrox'  'winvideo'}
        MATLABVersion: '6.5.1 (R13+)'
          ToolboxName: 'Image Acquisition Toolbox'
       ToolboxVersion: '1.0 (R13+)'
```

DeviceID

Use the `imaqhwinfo` function again, specifying the name of the adaptor, to find out how many devices are available through that adaptor. The `imaqhwinfo` function returns the device IDs for all the devices in the `DeviceIds` field.

```
info = imaqhwinfo('winvideo')

info =

        AdaptorDllName: [1x73 char]
     AdaptorDllVersion: '1.0 (R13+)'
           AdaptorName: 'winvideo'
             DeviceIDs: {[1]}
            DeviceInfo: [1x1 struct]
```

See Also   **Functions**

`imaqhwinfo`, `videoinput`

Properties

`Name`

Purpose　　　Specify the MATLAB AVI file object used to log data

Description　　The DiskLogger property specifies the AVI file object used to log data when the LoggingMode property is set to 'disk' or 'disk&memory'.

A MATLAB AVI file object specifies the name and other characteristics of an AVI file. For example, you can use AVI file object properties to specify the codec used for data compression and the desired quality of the output. For complete information about the AVI file object and its properties, see the avifile documentation.

Note　Do not use the variable returned by the avifile function to perform any operation on an AVI file object while it is being used by a video input object for data logging. For example, do not change any of the AVI file object properties, add frames, or close the object. Your changes could conflict with the video input object.

When the video input object finishes logging data to disk, the AVI file object remains open. The video input object does not open or close an AVI file object used for logging. The video input object, however, does update the Width, Height, and TotalFrames fields in the AVI file object to reflect the current acquisition settings.

After Logging and Running are off, it is possible that the DiskLogger might still be writing data to disk. When the DiskLogger finishes writing data to disk, the value of the DiskLoggerFrameCount property should equal the value of the FramesAcquired property. Do not close or modify the DiskLogger until this condition is met.

Characteristics

Access	Read only while running
Data type	AVI file object
Values	The default value is [].

DiskLogger

Example

Create and configure an AVI file object.

```
file = avifile('logfile.avi');
file.Quality = 50;
```

Create and configure a video input object.

```
vid = videoinput('winvideo', 1);
vid.LoggingMode = 'disk&memory';
vid.DiskLogger = file;
```

Start logging data to disk.

```
start(vid)
```

To ensure that the logged data is written to the disk file, close the AVI file. As an argument to the close function, specify the value of the video input object DiskLogger property, vid.DiskLogger, to reference the AVI file object, not the original variable, file, returned by the avifile function.

```
file = close(vid.DiskLogger);
```

Delete the image acquisition object from memory when it is no longer needed.

```
delete(vid)
clear vid
```

See Also

Functions

videoinput

Properties

DiskLoggerFrameCount, Logging, LoggingMode

Purpose   Specify the number of frames written to disk

Description   The `DiskLoggerFrameCount` property indicates the current number of frames written to disk by the `DiskLogger`. This value is only updated when the `LoggingMode` property is set to `'disk'` or `'disk&memory'`.

After `Logging` and `Running` are off, it is possible that the `DiskLogger` might still be writing data to disk. When the `DiskLogger` finishes writing data to disk, the value of the `DiskLoggerFrameCount` property should equal the value of the `FramesAcquired` property. Do not close or modify the `DiskLogger` until this condition is met.

Characteristics

Access	Read only
Data type	`double`
Values	Any nonnegative integer

See Also   **Functions**
`videoinput`

Properties
`DiskLogger`, `FramesAcquired`, `Logging`, `Running`

ErrorFcn

Purpose   Specify the M-file callback function to execute when a run-time error occurs

Description   The `ErrorFcn` property specifies the function to execute when an error event occurs. A run-time error event is generated immediately after a run-time error occurs.

Run-time errors include hardware errors and timeouts. Run-time errors do not include configuration errors such as setting an invalid property value.

Run-time error event information is stored in the `EventLog` property. You can retrieve any error message with the `Data.Message` field of `EventLog`.

Characteristics

Access	Read only while running
Data type	String, function handle, or cell array
Values	`imaqcallback` is the default callback function.

See Also   **Properties**
`EventLog, Timeout`

Purpose   Store information about events

Description   The EventLog property is an array of structures that stores information about events. Each structure in the array represents one event. Events are recorded in the order in which they occur. The first EventLog structure reflects the first event recorded, the second EventLog structure reflects the second event recorded, and so on.

Each event log structure contains two fields: Type and Data.

The Type field stores a character array that identifies the event type. The Image Acquisition Toolbox defines many different event types, listed in this table. Note that not all event types are logged.

Event Type	Description	Included in Log
Error	Run-time error occurred. Run-time errors include timeouts and hardware errors.	Yes
Frames Acquired	The number of frames specified in the FramesAcquiredFcnCount property has been acquired.	No
Start	Object was started by calling the start function.	Yes
Stop	Object stopped executing.	Yes
Timer	Timer expired.	No
Trigger	Trigger executed.	Yes

The Data field stores information associated with the specific event. For example, all events return the absolute time the event occurred in the AbsTime field. Other event-specific fields are included in Data. For more information, see "Retrieving Event Information" on page 6-6.

EventLog

Characteristics

Access	Read only
Data type	Structure array
Values	Default is empty structure array.

Example   Create a video input object.

```
vid = videoinput('winvideo');
```

Start the object.

```
start(vid)
```

View the event log to see which events occurred.

```
elog = vid.EventLog;

{elog.Type}

ans =
    'Start'    'Trigger'    'Stop'
```

View the data associated with a trigger event.

```
elog(2).Data
ans =

             AbsTime: [2003 2 11 17 22 18.9740]
     FrameMemoryLimit: 12288000
      FrameMemoryUsed: 0
          FrameNumber: 0
        RelativeFrame: 0
         TriggerIndex: 1
```

See Also   **Properties**

Logging

Purpose   Specify how often to acquire a frame from the video stream

Description   The FrameGrabInterval property specifies how often the video input object
acquires a frame from the video stream. By default, objects acquire every frame
in the video stream, but you can use this property to specify other acquisition
intervals.

Note Do not confuse the frame grab interval with the frame rate. The frame
rate describes the rate at which an image acquisition device provides frames,
typically measured in seconds, such as 30 frames per second. The frame grab
interval is measured in frames, not seconds. If a particular device's frame rate
is configurable, the video source object might include the frame rate as a
device-specific property.

For example, when you specify a FrameGrabInterval value of 3, the object
acquires every third frame from the video stream, as illustrated in this figure.
The object acquires the first frame in the video stream before applying the
FrameGrabInterval.
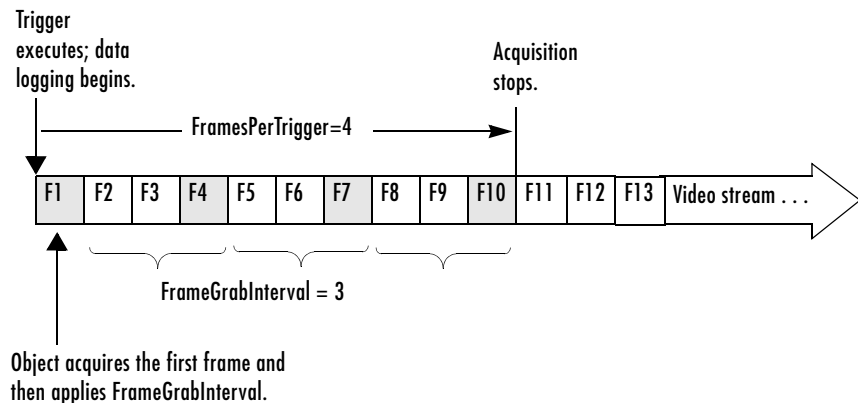
You specify the source of the video stream in the SelectedSourceName
property.

FrameGrabInterval

Characteristics

Access	Read only while running
Data type	`double`
Values	Any positive integer. The default value is 1 (acquire every frame).

See Also

Functions

`videoinput`

Properties

`SelectedSourceName`

Purpose   Indicate the total number of frames acquired

Description   The FramesAcquired property indicates the total number of frames that the object has acquired, regardless of how many frames have been extracted from the memory buffer. The video input object continuously updates the value of the FramesAcquired property as it acquires frames.

Note When you issue a start command, the video input object resets the value of the FramesAcquired property to 0 (zero) and flushes the buffer.

To find out how many frames are available in the memory buffer, use the FramesAvailable property.

Characteristics

Access	Read only
Data type	double
Values	Any nonnegative integer. The default value is 0 (zero).

See Also   **Functions**
start

Properties
FramesAvailable, FramesAcquiredFcn, FramesAcquiredFcnCount

FramesAcquiredFcn

Purpose   Specify the M-file executed when a specified number of frames have been acquired

Description   The `FramesAcquiredFcn` specifies the M-file function to execute every time a predefined number of frames have been acquired.

A frames acquired event is generated immediately after the number of frames specified by the `FramesAcquiredFcnCount` property is acquired from the selected video source. This event executes the M-file specified for `FramesAcquiredFcn`.

Use the `FramesAcquiredFcn` callback if you must access each frame that is acquired. If you do not have this requirement, you might want to use the `TimerFcn` property.

Frames acquired event information is not stored in the `EventLog` property.

Characteristics

Access	Read/write
Data type	String, function handle, or cell array
Values	The default value is an empty matrix (`[]`).

See Also   **Properties**

`EventLog`, `FramesAcquiredFcnCount`, `TimerFcn`

Purpose   Specify the number of frames that must be acquired before a frames acquired event is generated

Description   The `FramesAcquiredFcnCount` property specifies the number of frames to acquire from the selected video source before a frames acquired event is generated.

The object generates a frames acquired event immediately after the number of frames specified by `FramesAcquiredFcnCount` is acquired from the selected video source.

Characteristics

Access	Read only while running
Data type	`double`
Values	Any positive integer. The default value is `0` (zero).

See Also   **Properties**

`FramesAcquiredFcn`

FramesAvailable

Purpose　　　Indicate the number of frames available in the memory buffer

Description　　The FramesAvailable property indicates the total number of frames that are available in the memory buffer. When you extract data, the object reduces the value of the FramesAvailable property by the appropriate number of frames. You use the getdata function to extract data and move it into the MATLAB workspace.

Note　When you issue a start command, the video input object resets the value of the FramesAvailable property to 0 (zero) and flushes the buffer.

To view the total number of frames that have been acquired since the last start command, use the FramesAcquired property.

Characteristics

Access	Read only
Data type	double
Values	Any nonnegative integer. The default value is 0 (zero).

See Also　　**Functions**
getdata, start

Properties
FramesAcquired

Purpose   Specify the number of frames to acquire for each trigger using the selected video source

Description The FramesPerTrigger property specifies the number of frames the video input object acquires each time it executes a trigger using the selected video source.

When the value of the FramesPerTrigger property is set to Inf, the object keeps acquiring frames until an error occurs or you issue a stop command.

Note When the FramesPerTrigger property is set to Inf, the object ignores the value of the TriggerRepeat property.

Characteristics

Access	Read only while running
Data type	double
Values	Any positive integer. The default value is 10.

See Also   **Functions**
stop

Properties
TriggerRepeat

InitialTriggerTime

Purpose Record the absolute time of the first trigger

Description The `InitialTriggerTime` property records the absolute time of the first trigger. The absolute time is recorded as a MATLAB clock vector.

For all trigger types, `InitialTriggerTime` records the time when the `Logging` property is set to `'on'`.

To find the time when a subsequent trigger executed, view the `Data.AbsTime` field of the `EventLog` property for the particular trigger.

Characteristics

Access	Read only
Data type	Six-element vector of doubles (MATLAB clock vector)
Values	The default value is `[]`.

Example Create an image acquisition object, `vid`, for a USB-based Webcam.

```
vid = videoinput('winvideo',1);
```

Start the object. Because the `TriggerType` property is set to `'immediate'` by default, the trigger executes immediately. The object records the time of the initial trigger.

```
start(vid)

abstime = vid.InitialTriggerTime

abstime =

   1.0e+003 *

       1.9990    0.0020    0.0190    0.0130    0.0260    0.0208
```

Convert the clock vector into an integer form for display.

```
t = fix(abstime);

sprintf('%d:%d:%d', t(4),t(5),t(6))

ans =

13:26:20
```

See Also

Functions

clock, getdata

Properties

EventLog, Logging, TriggerType

Logging

Purpose   Indicate whether the object is currently logging data

Description   The Logging property indicates whether the video input object is currently logging data.

When a trigger occurs, the object sets the Logging property to 'on' and logs data to memory, a disk file, or both, depending on the value of the LoggingMode property.

The object sets the Logging property to 'off' when it acquires the requested number of frames, an error occurs, or you issue a stop command.

To acquire data when the object is running but not logging, use the peekdata function. Note, however, that the peekdata function does not guarantee that all the requested image data is returned. To acquire all the data without gaps, you must have the object log the data to memory or to a disk file.

Characteristics   Default value is enclosed in braces ({}).

Access	Read only
Data type	String
Values	[{'off'} | 'on']

See Also   **Functions**
getdata, peekdata, stop, trigger

Properties
LoggingMode, Running

Purpose   Specify the destination for acquired data

Description   The LoggingMode property specifies where you want the video input object to store the acquired data. You can specify any of the following values:

Value	Description
'disk'	Log acquired data to a disk file.
'disk&memory'	Log acquired data to both a disk file and to a memory buffer.
'memory'	Log acquired data to a memory buffer.

If you select 'disk' or 'disk&memory', you must specify the AVI file object used to access the disk file as the value of the DiskLogger property.

Note When logging data to memory, you must extract the acquired data in a timely manner with the getdata function to avoid using up all the memory that is available on your system. Use imaqmem to limit the amount of memory used.

Characteristics   Default value is enclosed in braces ({}).

Access	Read only while running
Data type	String
Values	['disk'	'disk&memory'	{'memory'}]

See Also   **Functions**
getdata

Properties
DiskLogger, Logging

Name

Purpose   Specify the name of the image acquisition object

Description   The Name property specifies a descriptive name for the image acquisition object.

Characteristics

Access	Read/write
Data type	String
Values	Any text string. The toolbox creates the default name by combining the values of the VideoFormat and DeviceID properties with the adaptor name in this format:

VideoFormat + '-' + adaptor name + '-' + DeviceID

Example   Create an image acquisition object.

```
vid = videoinput('winvideo');
```

Retrieve the value of the Name property using the get function.

```
get(vid,'Name')

ans =

    RGB555_128x96-winvideo-1
```

See Also   **Functions**
videoinput

Properties
DeviceID, VideoFormat

Purpose   Indicate the number of color bands in the data to be acquired

Description   The NumberOfBands property indicates the number of color bands in the data to be acquired. The toolbox defines *band* as the third dimension in a 3-D array, as shown in this figure.
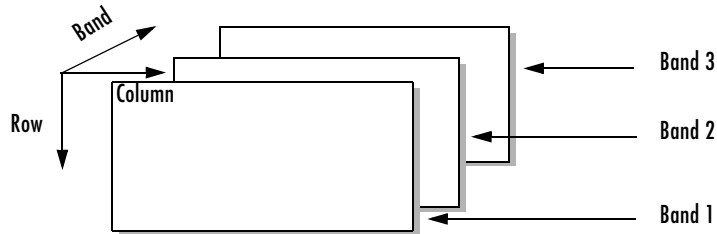
The value of the NumberOfBands property indicates the number of color bands in the data returned by getsnapshot, getdata, and peekdata.

Characteristics

Access	Read only
Data type	double
Values	Any positive integer. The default value is defined at object creation time based on the video format.

Example   Create an image acquisition object.

```
vid = videoinput('winvideo');
```

Retrieve the value of the NumberOfBands property using the get function.

```
get(vid,'NumberOfBands')

ans =

    3
```

NumberOfBands

If you retrieve the value of the `VideoFormat` property, you can see that the video data is in RGB format.

```
get(vid,'VideoFormat')

ans =

RGB24_320x240
```

See Also   **Functions**

getdata, getsnapshot, peekdata

Purpose   Identify the video input object that is the parent of a video source object

Description   The Parent property identifies the video input object that is the parent of a
video source object.

The parent of a video source object is defined as the video input object owning
the video source object.

Characteristics

Access	Read only
Data type	Video input object
Values	Defined at object creation time

See Also   **Functions**
videoinput

Previewing

Purpose   Indicate whether the object is currently previewing data in a separate window

Description   The Previewing property indicates whether the object is currently previewing data in a separate window.

The object sets the Previewing property to 'on' when you call the preview function.

The object sets the Previewing property to 'off' when you close the preview window using the closepreview function or by clicking the **Close** button in the preview window title bar.

Characteristics Default value is enclosed in braces ({}).

Access	Read only
Data type	String
Values	[{'off'} | 'on']

See Also   **Functions**
closepreview, preview

Purpose   Specify the color space used in MATLAB

Description   The ReturnedColorSpace property specifies the color space used when acquired image data is returned to the MATLAB workspace. This is only relevant when you are accessing acquired image data with the getsnapshot, getdata, and peekdata functions.

This property can have any of the following values:

Value	Description
'grayscale'	Grayscale color space
'rgb'	RGB color space
'YCbCr'	YCbCr color space
'ntsc'	NTSC color space
'hsv'	HSV color space
'unknown'	The color space used in MATLAB could not be determined.

Characteristics

Access	Read only
Data type	String
Values	Defined at object creation time and depends on the video format selected

See Also   **Functions**

getsnapshot, getdata, peekdata, videoinput

Properties

VideoFormat

ROIPosition

Purpose
Specify the region-of-interest (ROI) window

Description
The `ROIPosition` property specifies the region-of-interest acquisition window. The ROI window defines the actual size of the frame logged by the toolbox, measured with respect to the top left corner of an image frame.

Note Video input objects only support this property if the image acquisition device supports the definition of ROIs. Generic Windows image acquisition devices, accessible through the `'winvideo'` adaptor, do not support the definition of ROIs.

`ROIPosition` is specified as a 1-by-4 element vector

```
[XOffset YOffset Width Height]
```

where

XOffset	Position of the upper left corner of the ROI, measured in pixels.
YOffset	Position of the upper left corner of the ROI, measured in rows.
Width	Width of the ROI, measured in pixels. The sum of `XOffset` and `Width` cannot exceed the width specified in `VideoResolution`.
Height	Height of the ROI, measured in rows. The sum of `YOffset` and `Height` cannot exceed the height specified in `VideoResolution`.
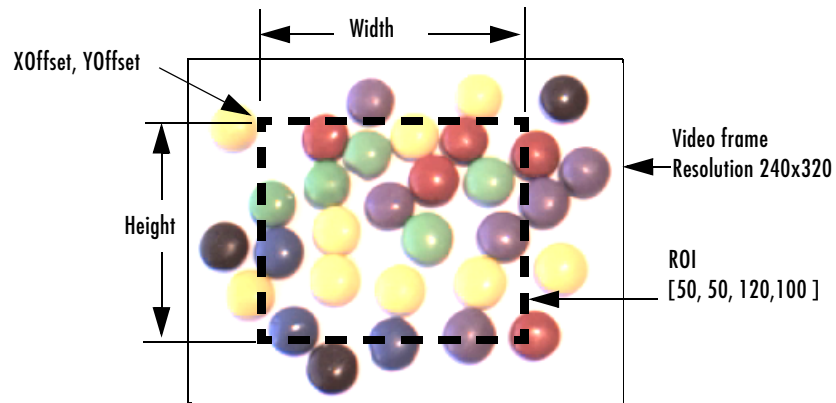
Characteristics

Access	Read only while running
Data type	1-by-4 element vector of doubles
Values	Default is [0 0 width height] where width and height are determined by VideoResolution.

See Also   **Properties**
VideoResolution

Running

Purpose	Indicate if the video input object is ready to acquire data
Description	The Running property indicates if the video input object is ready to acquire data.

Along with the Logging property, Running reflects the state of a video input object. The Running property indicates that the object is ready to acquire data, while the Logging property indicates that the object is acquiring data.

The object sets the Running property to 'on' when you issue the start command. When Running is 'on', you can acquire data from a video source.

The object sets the Running property to 'off' when any of the following conditions are met:

- The specified number of frames has been acquired.
- A run-time error occurs.
- You issue the stop command.

When Running is 'off', you cannot acquire image data. However, you can acquire one image frame with the getsnapshot function.

Characteristics   Default value is enclosed in braces ({}).

Access	Read only
Data type	String
Values	[{'off'} | 'on']

See Also   **Properties**
getsnapshot, start, stop

Properties
Logging

Purpose   Indicate if the video source object will be used for acquisition

Description   The Selected property indicates if the video source object will be used for
                     acquisition. You select a video source object by specifying its name as the value
                     of the video input object's SelectedSourceName property. The video input
                     object Source property is an array of all the video source objects associated with
                     the video input object.

                     If Selected is 'on', the video source object is selected. If the value is 'off', the
                     video source object is not selected.

                     A video source is defined to be a collection of one or more physical data sources
                     that are treated as a single entity. For example, hardware supporting multiple
                     RGB sources, each of which is made up of three physical connections (red,
                     green, and blue), is treated as a single video source object.

Characteristics Default value is enclosed in braces ({}).

Access	Read only
Data type	String
Values	[{'off'} | 'on']

Example   Create an image acquisition object.

```
vid = videoinput('winvideo');
```

                     Determine the currently selected video source object.

```
vid.SelectedSourceName

ans =

input1
```

                     Retrieve the currently selected video source object.

```
src = getselectedsource(vid);
```

Selected

View its `Name` and `Selected` properties.

```
src.SourceName

ans =

input1

src.Selected

ans =

on
```

See Also   **Functions**
getselectedsource

Properties
SelectedSourceName

Purpose   Specify the name of the currently selected video source

Description   The SelectedSourceName property specifies the name of the video source object from which the video input object acquires data. The name is specified as a string. By default, the video input object selects the first available video source object stored in the Source property.

The toolbox defines a video source as one or more hardware inputs that are treated as a single entity. For example, hardware supporting multiple RGB sources, each of which is made up of three physical connections (red-green-blue), is treated as a single video source object.

Characteristics

Access	Read only while running
Data type	String
Values	The video input object assigns a name to each video source object it creates. Names are defined at object creation time and are vendor-specific.
	By default, the toolbox uses the first available source name.

Example   To see a list of all available sources, create a video input object.

```
vid = videoinput('matrox');
```

Use the set function to view a list of all available video source objects.

```
src_names = set(obj,'SelectedSourceName');
```

See Also   **Functions**
set

Properties
Source

Source

Purpose   Indicate the video source objects associated with a video input object

Description   The `Source` property is a vector of video source objects that represent the physical data sources connected to a device. When a video input object is created, the toolbox creates a vector of video source objects associated with the video input object.

Each video source object created is provided a unique source name. You can use the source name to select the desired acquisition source by configuring the `SelectedSourceName` property of the video input object.

A video source object's name is stored in its `SourceName` property. If a video source object's `SourceName` is equivalent to the video input object's `SelectedSourceName`, the video source object's `Selected` property has a value of `'on'`.

The video source object supports a set of common properties, such as `SourceName`. Each video source object can also support device-specific properties that control characteristics of the physical device such as brightness, hue, and saturation. Different image acquisition devices expose different sets of properties.

A video source is defined to be a collection of one or more physical data sources that are treated as a single entity. For example, hardware supporting multiple RGB sources, each of which is made up of three physical connections (red-green-blue), is treated as a single video source object.

The `Source` property encapsulates one or more video sources. To reference a video source, you use a numerical integer to index into the vector of video source objects.

Characteristics

Access	Read only
Data type	Vector of video source objects
Values	Defined at object creation time

Example   Create an image acquisition object.

```
vid = videoinput('matrox');
```

To access all the video source objects associated with a video input object, use the Source property of the video input object. (To view only the currently selected video source object, use the getselectedsource function.)

```
sources = vid.Source;
src = sources(1);
```

To view the properties of the video source object src, use the get function.

```
get(src)
  General Settings:
    Parent = [1x1 videoinput]
    Selected = on
    SourceName = CH1
    Tag =
    Type = videosource

  Device Specific Properties:
    InputFilter = lowpass
    UserOutputBit3 = off
    UserOutputBit4 = off
    XScaleFactor = 1
    YScaleFactor = 1
```

See Also   **Functions**

videoinput, getselectedsource

Properties

SelectedSourceName

SourceName

Purpose   Indicate the name of a video source object

Description The `SourceName` property indicates the name of a video source object.

SourceName is one of the values in the video input object's `SelectedSourceName` property.

Characteristics

Access	Read only
Data type	String
Values	Defined at object creation time

See Also   **Functions**

getselectedsource

Properties

SelectedSourceName, Source

Purpose   Specify the M-file executed when a start event occurs

Description   The StartFcn property specifies the M-file function to execute when a start event occurs. A start event occurs immediately after you issue the start command.

When the StartFcn M-file finishes executing, the toolbox sets the object's Running property to 'on' and the object begins executing.

Start event information is stored in the EventLog property.

Characteristics

Access	Read/write
Data type	String, function handle, or cell array
Values	The default value is an empty matrix ([]).

See Also   **Properties**
EventLog, Running

StopFcn

Purpose　　　Specify the M-file executed when a stop event occurs

Description　　The StopFcn property specifies the M-file function to execute when a stop event occurs. A stop event occurs immediately after you issue the stop command.

Under most circumstances, the M-file is not guaranteed to complete execution until sometime after the image acquisition object stops and the Running property is set to 'off'.

Stop event information is stored in the EventLog property.

Characteristics

Access	Read/write
Data type	String, function handle, or cell array
Values	The default value is an empty matrix ([]).

See Also　　　**Properties**

EventLog, Running

Purpose   Specify descriptive text to associate with an image acquisition object

Description   The Tag property specifies any descriptive text that you want to associate with an image acquisition object.

The Tag property can be useful when you are constructing programs that would otherwise need to define the image acquisition object as a global variable, or pass the object as an argument between callback routines.

You can use the value of the Tag property to search for particular image acquisition objects when using the imaqfind function.

Characteristics

Access	Read/Write
Data type	String
Values	Any text string

See Also   **Functions**
imaqfind

Timeout

Purpose   Specify how long to wait for image data

Description   The `Timeout` property specifies the amount of time (in seconds) that the
                     `getdata` and `getsnapshot` functions wait for data to be returned. The `Timeout`
                     property is only associated with these blocking functions. If the specified time
                     period expires, the functions return control to the MATLAB command line.

                     A timeout is one of the conditions for stopping an acquisition. When a timeout
                     occurs, and the object is running, the M-file function specified by `ErrorFcn` is
                     called.

                     Note The `Timeout` property is not associated with hardware timeout
                     conditions.

Characteristics

Access	Read only while running
Data type	`double`
Values	Any positive integer. The default value is 10 seconds.

See Also   **Functions**
                     `getdata`, `getsnapshot`

                     Properties
                     `EventLog`, `ErrorFcn`

Purpose   Specify the M-file callback function to execute when a timer event occurs

Description   The TimerFcn property specifies the M-file callback function to execute when a timer event occurs. A timer event occurs when the time period specified by the TimerPeriod property expires.

The toolbox measures time relative to when the object is started with the start function. Timer events stop being generated when the image acquisition object stops running.

Note Some timer events might not be processed if your system is significantly slowed or if the TimerPeriod value you specify is too small.

Characteristics

Access	Read/write
Data type	String, function handle, or cell array
Values	The default value is an empty matrix ([ ]).

See Also   **Functions**
start, stop

Properties
TimerPeriod

TimerPeriod

Purpose   Specify the number of seconds between timer events

Description   The TimerPeriod property specifies the amount of time, in seconds, that must pass before a timer event is triggered.

The toolbox measures time relative to when the object is started with the start function. Timer events stop being generated when the image acquisition object stops running.

Note Some timer events might not be processed if your system is significantly slowed or if the TimerPeriod value you specify is too small.

Characteristics

Access	Read only while running
Data type	double
Values	Any positive value. The minimum value is 0.01 seconds. The default value is 1 (second).

See Also   **Functions**
start, stop

Properties
EventLog, TimerFcn

Purpose   Indicate the condition that must be met before a trigger event occurs

Description   The TriggerCondition property indicates the condition that must be met, via the TriggerSource, before a trigger event occurs. The trigger conditions that you can specify depend on the value of the TriggerType property.

TriggerType Value	Conditions Available
'hardware' (if available for your device)	Device-specific. For example, some Matrox hardware support conditions such as 'risingEdge' and 'fallingEdge'. Use the triggerinfo function to view a list of valid values to use with your image acquisition hardware.
'immediate'	'none'
'manual'	'none'

You must use the triggerconfig function to set the value of this property.

Characteristics

Access	Read only. Use the triggerconfig function to set the value of this property.
Data type	String
Values	Device specific. Use the triggerinfo function to view a list of valid values to use with your image acquisition hardware.

See Also   **Functions**
trigger, triggerconfig, triggerinfo

Properties
TriggerSource, TriggerType

TriggerFcn

Purpose   Specify the M-file callback function to execute when a trigger event occurs

Description   The `TriggerFcn` property specifies the M-file callback function to execute when a trigger event occurs. The toolbox generates a trigger event when a trigger is executed based on the configured `TriggerType`, and data logging is initiated.

Under most circumstances, the M-file callback function is not guaranteed to complete execution until sometime after the toolbox sets the `Logging` property to `'on'`.

Trigger event information is stored in the `EventLog` property.

Characteristics

Access	Read/write
Data type	String, function handle, or cell array
Values	The default value is an empty matrix (`[]`).

See Also   **Functions**
`trigger`

Properties
`EventLog`, `Logging`

Purpose   Specify the number of frames to skip before acquiring frames after a trigger occurs

Description   The TriggerFrameDelay property specifies the number of frames to skip before acquiring frames after a trigger occurs. The object waits the specified number of frames after the trigger before starting to log frames.

In this figure, the TriggerFrameDelay is set to 5, so the object lets five frames pass before starting to acquire frames. The number of frames captured is defined by the FramesPerTrigger property.
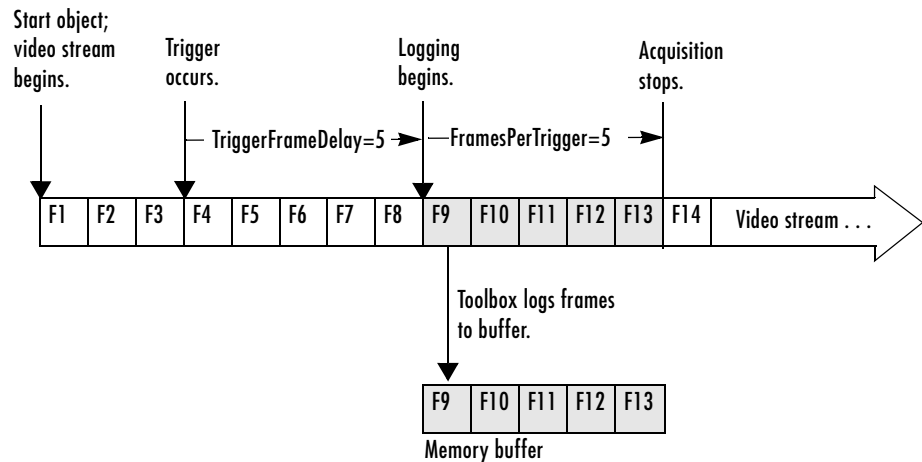
Characteristics

Access	Read only while running
Data type	double
Values	Any integer. The default value is 0 (zero).

See Also   **Functions**
trigger

Properties
FramesPerTrigger

TriggerRepeat

Purpose   Specify the number of additional times to execute a trigger

Description   The TriggerRepeat property specifies the number of additional times you want the object to execute a trigger. This table describes the behavior for several typical TriggerRepeat values.

Value	Behavior
0 (default)	Execute the trigger once when the trigger condition is met.
Any positive integer	Execute the trigger the specified number of additional times when the trigger condition is met.
Inf	Keep executing the trigger every time the trigger condition is met until the stop function is called or an error occurs.

To determine how many triggers have executed, check the value of the TriggersExecuted property.

Note If the FramesPerTrigger property is set to Inf, the object ignores the value of the TriggerRepeat property.

Characteristics

Access	Read only while running
Data type	double
Values	Any nonnegative integer. The default value is 0 (zero).

See Also

Functions

stop, trigger

Properties

FramesPerTrigger, TriggersExecuted, TriggerType

TriggersExecuted

Purpose   Indicate the total number of triggers that have been executed

Description   The `TriggersExecuted` property indicates the total number of triggers that the
                   video input object has executed.

Characteristics

Access	Read only
Data type	`double`
Values	Any nonnegative integer. The default value is `0` (zero).

See Also   **Functions**
                   `trigger`

                   Properties
                   `EventLog`

Purpose   Indicate the hardware source to monitor for trigger conditions

Description   The TriggerSource property indicates the hardware source the image acquisition object monitors for trigger conditions. When the condition specified in the TriggerCondition property is met, the object executes the trigger and starts acquiring data.

You use the triggerconfig function to specify this value. The value of the TriggerSource property is device specific. You specify whatever mechanism a particular device uses to generate triggers.

For example, for Matrox hardware, the TriggerSource property could have values such as 'Port0' or 'Port1'. Use the triggerinfo function to view a list of values that are valid for your image acquisition device.

You must use the triggerconfig function to set the value of this property.

Note The TriggerSource property is only used when the TriggerType property is set to 'hardware'.

Characteristics

Access	Read only. Use the triggerconfig function to set the value of this property.
Data type	String
Values	Device-specific. Use the triggerinfo function to get a list of valid values.

See Also   **Functions**

trigger, triggerconfig, triggerinfo

Properties

TriggerCondition, TriggerType

TriggerType

Purpose   Indicate the type of trigger used by a video input object

Description   The TriggerType property indicates the type of trigger used by the video input object. Triggers initiate data acquisition.

You use the triggerconfig function to specify one of the following values for this property.

Trigger Type	Description
'hardware' (if available for your device)	Trigger executes when a specified condition is met. You specify the condition using the TriggerCondition property and you specify the hardware source to monitor for the condition in the TriggerSource property. You use the triggerconfig function to set the values of these properties.
'immediate'	Trigger executes immediately after you call the start function.
'manual'	Trigger executes immediately after you call the trigger function.

Characteristics Default value is enclosed in braces ({}).

Access	Read only. Use the triggerconfig function to set the value of this property.
Data type	String
Values	['hardware'	{'immediate'}	'manual'] The 'hardware' option is only included for devices that support hardware triggers.

See Also   **Functions**
trigger, triggerconfig, triggerinfo

Properties
TriggerCondition, TriggerSource

Purpose   Identify the type of image acquisition object

Description   The Type property identifies the type of image acquisition object. An image acquisition object can be either one of two types:

- Video input object
- Video source object

Characteristics

Access	Read only
Data type	String
Values	['videoinput' | 'videosource'] Defined at object creation time

Example
```
vid = videoinput('winvideo',1)

get(vid,'Type')

ans =

videoinput
```
This example gets the type of a video source object.
```
src = getselectedsource(vid);
get(src,'type')
ans =
 videosource
```

See Also   **Functions**

getselectedsource, videoinput

UserData

Purpose   Store data that you want to associate with an image acquisition object

Description   The UserData property specifies any data that you want to associate with an image acquisition object.

Note The object does not use the data in UserData directly. However, you can access the data by using the get function or by referencing the property as you would a field in a MATLAB structure.

Characteristics

Access	Read/Write
Data type	Any
Values	User-defined

See Also   **Functions**
get

Purpose   Specify the video format or the name of a device configuration file

Description   The VideoFormat property specifies the video format used by the image acquisition device or the name of a device configuration file, depending on which you specified when you created the object using the videoinput function.

Image acquisition devices typically support multiple video formats. When you create a video input object, you can specify the video format that you want the device to use. If you do not specify the video format as an argument, the videoinput function uses the default format. Use the imaqhwinfo function to determine which video formats a particular device supports and find out which format is the default.

As an alternative, you can specify the name of a device configuration file, also known as a camera file or Digitizer Configuration Format (DCF) file. Some image acquisition devices use these files to store device configuration information. The videoinput function can use this file to determine the video format and other configuration information.

Use the imaqhwinfo function to determine if your device supports device configuration files.

Characteristics

Access	Read only
Data type	String
Values	Device-specific. The example describes how to get a list of all the formats supported by a particular image acquisition device.

Example   To determine the video formats supported by a device, check the SupportedFormats field in the device information structure returned by imaqhwinfo.

VideoFormat

```
info = imaqhwinfo('winvideo')

info =

       AdaptorDllName: [1x73 char]
    AdaptorDllVersion: '1.0 (R13+)'
          AdaptorName: 'winvideo'
            DeviceIDs: {[1]}
           DeviceInfo: [1x1 struct]

info.DeviceInfo

ans =

           DefaultFormat: 'RGB555_128x96'
      DeviceFileSupported: 0
             DeviceName: 'IBM PC Camera'
               DeviceID: 1
       ObjectConstructor: 'videoinput('winvideo', 1)'
        SupportedFormats: {1x34 cell}
```

See Also   **Functions**

imaqhwinfo, videoinput

Purpose   Indicate the width and height of the incoming video stream

Description   The `VideoResolution` property is a two-element vector indicating the width and height of the frames in the incoming video stream. `VideoResolution` is specified as

```
[Width Height]
```

Width is measured in pixels and height is measured in rows.

> **Note** You specify the video resolution when you create the video input object, by passing in the video format argument to the `videoinput` function. If you do not specify a video format, the `videoinput` function uses the default video format. Use the `imaqhwinfo` function to determine which video formats a particular device supports and find out which format is the default.

Characteristics

Access	Read only
Data type	Vector of `doubles`
Values	Defined by video format

See Also   **Functions**
`imaqhwinfo`, `videoinput`

Properties
`ROIPosition`, `VideoFormat`

VideoResolution

Index