Image Acquisition Toolbox

For Use with MATLAB® and Simulink®

- Computation
- Visualization
- Programming
- Simulation

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.

Image Acquisition Toolbox User's Guide

Revision History

March 2003	First printing	New for Version 1.0 (Release 13+)
September 2003	Online only	Revised for Version 1.1 (Release 13SP1)
June 2004	Online only	Revised for Version 1.5 (Release 14)
July 2004	Online only	Revised for Version 1.6 (Release 14+)
October 2004	Online only	Revised for Version 1.7 (Release 14 SP1)
March 2005	Online only	Revised for Version 1.8 (Release 14 SP2)
March 2005	Second printing	Minor Revision for Version 1.8
August 2005	Third printing	Minor Revision for Version 1.8
September 2005	Online only	Revised for Version 1.9 (Release 14 SP3)

Contents

1 Getting Started

2 Introduction

3 Connecting to Hardware

4 Acquiring Image Data

5 Working with Acquired Image Data

6 Using Events and Callbacks

7 Using the Image Acquisition Toolbox Block Library

8 Adding Support for Additional Hardware

9 Troubleshooting

10 Function Reference

11 Property Reference

12 Block Reference

Index

1

Getting Started

The best way to learn about the capabilities of the Image Acquisition Toolbox is to look at a simple example. This chapter introduces the toolbox and 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.

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.

The Image Acquisition Toolbox also includes a Simulink® block, called the Video Input block, that can be used to bring live video data into a model.

Installation and 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`).

Related Products

The MathWorks provides several products that are relevant to the kinds of tasks you can perform with the Image Acquisition Toolbox and that extend the capabilities of MATLAB. For information about these related products, see `www.mathworks.com/products/imaq/related.html`.

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	Description
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 1-8.

The following table lists this information. You use the `imaqhwinfo` function to retrieve each item.

Device Information	Description
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 "Determining the Adaptor Name" on page 1-7 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 "Determining the Device ID" on page 1-7 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 "Determining the Supported Video Formats" on page 1-8 for more information. **Note:** Specifying the video format is optional; the toolbox uses one of the supported formats as the default.

Determining the Adaptor Name

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

```
imaqhwinfo
ans =

    InstalledAdaptors: {'dcam'  'winvideo'}
        MATLABVersion: '7.0.4 (R14SP2)'
          ToolboxName: 'Image Acquisition Toolbox'
       ToolboxVersion: '1.8 (R14SP2)'
```

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: `'dcam'` 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.

Determining the 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 "Determining the Adaptor Name" on page 1-7.) In the data returned, the `DeviceIDs` field is a cell array containing the device IDs of all the devices accessible through the specified adaptor.

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

```
info = imaqhwinfo('dcam')
info =

       AdaptorDllName: [1x77 char]
    AdaptorDllVersion: '1.8 (R14SP2)'
          AdaptorName: 'dcam'
            DeviceIDs: {[1]}
           DeviceInfo: [1x1 struct]
```

Determining the Supported Video Formats

To determine which video formats an image acquisition device supports, look in the `DeviceInfo` field of the data returned by `imaqhwinfo`. The `DeviceInfo` field is a structure array where each structure provides information about a particular device. To view the device information for a particular device, you can use the device ID as a reference into the structure array. Alternatively, you can view the information for a particular device by calling the `imaqhwinfo` function, specifying the adaptor name and device ID as arguments.

To get the list of the video formats supported by a device, look at `SupportedFormats` field in the device information structure. The `SupportedFormats` field is a cell array of strings where each string is the name of a video format supported by the device. For more information, see "Determining Supported Video Formats" on page 3-6.

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

dev_info =

          DefaultFormat: 'F7_Y8_1024x768'
    DeviceFileSupported: 0
             DeviceName: 'XCD-X700  1.05'
               DeviceID: 1
      ObjectConstructor: 'videoinput('dcam', 1)'
       SupportedFormats: {'F7_Y8_1024x768'  'Y8_1024x768'}
```

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. For more information about image acquisition objects, see Chapter 3, "Connecting to Hardware."

To create a video input object, use the `videoinput` function at the MATLAB prompt. The `DeviceInfo` structure returned by the `imaqhwinfo` function contains the default `videoinput` function syntax for a device in the `ObjectConstructor` field. For more information the device information structure, see "Determining the Supported Video Formats" on page 1-8.

The following example creates a video input object for the DCAM adaptor. Substitute the adaptor name of the image acquisition device available on your system.

```
vid = videoinput('dcam',1,'Y8_1024x768')
```

The `videoinput` function accepts three arguments: the adaptor name, device ID, and video format. You retrieved this information in step 2. The adaptor name is the only required argument; the `videoinput` function can use defaults for the device ID and video format. To determine the default video format, look at the `DefaultFormat` field in the device information structure. See "Determining the Supported Video Formats" on page 1-8 for more information.

Instead of specifying the video format, you can optionally specify the name of a device configuration file, also known as a camera file. Device configuration files are typically supplied by frame grabber vendors. These files contain all the required configuration settings to use a particular camera with the device. See "Using Device Configuration Files (Camera Files)" on page 3-14 for more information.

Viewing the Video Input Object Summary

To view a summary of the video input object you just created, enter the variable name (`vid`) at the MATLAB command prompt. The summary information displayed 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 1-12 for more information.

```
vid

Summary of Video Input Object Using 'XCD-X700  1.05'.

   Acquisition Source(s):  input1 is available.

  Acquisition Parameters:  'input1' is the current selected source.
                           10 frames per trigger using the selected source.
                           'Y8_1024x768' 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.
```

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.

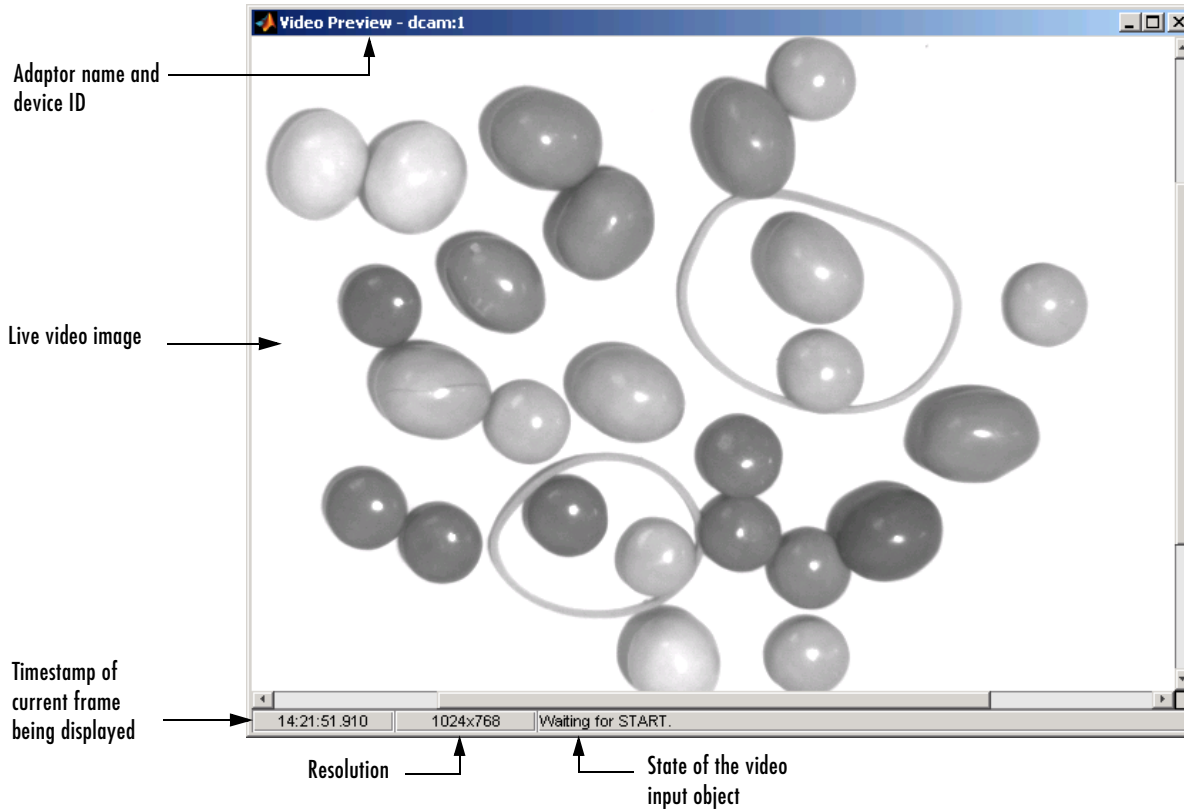
Note This step is optional at this point in the procedure because you can preview a video stream at any time after you create a video input object.

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 Video Preview figure window on your screen containing the live video stream. To stop the stream of live video, you can call the `stoppreview` function. To restart the preview stream, call `preview` again on the same video input object.

While a preview window is open, the video input 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 Video Preview window for the example.



Video Preview Window

To close the Video 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)
```

Calling `closepreview` without any arguments closes all open Video Preview windows.

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 = 0
    EventLog = [1x0 struct]
    FrameGrabInterval = 1
    FramesAcquired = 0
    FramesAvailable = 0
    FramesPerTrigger = 10
    Logging = off
    LoggingMode = memory
    Name = Y8_1024x768-dcam-1
    NumberOfBands = 1
    Previewing = on
    ReturnedColorSpace = grayscale
    ROIPosition = [0 0 1024 768]
    Running = off
    Tag =
    Timeout = 10
    Type = videoinput
    UserData = []
    VideoFormat = Y8_1024x768
    VideoResolution = [1024 768]
    .
    .
    .
```

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 = input1
    Tag =
    Type = videosource

  Device Specific Properties:
    FrameRate = 15
    Gain = 2048
    Shutter = 2715
```

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 following 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.

Running the Example

To run the example, enter the following 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 example uses functions in the Image Processing Toolbox.

```
% Create video input object.
vid = videoinput('dcam',1,'Y8_1024x768')

% Set video input object properties for this application.
% Note that example uses both SET method and dot notation method.
set(vid,'TriggerRepeat',Inf);
vid.FrameGrabInterval = 5;

% Set value of a video source object property.
vid_src = getselectedsource(vid);
set(vid_src,'Tag','motion detection setup');

% Create a figure window.
figure;

% Start acquiring frames.
start(vid)

% Calculate difference image and display it.
while(vid.FramesAcquired<=100) % Stop after 100 frames
    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.

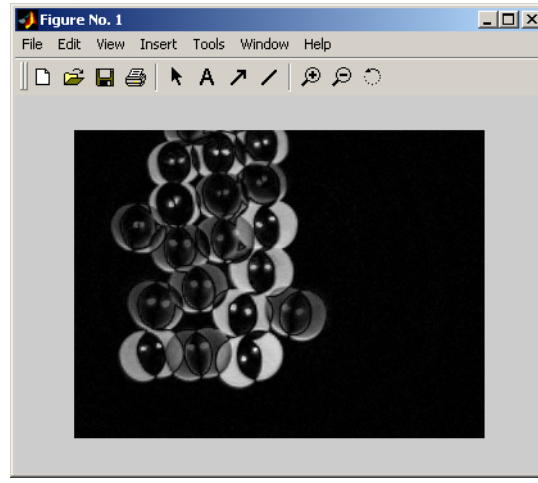


Figure Window Displayed by Example

Image Data in the MATLAB Workspace

In the example, the `getdata` function returns the image frames in the variable `data` as a 480-by-640-by-1-by-10 array of 8-bit data (`uint8`).

```
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
```

The height and width of the array are primarily determined by the video resolution of the video format. However, you can use the `ROIPosition` property to specify values that supersede the video resolution. Devices typically express video resolution as column-by-row; MATLAB expresses matrix dimensions as row-by-column.

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.

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 (p. 2-2)	Provides an overview of the Image Acquisition Toolbox
Setting Up Image Acquisition Hardware (p. 2-5)	Describes how to set up your image acquisition device
Previewing Data (p. 2-8)	Describes how to view the incoming video stream, without actually acquiring data

Overview

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.

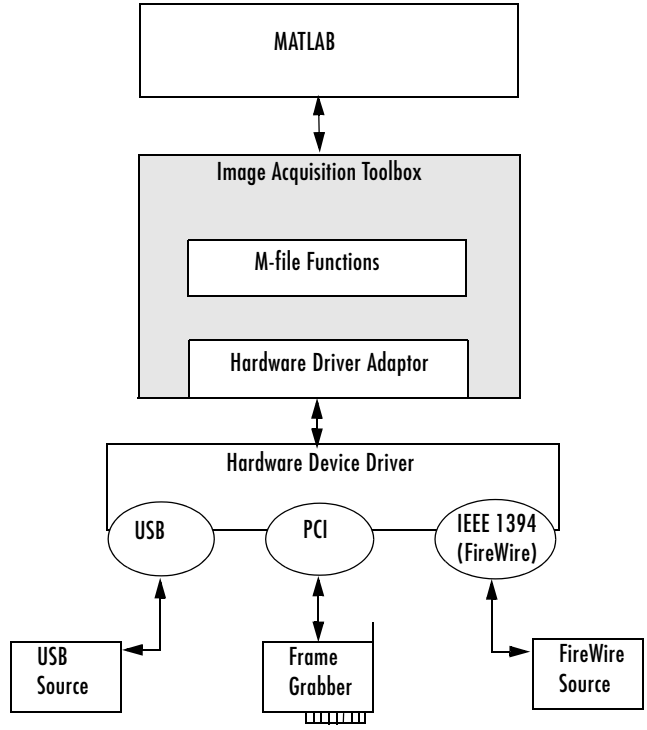


Image Acquisition Toolbox Components

Supported Devices

The Image Acquisition Toolbox includes adaptors that provide support for several vendors of professional grade image acquisition equipment, including

- Coreco Imaging, Inc.
- Data Translation, Inc.
- Matrox, Inc.

The toolbox can also connect with digital cameras that support the IIDC 1394-based Digital Camera Specification (DCAM), developed by the 1394 Trade Association. The DCAM specification describes a generic interface for exchanging data with IEEE 1394 (FireWire) digital cameras that is often used in scientific applications. The DCAM adaptor supports format 7, also known as

partial scan mode. The toolbox uses the prefix `F7_` to identify Format 7 video format names.

In addition, the toolbox supports many Windows video acquisition devices that provide Windows Driver Model (WDM) or Video for Windows (VFW) drivers, such as

- USB and IEEE 1394 (FireWire, i.LINK®) Web cameras
- Digital video (DV) camcorders
- TV tuner cards

The toolbox can support additional hardware through adaptors developed by image acquisition device vendors or other third parties. 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

To acquire image data, you must perform the setup required by your particular image acquisition device. 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) or IEEE 1394 (FireWire) port. The setup required varies with the type of device.

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.

Setting Up Frame Grabbers

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

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

Setting Up Generic Windows Video Acquisition Devices

IEEE 1394 (FireWire) and generic Windows video acquisition devices that use Windows Driver Model (WDM) or Video for Windows (VFW) device drivers typically require less setup. Plug the device into the USB or IEEE 1394 (FireWire) port on your computer and install the device driver provided by the vendor.

Setting Up DCAM Devices

If you intend to access a DCAM-compliant IEEE 1394 (FireWire) camera, you must install and configure the Carnegie Mellon University (CMU) DCAM driver. The toolbox is not compatible with any other vendor-supplied driver, even if the driver is DCAM compliant. See "Installing and Configuring the CMU DCAM Driver" on page 9-8 for more information.

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 Chapter 3, "Connecting to Hardware"), you can view the live video stream using the Video Preview window. Previewing the video data can help you 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. If you change characteristics of the image, by using video input object and video source object properties, the image displayed in the Video Preview window changes to reflect the new property settings.

The following sections provide more information about using the Video Preview window.

- "Opening a Video Preview Window" on page 2-8
- "Stopping the Preview Video Stream" on page 2-10
- "Closing a Video Preview Window" on page 2-11

Instead of using the toolbox's Video Preview window, you can display the live video preview stream in any Handle Graphics image object you specify. In this way, you can include video previewing in a GUI of your own creation. The following sections describe this capability.

- "Previewing Data in Custom GUIs" on page 2-11
- "Performing Custom Processing of Previewed Data" on page 2-13

Opening a Video Preview Window

To open a Video Preview window, use the `preview` function. The Video Preview window displays the live video stream from the device. You can only open one preview window per device. If multiple devices are used, you can open multiple preview windows at the same time.

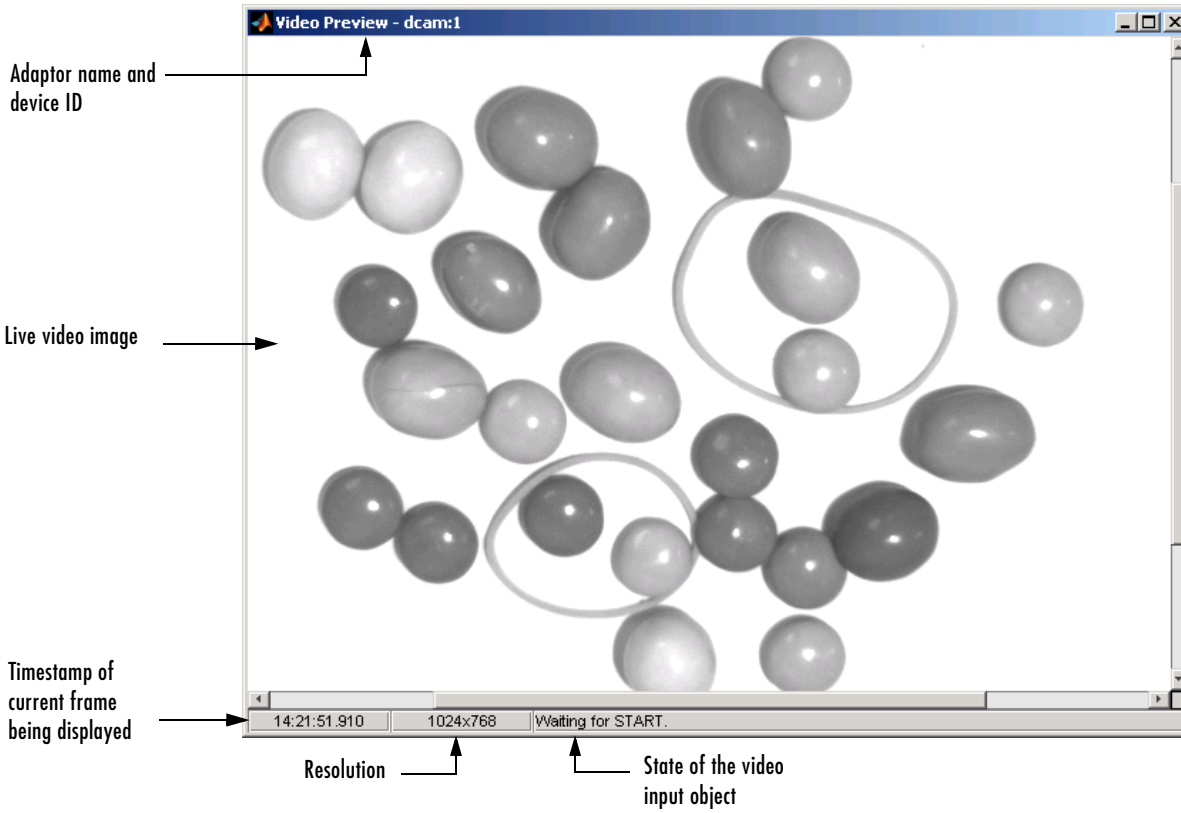
The following example creates a video input object and then opens a Video Preview window for the video input object.

```
vid = videoinput('winvideo');
preview(vid);
```

The following figure shows the Video Preview window created by this example. The Video Preview window displays the live video stream. The size of the preview image is determined by the value of the video input object's `ROIPosition` property. The Video Preview window displays the video data at 100% magnification (one screen pixel represents one image pixel).

In addition to the preview image, the Video Preview window includes information about the image, such as the timestamp of the video frame, the video resolution, and the current status of the video input object.

Note Because video formats typically express resolution as width-by-height, the Video Preview window expresses the size of the image frame as column-by-row, rather than the standard MATLAB row-by-column format.



Stopping the Preview Video Stream

When you use the `preview` function to start previewing image data, the Video Preview window displays a view of the live video stream coming from the device. To stop the updating of the live video stream, call the `stoppreview` function.

This example creates a video input object and opens a Video Preview window. The example then calls the `stoppreview` function on this video input object. The Video Preview window stops updating the image displayed and stops updating the timestamp. The status displayed in the Video Preview window also changes to indicate that previewing has been stopped.

```
vid = videoinput('winvideo');
preview(vid)
stoppreview(vid)
```

To restart the video stream in the Video Preview window, call `preview` again on the same video input object.

```
preview(vid)
```

Closing a Video Preview Window

To close a particular Video Preview window, use the `closepreview` function, specifying the video input object as an argument. You do not need to stop the live video stream displayed in the Video Preview window before closing it.

```
closepreview(vid)
```

To close all currently open Video Preview windows, use the `closepreview` function without any arguments.

```
closepreview
```

Note When called without an argument, the `closepreview` function only closes Video Preview windows. The `closepreview` function does not close any other figure windows in which you have directed the live preview video stream. For more information, see "Previewing Data in Custom GUIs".

Previewing Data in Custom GUIs

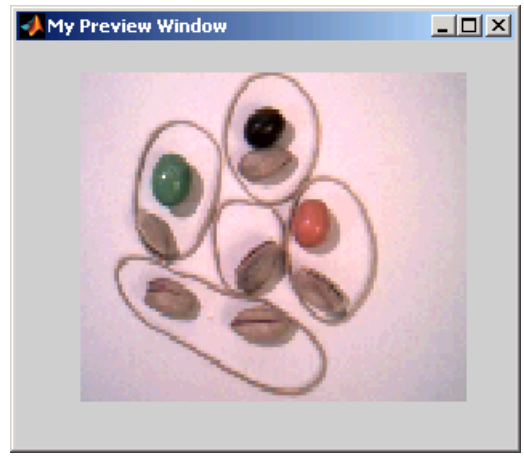
Instead of using the toolbox's Video Preview window, you can use the `preview` function to direct the live video stream to any Handle Graphics image object. In this way, you can incorporate the toolbox's previewing capability in a GUI of your own creation. (You can also perform custom processing as the live video is displayed. For information, see "Performing Custom Processing of Previewed Data" on page 2-13.)

To use this capability, create an image object and then call the `preview` function, specifying a handle to the image object as an argument. The `preview` function outputs the live video stream to the image object you specify.

The following example creates a figure window and then creates an image object in the figure, the same size as the video frames. The example then calls the `preview` function, specifying a handle to the image object.

```
% Create a video input object.
vid = videoinput('winvideo');

% Create a figure window. This example turns off the default
% toolbar, menubar, and figure numbering.

figure('Toolbar','none',...
       'Menubar', 'none',...
       'NumberTitle','Off',...
       'Name','My Preview Window');

% Create the image object in which you want to display
% the video preview data. Make the size of the image
% object match the dimensions of the video frames.

vidRes = get(vid, 'VideoResolution');
nBands = get(vid, 'NumberOfBands');
hImage = image( zeros(vidRes(2), vidRes(1), nBands) );

% Display the video data in your GUI.

preview(vid, hImage);
```

When you run this example, it creates the GUI shown in the following figure.



Custom Preview

Performing Custom Processing of Previewed Data

When you specify an image object to the `preview` function (see "Previewing Data in Custom GUIs" on page 2-11), you can optionally also specify a function that `preview` executes every time it receives an image frame.

To use this capability, follow these steps:

1. Create the function you want executed for each image frame, called the update preview window function. For information about this function, see "Creating the Update Preview Window Function" on page 2-14.

2. Create an image object.

3. Configure the value of the image object's `'UpdatePreviewWindowFcn'` application-defined data to be a function handle to your update preview window function. For more information, see "Specifying the Update Preview Function" on page 2-15.

4. Call the `preview` function, specifying the handle of the image object as an argument.

Note If you specify an update preview window function, in addition to whatever processing your function performs, it must display the video data in the image object. You can do this by updating the `CData` of the image object with the incoming video frames. For some performance guidelines about updating the data displayed in an image object, see Technical Solution 1-1B022.

Creating the Update Preview Window Function

When `preview` calls the update preview window function you specify, it passes your function the following arguments.

Argument	Description	
`obj`	Handle to the video input object being previewed	
`event`	A data structure containing the following fields:	
	`Data`	Current image frame specified as an H-by-W-by-B array, where H is the image height and W is the image width, as specified in the `ROIPosition` property, and B is the number of color bands, as specified in the `NumberOfBands` property
	`Resolution`	Text string specifying the current image width and height, as defined by the `ROIPosition` property
	`Status`	String describing the status of the video input object
	`Timestamp`	String specifying the time associated with the current image frame, in the format `hh:mm:ss:ms`
`himage`	Handle to the image object in which the data is to be displayed	

The following example creates an update preview window function that displays the timestamp of each incoming video frame as a text label in the custom GUI. The update preview window function uses `getappdata` to retrieve a handle to the text label `uicontrol` object from application-defined data in the image object. The custom GUI stores this handle to the text label `uicontrol` object — see "Specifying the Update Preview Function" on page 2-15.

Note that the update preview window function also displays the video data by updating the `CData` of the image object.

```
function mypreview_fcn(obj,event,himage)
% Example update preview window function.

% Get timestamp for frame.
tstampstr = event.Timestamp;

% Get handle to text label uicontrol.
ht = getappdata(himage,'HandleToTimestampLabel');

% Set the value of the text label.
set(ht,'String',tstampstr);

% Display image data.
set(himage, 'CData', event.Data)
```

Specifying the Update Preview Function

To use an update preview window function, store a function handle to your function in the `'UpdatePreviewWindowFcn'` application-defined data of the image object. The following example uses the `setappdata` function to configure this application-defined data to a function handle to the update preview window function described in "Creating the Update Preview Window Function" on page 2-14.

This example extends the simple custom preview window created in "Previewing Data in Custom GUIs" on page 2-11. This example adds three push button `uicontrol` objects to the GUI: **Start Preview**, **Stop Preview**, and **Close Preview**.

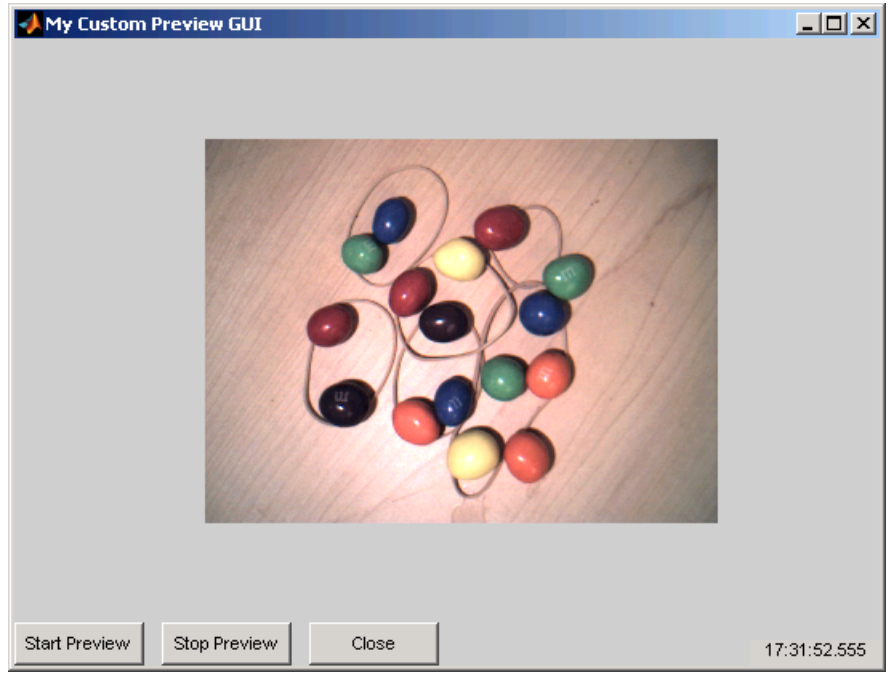
In addition, to illustrate using an update preview window function, the example GUI includes a text label `uicontrol` object to display the timestamp value. The update preview window function updates this text label each time a

framed is received. The example uses `setappdata` to store a handle to the text label `uicontrol` object in application-defined data in the image object. The update preview window function retrieves this handle to update the timestamp display.

```
% Create a video input object.
vid = videoinput('winvideo');

% Create a figure window. This example turns off the default
% toolbar and menubar in the figure.
hFig = figure('Toolbar','none',...
       'Menubar', 'none',...
       'NumberTitle','Off',...
       'Name','My Custom Preview GUI');

% Set up the push buttons
uicontrol('String', 'Start Preview',...
    'Callback', 'preview(vid)',...
    'Units','normalized',...
    'Position',[0 0 0.15 .07]);
uicontrol('String', 'Stop Preview',...
    'Callback', 'stoppreview(vid)',...
    'Units','normalized',...
    'Position',[.17 0 .15 .07]);
uicontrol('String', 'Close',...
    'Callback', 'close(gcf)',...
    'Units','normalized',...
    'Position',[0.34 0 .15 .07]);

% Create the text label for the timestamp
hTextLabel = uicontrol('style','text','String','Timestamp', ...
    'Units','normalized',...
    'Position',[0.85 -.04 .15 .08]);

% Create the image object in which you want to
% display the video preview data.
vidRes = get(vid, 'VideoResolution');
imWidth = vidRes(1);
imHeight = vidRes(2);
nBands = get(vid, 'NumberOfBands');
```

```
hImage = image( zeros(imHeight, imWidth, nBands) );

% Specify the size of the axes that contains the image object
% so that it displays the image at the right resolution and
% centers it in the figure window.
figSize = get(hFig,'Position');
figWidth = figSize(3);
figHeight = figSize(4);
set(gca,'unit','pixels',...
        'position',[ ((figWidth - imWidth)/2)...
                     ((figHeight - imHeight)/2)...
                       imWidth imHeight ]);

% Set up the update preview window function.
setappdata(hImage,'UpdatePreviewWindowFcn',@mypreview_fcn);

% Make handle to text label available to update function.
setappdata(hImage,'HandleToTimestampLabel',hTextLabel);

preview(vid, hImage);
```

When you run this example, it creates the GUI shown in the following figure. Each time `preview` receives a video frame, it calls the update preview window function that you specified, which updates the timestamp text label in the GUI.



Custom Preview GUI with Timestamp Text Label

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. For information about connecting to an image acquisition device from a Simulink model, see Chapter 7, "Using the Image Acquisition Toolbox Block Library."

Getting Hardware Information (p. 3-2)	Describes how to get the information the toolbox needs to connect to a specific image acquisition device
Creating Image Acquisition Objects (p. 3-9)	Describes how to create the objects that the Image Acquisition Toolbox uses to establish the connection between MATLAB and an image acquisition device
Configuring Image Acquisition Object Properties (p. 3-17)	Describes how to modify characteristics of the acquisition using properties of the image acquisition objects
Starting and Stopping a Video Input Object (p. 3-24)	Describes how to start and stop a video input object
Deleting Image Acquisition Objects (p. 3-27)	Describes how to delete the image acquisition objects you create
Saving Image Acquisition Objects (p. 3-29)	Describes how to save an image acquisition object so that it can be loaded into the MATLAB workspace at a later time

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
- The video format of the video stream or, optionally, a device configuration file (camera file)

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 "Supported Devices" on page 2-3.

Adaptor Name	Description
`'coreco'`	Adaptor for image acquisition devices produced by Coreco Imaging, Inc.
`'dcam'`	Adaptor for IEEE 1394 (FireWire) image acquisition devices that comply with the IIDC 1394-based Digital Camera Specification (DCAM), developed by the Digital Camera subgroup of the 1394 Trade Association's Instrumentation and Industrial Control working group.
`'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 Windows Driver Model (WDM) or Video for Windows (VFW) driver, including USB and IEEE 1394 (FireWire) 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: '7.0.4 (R14SP2)'
          ToolboxName: 'Image Acquisition Toolbox'
       ToolboxVersion: '1.8 (R14SP2)'
```

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: [1x73 char]
    AdaptorDllVersion: '1.8 (R14SP2)'
          AdaptorName: 'matrox'
            DeviceIDs: {[1]}
           DeviceInfo: [1x1 struct]
```

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

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, you can use the device ID as a reference into the `DeviceInfo` structure array. 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.

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
`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

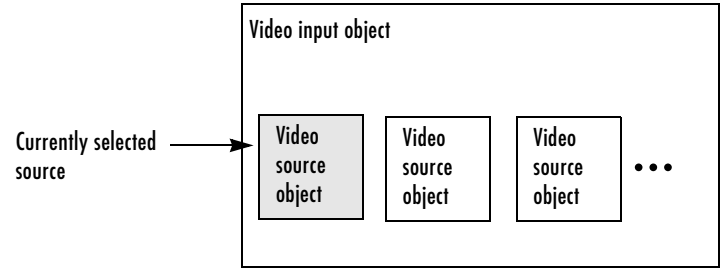
Video Input Objects

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 4-17.

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');
```

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 4-13 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.

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

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.
```

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.

Getting Information About a Video Input Object

After creating a video input object, you can get information about the device it represents using the `imaqhwinfo` function. When called with a video input object as an argument, `imaqhwinfo` returns a structure containing information about the object such as the name of the adaptor, name of the device, video resolution, and details of the vendor's device driver and version.

```
out = imaqhwinfo(vid)
out =

                AdaptorName: 'winvideo'
                 DeviceName: 'IBM PC Camera'
                  MaxHeight: 96
                   MaxWidth: 128
             NativeDataType: 'uint8'
               TotalSources: 1
    VendorDriverDescription: 'Windows WDM Compatible Driver'
        VendorDriverVersion: 'DirectX 9.0'
```

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: 0
```

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
   ReturnedColorSpace: [ {rgb} | grayscale | YCbCr ]
   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.

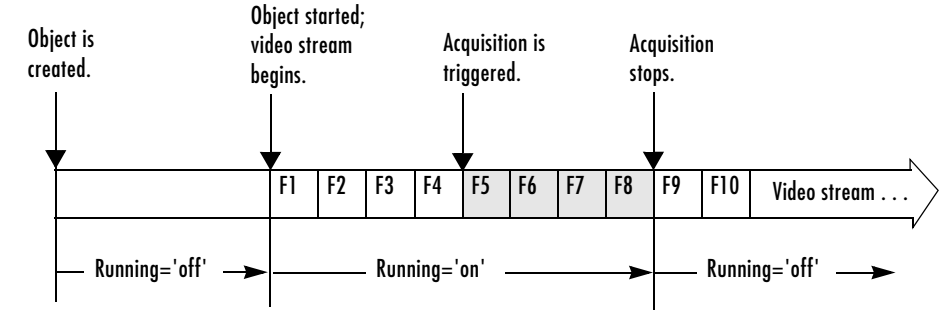
An image acquisition object stops running when any of the following conditions is met:

- 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. **Verify that the image is in a stopped state** — Use the `isrunning` function to determine the current state of the video input object.

   ```
   isrunning(vid)

   ans =

        0
   ```

3. **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')
   ```

   Configure an acquisition that takes several seconds so that you can see the video input in logging state.

   ```
   vid.FramesPerTrigger = 100;
   ```

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

   ```
   start(vid)
   ```

5. **Verify that the image is running but not logging** — Use the `isrunning` and `islogging` functions to determine the current state of the video input object. With manual triggers, the video input object is in running state after being started but does not start logging data until a trigger executes.

```
isrunning(vid)

ans =

     1

islogging(vid)

ans =

     0
```

6 **Execute the manual trigger** — Call the `trigger` function to execute the manual trigger.

```
trigger(vid)
```

While the acquisition is underway, check the logging state of the video input object.

```
islogging(vid)

ans =

     1
```

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

```
isrunning(vid)

ans =

     0
```

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
```

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 Image Acquisition Objects

This section describes two ways you can save a video input object:

Using the save Command

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 to their default values. To determine if a property is read only, use the `propinfo` function or read the property reference page.

Using the obj2mfile Command

Another way to save a video input object is to create an M-file that contains the set of commands used to create the video input object and configure its properties. You can use the `obj2mfile` function to create such an M-file. When you execute the M-file, it can create a new video input object or reuses an existing video input object, if one exists that has the same video format and adaptor.

4

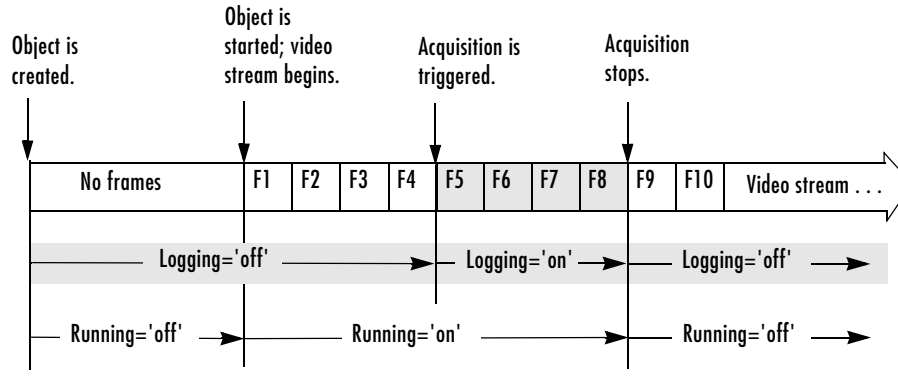
Acquiring Image Data

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

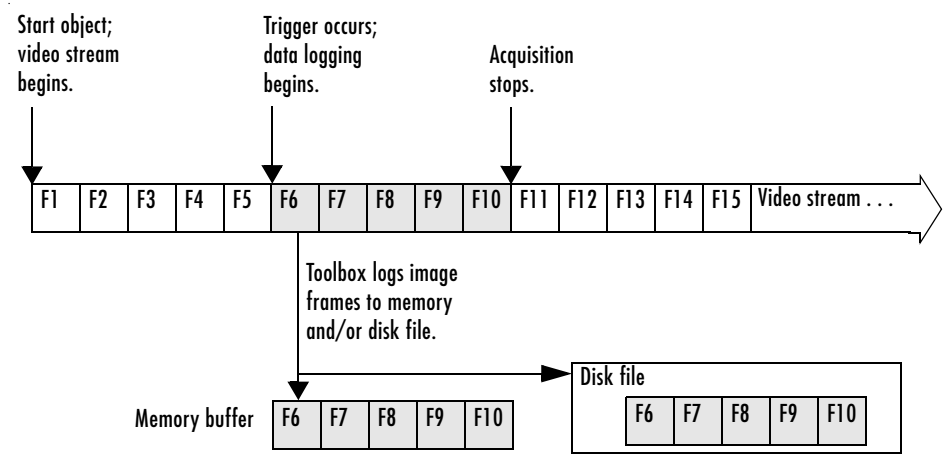
When a trigger occurs, the toolbox sets the object's `Logging` property to `'on'` and starts storing the acquired frames in a buffer in memory, a disk file, or both. When the acquisition stops, the toolbox sets the object's `Logging` property to `'off'`. The following figure illustrates when an object 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 4-35.

The following figure illustrates a group of frames being acquired from the video stream and being logged to memory and disk.



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.

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 4-25.
`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 4-26.
`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 4-13.

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

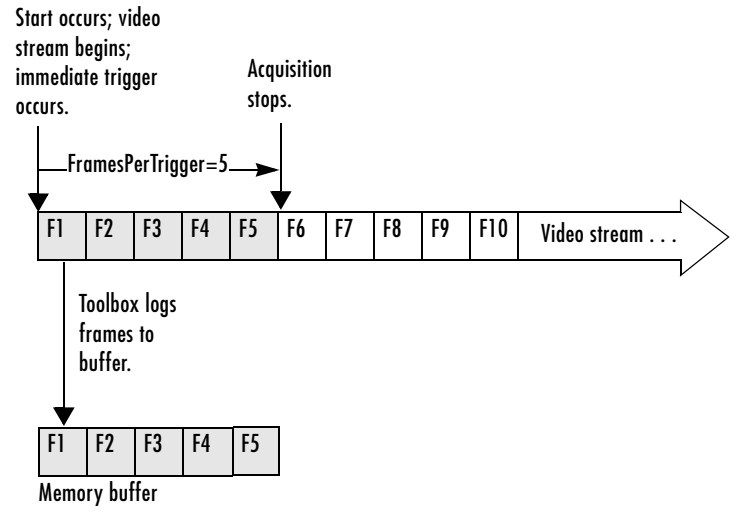
To specify the type of trigger you want to execute, set the value of the `TriggerType` property of the video input object. You must use the `triggerconfig` function to set the value of this property. The following table lists all the trigger types supported by the toolbox, with information about when 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 4-13. **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
```

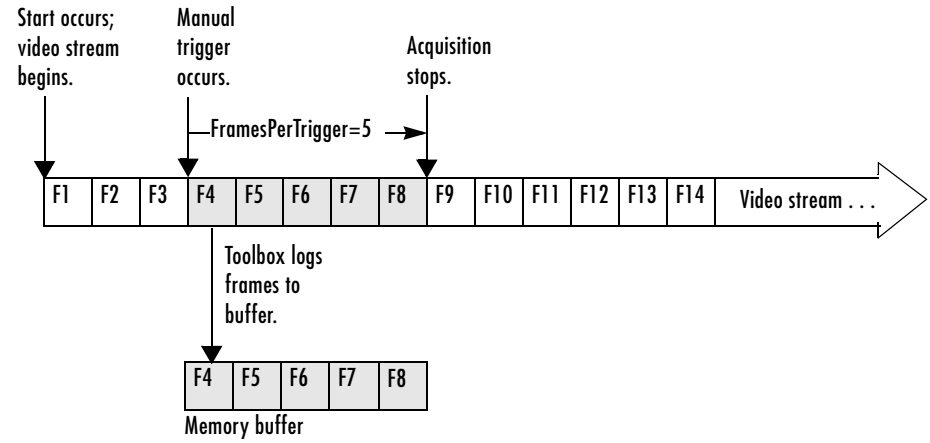
To execute another immediate trigger, you must restart the object. Note, however, that this deletes the data acquired by the first trigger. To execute multiple immediate triggers, specify a value for the `TriggerRepeat` property. See "Specifying Multiple Triggers" on page 4-26 for more information.

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.

```
isrunning(vid)

ans =

     1

islogging(vid)

ans =

     0
```

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.

```
isrunning(vid)

ans =

     0
```

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 4-26 for more information.

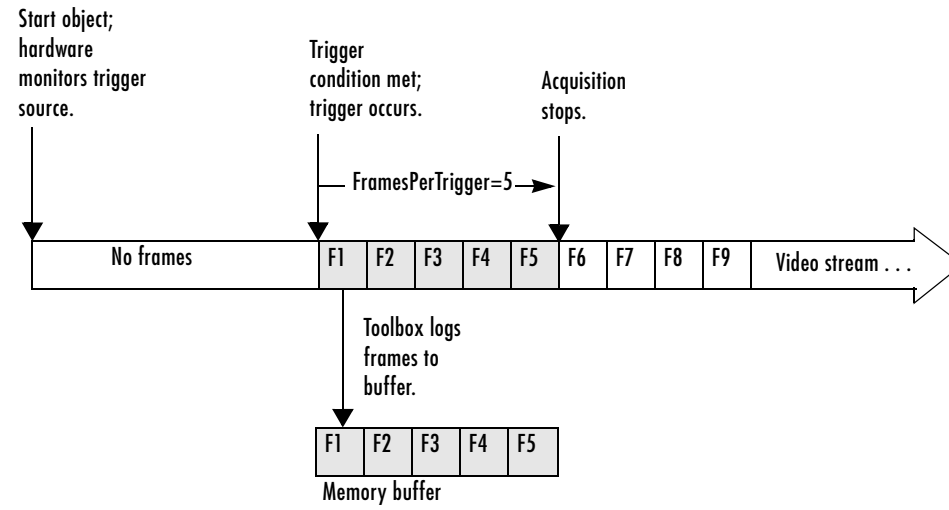
5 **Clean up** — Always remove image acquisition objects from memory, and the variables that reference them, when you no longer need them.

```
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.



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.

```
isrunning(vid)

ans =

     1

islogging(vid)

ans =

     0
```

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.

```
isrunning(vid)

ans =

     0
```

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 4-26 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'`.

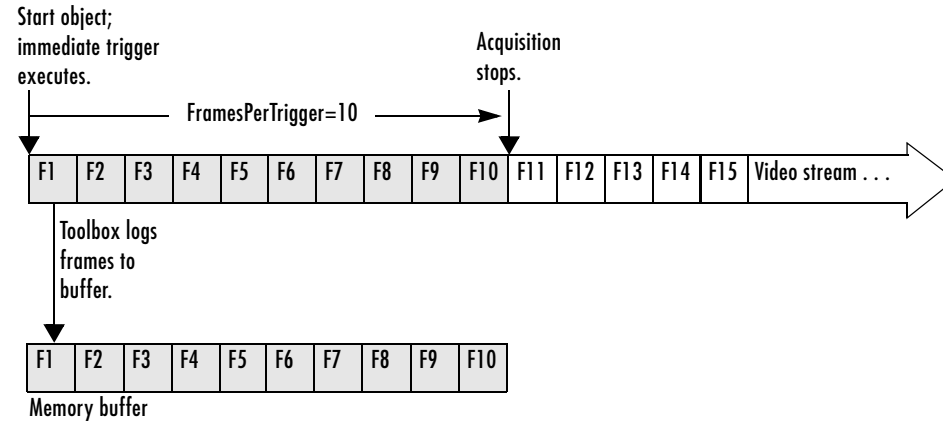
```
set(vid,'LoggingMode','disk&memory');
```

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 4-35.

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 4-20.



Specifying the Amount of Data to Log

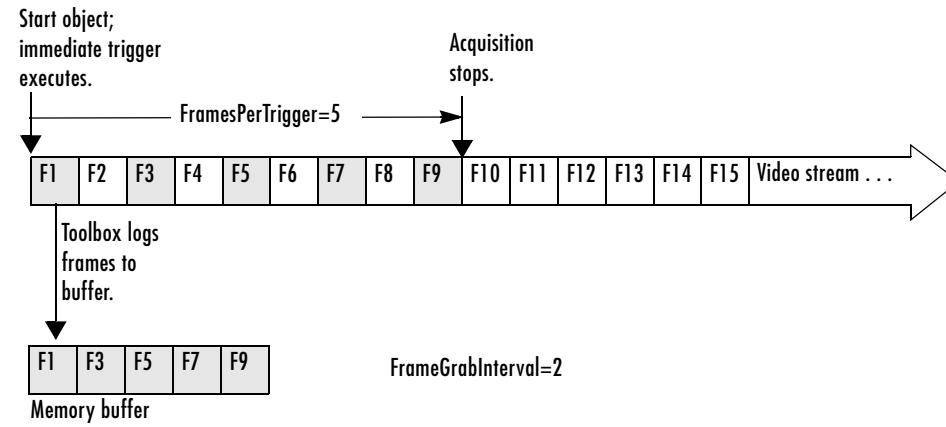
Note While you can specify any size acquisition, the number of frames you can acquire is limited by the amount of memory you have available on your system for image storage. A large acquisition can potentially fill all available system memory. For large acquisitions, you might want to remove frames from 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 4-32.

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 4-21.

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. To verify if the video input object is logging data, use the `islogging` function.

```
islogging(vid)
ans =

     1
```

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

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 4-20. 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 4-35 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.

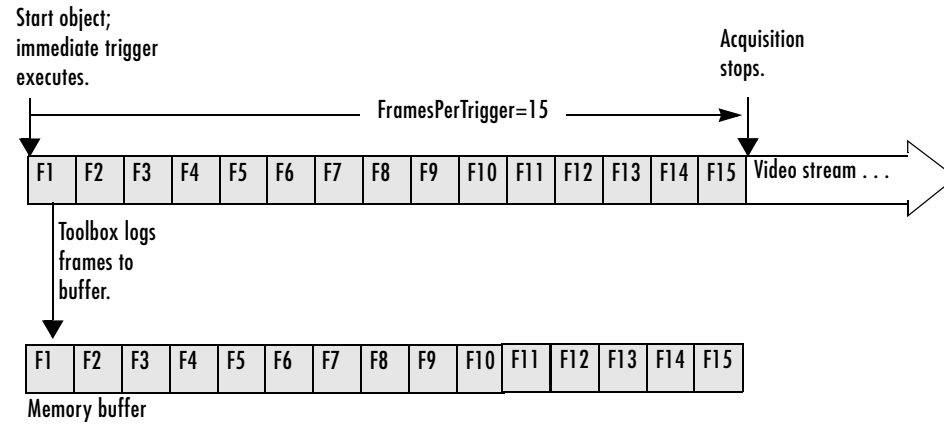
   ```
   start(vid)
   ```

   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.

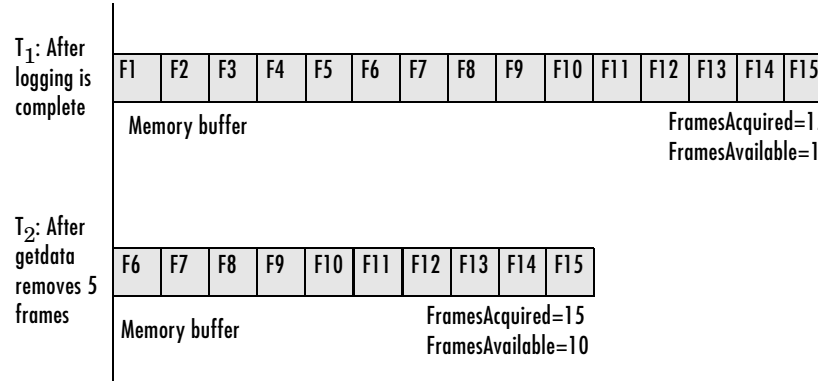
To remove frames from the memory buffer, call the `getdata` function, specifying the number of frames to retrieve. For more information about using `getdata`, see "Bringing Image Data into the MATLAB Workspace" on page 5-3.

```
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.



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

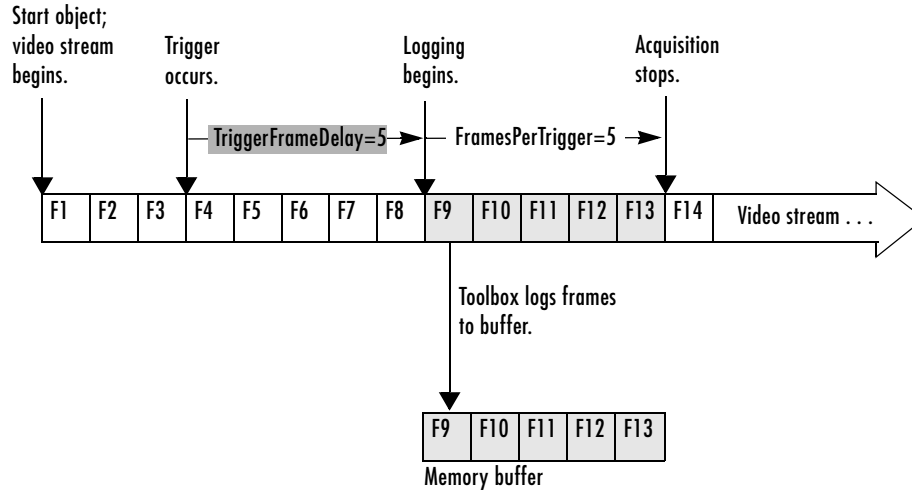
In some image acquisition setups, you might not want to log the first few frames returned from your camera or other imaging device. For example, some cameras require a short warmup time when activated. The quality of the first few images returned by these cameras might be too dark to be useful for your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

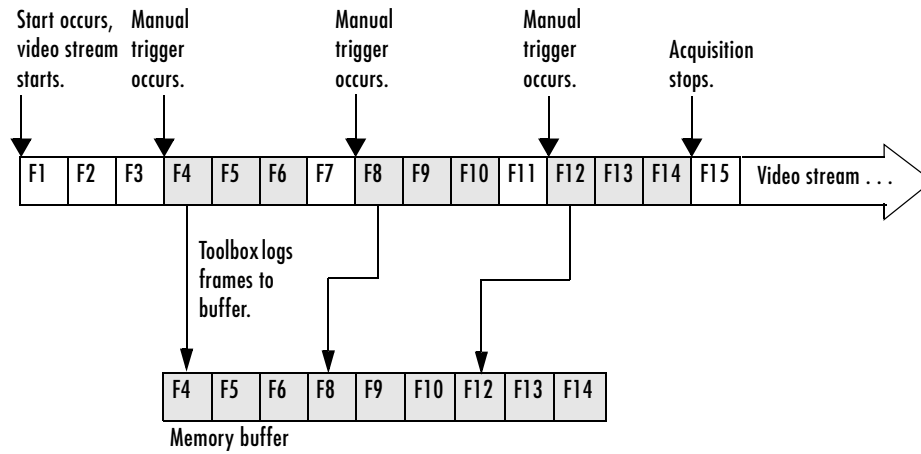
When a trigger occurs, a video input object acquires the number of frames specified by the `FramesPerTrigger` property and logs the data to a memory buffer, a disk file, or both.

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

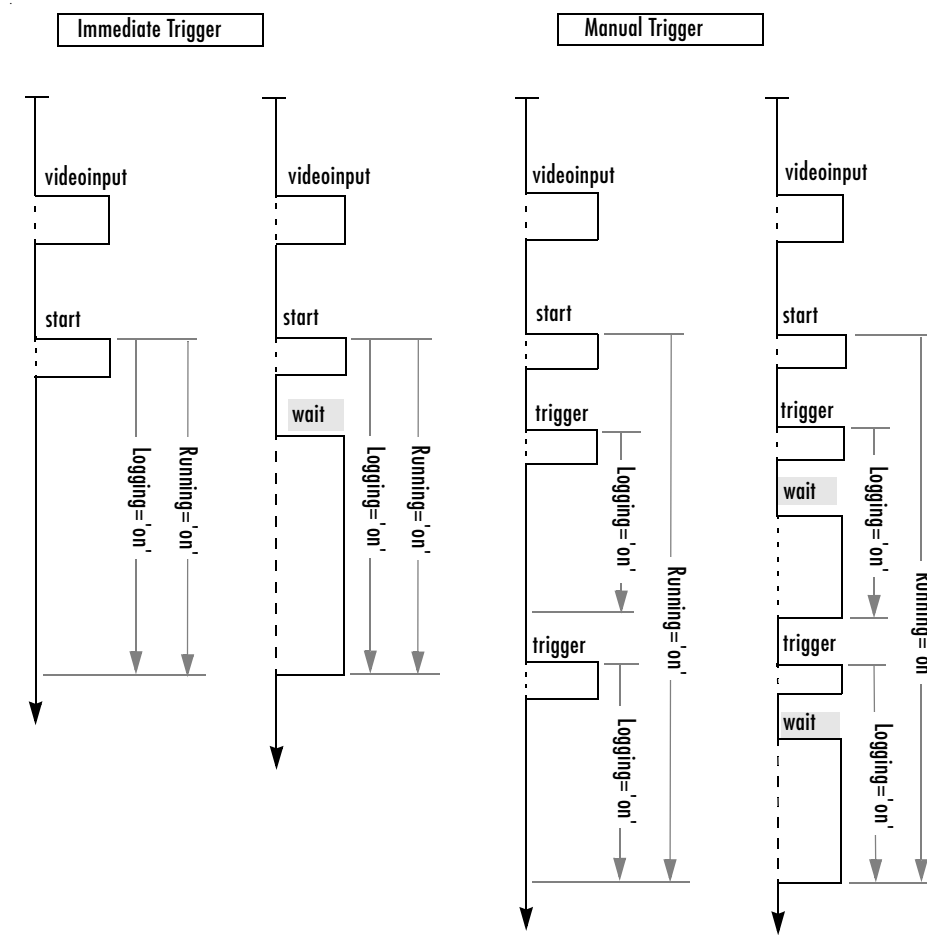
Waiting for an Acquisition to Finish

The `start` function and the `trigger` function are asynchronous functions. That is, they start the acquisition of frames and return control to the MATLAB command line immediately.

In some scenarios, you might want your application to wait until the acquisition completes before proceeding with other processing. To do this, call the `wait` function immediately after the `start` or `trigger` function returns. The `wait` function blocks the MATLAB command line until an acquisition completes or a timeout value expires, whichever comes first.

By default, `wait` blocks the command line until a video input object stops running. You can optionally specify that `wait` block the command line until the object stops logging. For acquisitions using an immediate trigger, video input objects always stop running and stop logging at the same time. However, with a manual trigger configured for multiple executions (`TriggerRepeat` > 0), you can use `wait` immediately after each call to the `trigger` function to block the command line while logging is in progress, even though the object remains in running state throughout the entire acquisition.

The following figure illustrates the flow of control at the MATLAB command line for a single execution of an immediate trigger and a manual trigger, with and without the `wait` function. A hardware trigger is similar to the manual trigger diagram, except that the acquisition is triggered by an external signal to the camera or frame grabber board, not by the `trigger` function. For an example, see “Example: Blocking the Command Line Until an Acquisition Completes” on page 4-28.



Using wait to Block the MATLAB Command Line

Example: Blocking the Command Line Until an Acquisition Completes

The following example illustrates how to use the `wait` function to put a 60 second time limit on the execution of a hardware trigger. If the hardware trigger does not execute within the time limit, `wait` returns control to the MATLAB command line.

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 a hardware trigger** — Use the `triggerinfo` function to determine 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'
```

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));
```

3 **Configure other object properties** — This example also sets the value of the `FramesPerTrigger` property to configure an acquisition large enough to produce a noticeable duration. (The default is 10 frames per trigger.)

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

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

```
start(vid)
```

The `start` function sets the object running and returns control to the command line.

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

```
wait(vid,60)
```

The `wait` function blocks the command line until the hardware trigger fires and acquisition completes or until the amount of time specified by the timeout value expires.

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
```

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. By default, the toolbox sets this value, called the *frame memory limit*, to equal 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

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;
```

To see an example of using a callback function to monitor memory usage, see "Example: Monitoring Memory Usage" on page 6-17.

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. The following example illustrates this process.

1 **Determine the current frame memory limit** — This example calls the `imaqmem` function, requesting the value of the `FrameMemoryLimit` field.

```
out = imaqmem('FrameMemoryLimit')

out =

    15425536
```

2 **Set the frame memory limit to a new value** — When you call `imaqmem` with a numeric argument, it sets the `FrameMemoryLimit` field to the value.

```
imaqmem(36000000)
```

3 **Verify the frame memory limit setting** — Call `imaqmem` again, requesting the value of the `FrameMemoryLimit` field.

```
out = imaqmem('FrameMemoryLimit')

out =

    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.

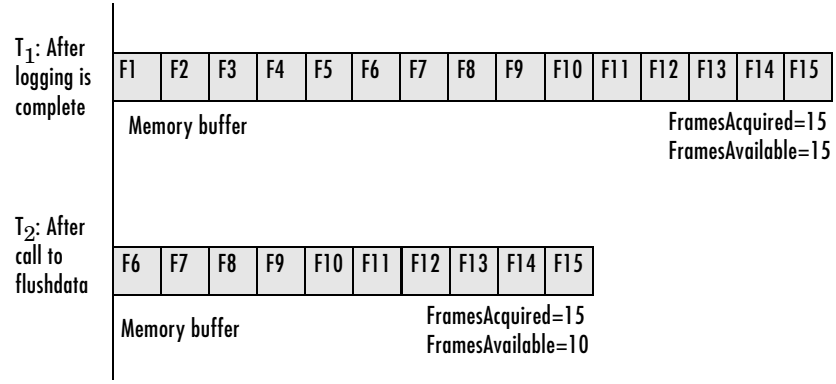
To verify that the object acquired data, view the value of the `FramesAvailable` property. This property reports how many frames are currently stored in the memory buffer.

```
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.



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.

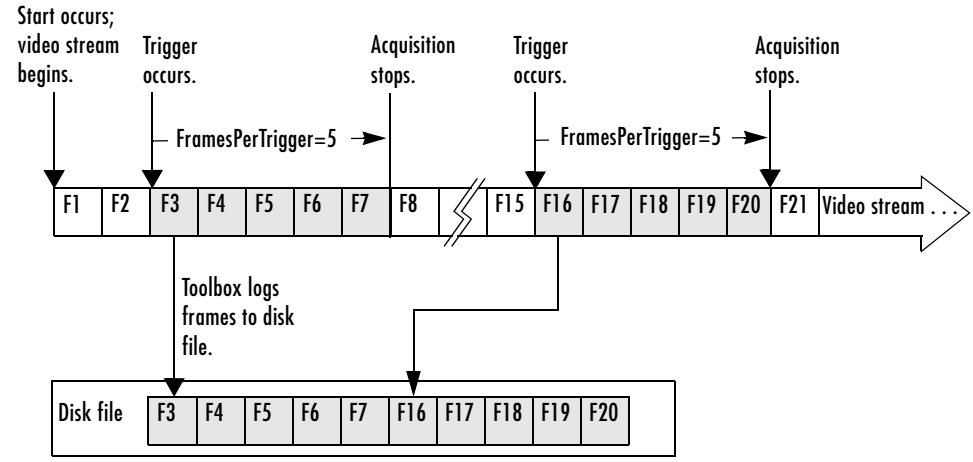
To set up data logging to disk, perform these steps:

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 4-36.

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 4-38.



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 more information about AVI file objects, see the MATLAB `avifile` documentation. For more information about using AVI files to log image data, see the following topics.

- "Logging Grayscale Images" on page 4-37
- "Guidelines for Using an AVI File Object to Log Image Data" on page 4-37
- "Closing the DiskLogger AVI file" on page 4-37

Logging Grayscale Images

When logging images in grayscale format, such as RS170, you must set the value of the AVI object's `Colormap` property to be a grayscale colormap. Otherwise, the image data in the AVI file will not display correctly.

This example uses the MATLAB `gray` function to create a grayscale colormap and sets the value of the AVI file object's `Colormap` property with this colormap.

```
logfile = avifile('my_datalog.avi','Colormap',gray(256));
```

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;
```

Because this example acquires image data in grayscale format (RS170), you must also specify the colormap used with the AVI object to ensure that the stored data displays correctly.

```
aviobj.Colormap = gray(256);
```

3 **Create a video input object** — This example creates a video input object for a Matrox image acquisition device, using the default video format `M_RS170`. 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

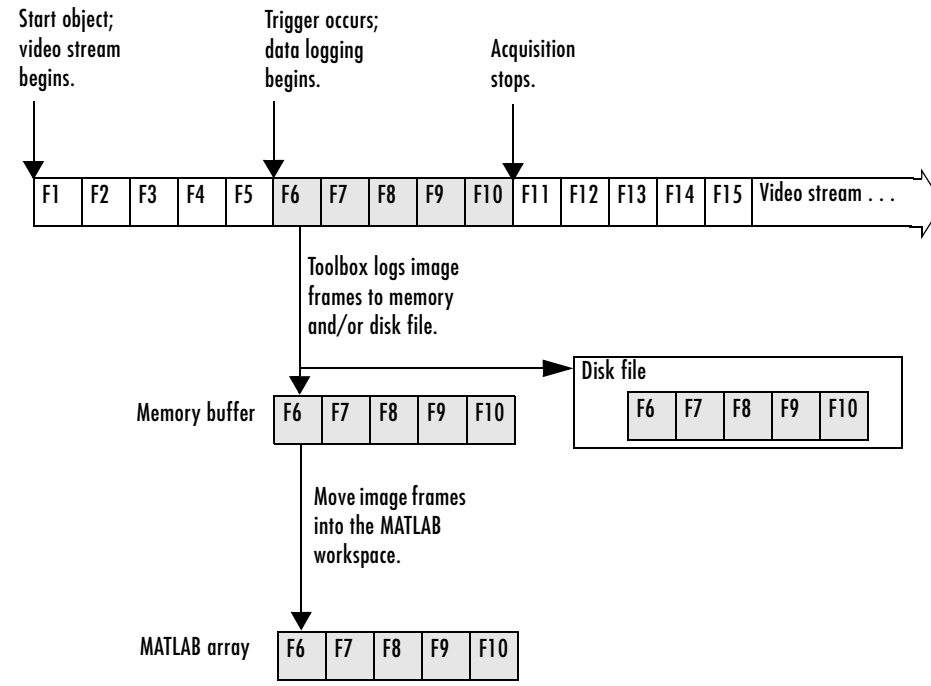
When you trigger an acquisition, the toolbox stores the image data in a memory buffer, a disk file, or both. To work with this data, you must bring it into the MATLAB workspace.

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:

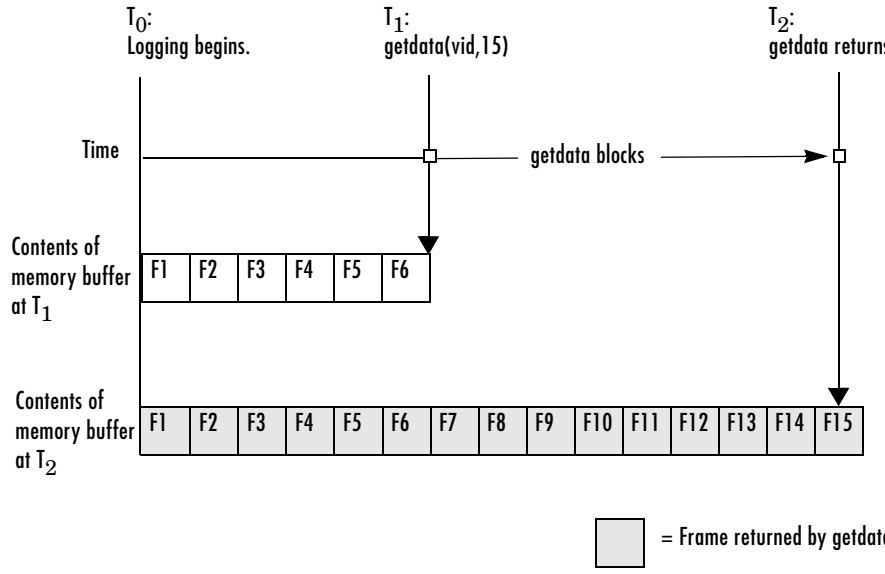
- **Removing multiple frames from the buffer** — To move a specified number of frames from the memory buffer into the workspace, use the `getdata` function. The `getdata` function removes the frames from the memory buffer as it moves them into the workspace. The function blocks the MATLAB command line 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.
- **Bringing 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 When the `getdata` function moves frames from the memory buffer into the workspace, it removes the frames from the memory buffer.

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(300) - 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
```

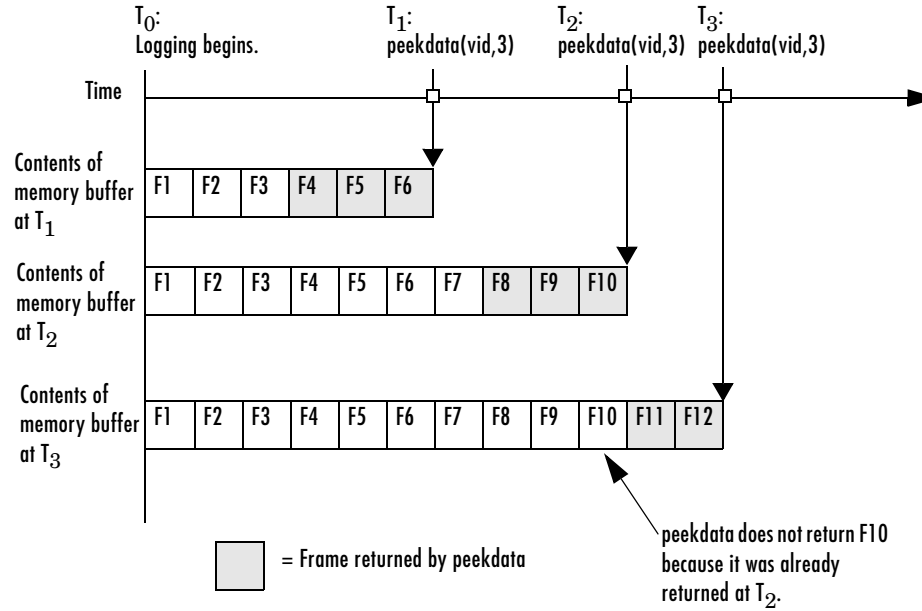
Viewing Frames in the Memory Buffer

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.



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.

```
isrunning(vid)

ans =

     1

islogging(vid)

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
```

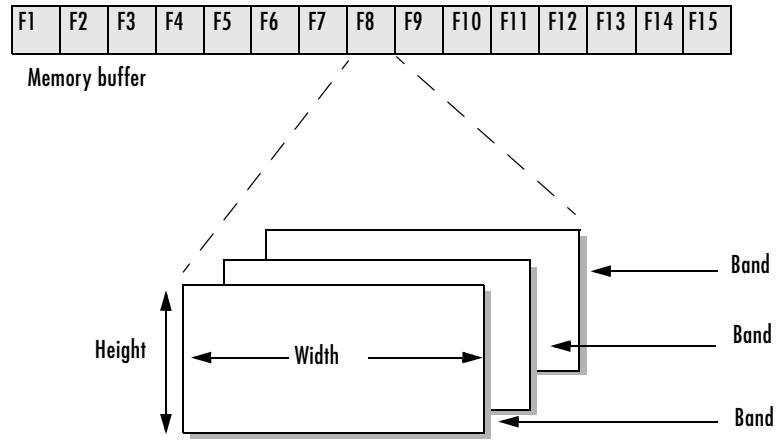
Note that the frame returned by `getsnapshot` is not removed from the memory buffer, if frames are stored there, and does not affect the value of the `FramesAvailable` property.

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.

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.

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.

ROIs and Image Dimensions

When you specify 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.

Example: Video Format and Image Dimensions

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 *n*th 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, you can specify that `getdata` return image frames as an array of class `double`. To see a list of all the data types supported, see the `getdata` reference page.

The following example illustrates the data type of returned image data.

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
```

Specifying the Color Space

For most image acquisition devices, the video format of the video stream determines the color space of the acquired image data, that is, the way color information is represented numerically.

For example, 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. Another color space, widely used for digital video, is the YCbCr color space. In this color space, 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.

The toolbox can return image data in grayscale, RGB, and YCbCr. To specify the color representation of the image data, set the value of the `ReturnedColorSpace` property. To display image frames using the `image` or `imagesc` functions, the data must use the RGB color space. Another MathWorks product, the Image Processing Toolbox, includes functions that convert YCbCr data to RGB data, and vice versa.

Note Some devices that claim to support the YUV color space actually support the YCbCr color space. YUV is similar to YCbCr but not identical. The difference between YUV and YCbCr is the scaling factor applied to the result. YUV refers to a particular scaling factor used in composite NTSC and PAL formats. In most cases, you can specify the YCbCr color space for devices that support YUV.

The following example illustrates how to specify the color space of the returned image data.

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 default color space used for the data** — The value of the `ReturnedColorSpace` property indicates the color space of the image data.

   ```
   vid.ReturnedColorSpace

   ans =

   rgb
   ```

3. **Modify the color space used for the data** — To change the color space of the returned image data, set the value of the `ReturnedColorSpace` property.

   ```
   set(vid,'ReturnedColorSpace','grayscale')

   ans =

   grayscale
   ```

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
```

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,

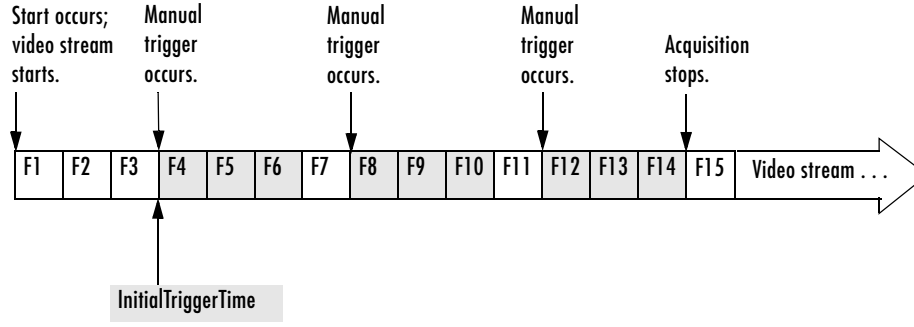
- Determining when a trigger executed
- Determining when a particular frame was acquired

To see an example of retrieving timing information, see "Example: Determining the Frame Delay Duration" on page 5-22.

Determining When a Trigger Executed

To determine when 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 6-7.

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 5-22.

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 6-7 for more information.

Frame Metadata

Field Name	Description
`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
```

6

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 continues running. For example, you can display a message, analyze data, or perform other tasks. The start and stop callbacks, however, execute synchronously; the object does not perform any further processing until the callback function finishes.

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 6-12.

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 6-7.

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. When a start event occurs, the toolbox executes the function specified by the `StartFcn` property. **Note**: The `StartFcn` callback executes synchronously. If you specify a `StartFcn` callback function, the toolbox waits for the function to finish executing before performing any other processing. If an error occurs in the start callback function, the object never starts.
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. **Note**: The `StopFcn` callback executes synchronously. If you specify a `StopFcn` callback function, the toolbox waits for the function to finish executing before performing any other processing.

Events and Callback Function Properties (Continued)

Event	Callback Property	Description
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 6-12.

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 6-8
- "Data Fields for Error Events" on page 6-8
- "Data Fields for Timer Events" on page 6-9

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 6-7.

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 6-12.

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: [2004 12 29 16 40 52.5990]
    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 6-7.

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 6-14.

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 6-16.

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};
```

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 6-13.

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 6-13. 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 6-17, 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
```

7

Using the Image Acquisition Toolbox Block Library

The Image Acquisition Toolbox includes a block that can be used in Simulink to bring live video data into models.

Overview

This chapter describes how to use the Image Acquisition Toolbox block library. The toolbox block library contains one block called the Video Input block. You can use this block to acquire live video data in a Simulink model. You can interconnect this block with blocks in other Simulink libraries to create sophisticated models.

To use the Image Acquisition Toolbox Video Input block requires Simulink, a tool for simulating dynamic systems. Simulink is a model definition environment. Use Simulink blocks to create a block diagram that represents the computations of your system or application. Simulink is also a model simulation environment. Run the block diagram to see how your system behaves. If you are new to Simulink, read the Getting Started section of the Simulink documentation to better understand its functionality.

Topics covered include:

For more detailed information about the block in the Image Acquisition toolbox, see the reference page for the Video Input block in Chapter 12, "Block Reference."

Opening the Block Library

There are several ways to open the Image Acquisition Toolbox block library:

- "Using the imaqlib Command" on page 7-3
- "Using the Simulink Library Browser" on page 7-3

Using the imaqlib Command

To open the Image Acquisition Toolbox block library, enter

```
imaqlib
```

at the MATLAB prompt. MATLAB displays the contents of the library in a separate window.

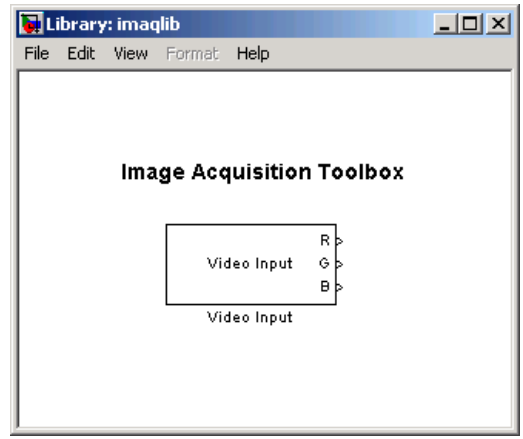


Image Acquisition Toolbox Block Library

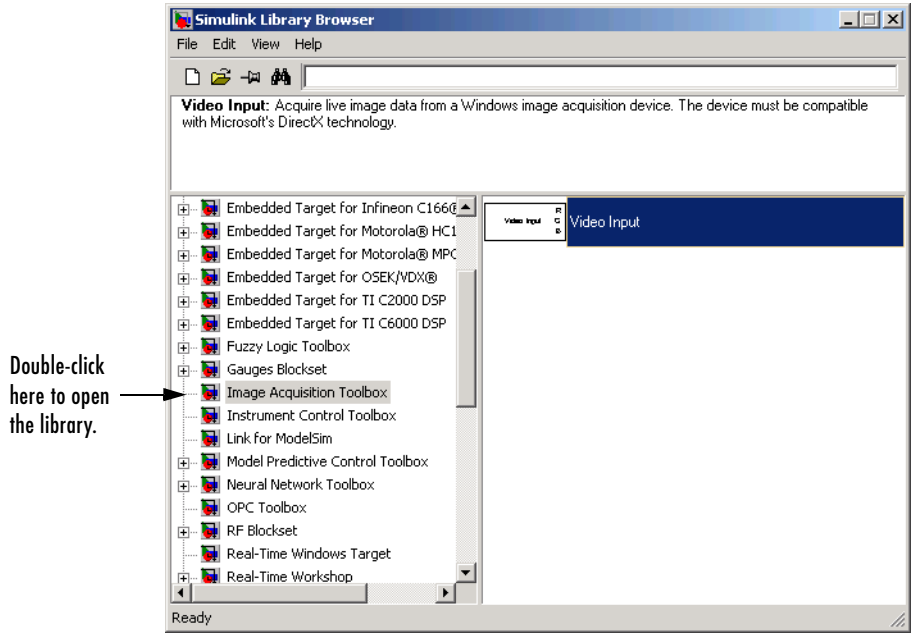
Using the Simulink Library Browser

To open the Image Acquisition Toolbox block library, start the Simulink Library Browser and choose the library from the list of available block libraries displayed in the browser.

To start the Simulink Library Browser, enter

```
simulink
```

at the MATLAB prompt. MATLAB opens the browser window. The left pane contains a list of available block libraries in alphabetical order. To open the Image Acquisition Toolbox block library, double-click its icon.



Example: Saving Video Data to a File

The best way to learn about the Image Acquisition Toolbox Video Input block is to see an example. This section provides a step-by-step example that builds a simple model using the block in conjunction with blocks from other blockset libraries. Steps described include

Step 1: Open the Image Acquisition Toolbox Library

To use the Video Input block, you must open the Image Acquisition Toolbox block library. To open the library, start the Simulink Library Browser and choose the Image Acquisition Toolbox entry from the list displayed in the browser.

To start the Simulink Library Browser, enter

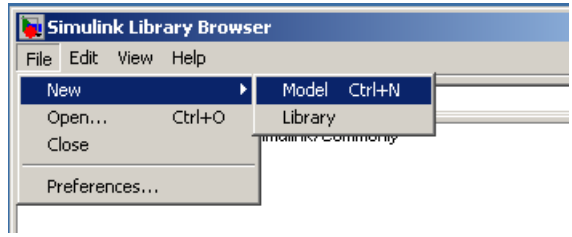
```
simulink
```

at the MATLAB prompt. (For more information about opening the library, see “Opening the Block Library” on page 7-3.)

Step 2: Open a Model or Create a New Model

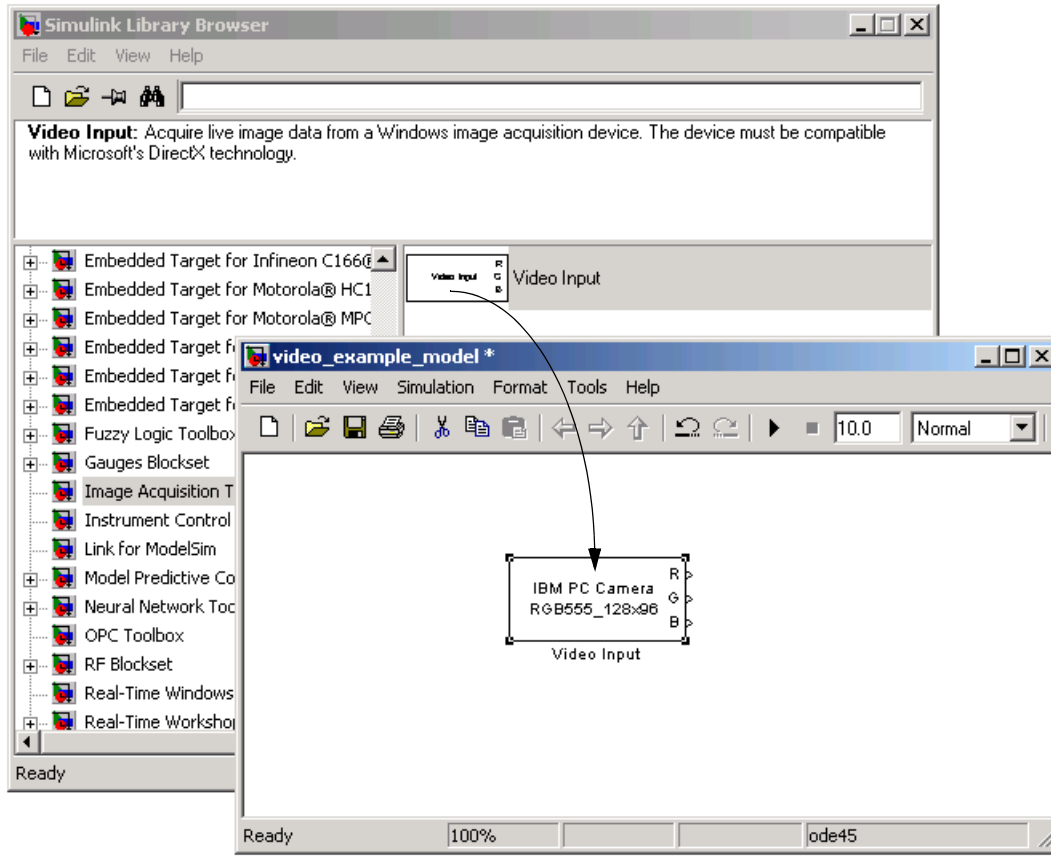
To use a block, you must add it to an existing model or create a new model.

To create a new model, click the **File** menu in the Simulink Library Browser and select the **New** option. From this menu, choose **Model**. Simulink opens an empty model window on the display. To assign the new model a name, use the **Save** option.



Step 3: Drag the Video Input Block into the Model

To use the Video Input block in a model, click the block in the library and, holding the mouse button down, drag it into the model window. Note how the name on the block changes to reflect the camera connected to your system.

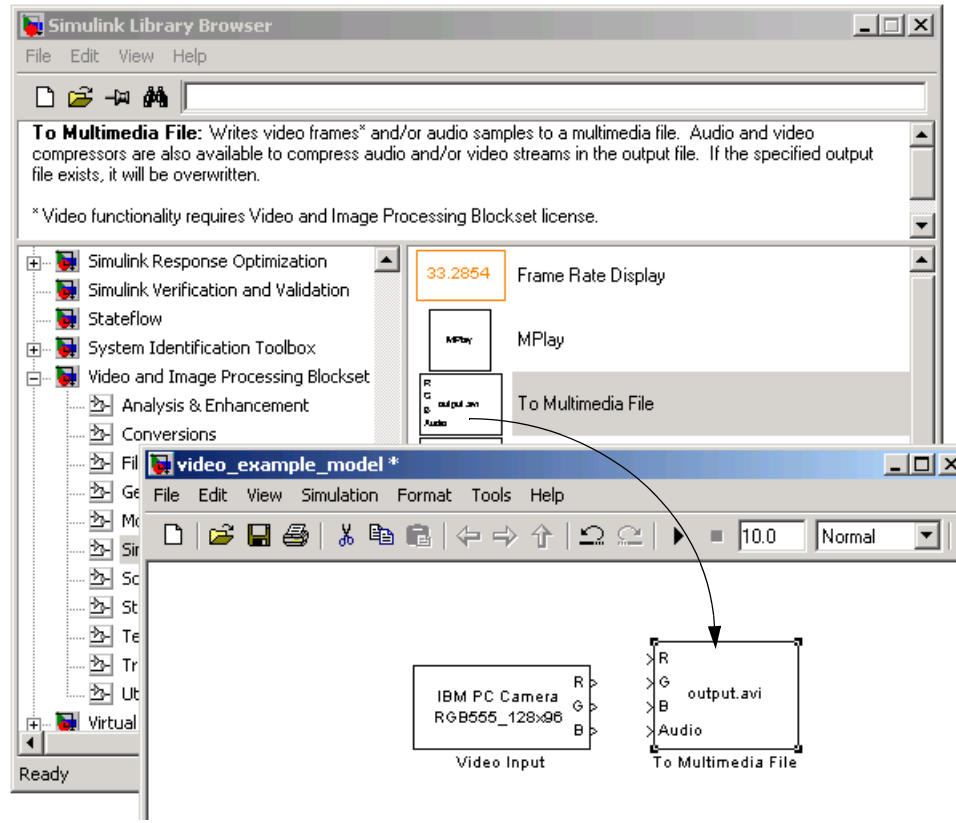


Drag Video Input Block into Model Window

Step 4: Drag Other Blocks to Complete the Model

To illustrate using the block, this example creates a simple model that acquires data and then outputs the data to a file in Audio Video Interleave (AVI) format. To create this model, the example uses a block from the Video and Image Processing Blockset.

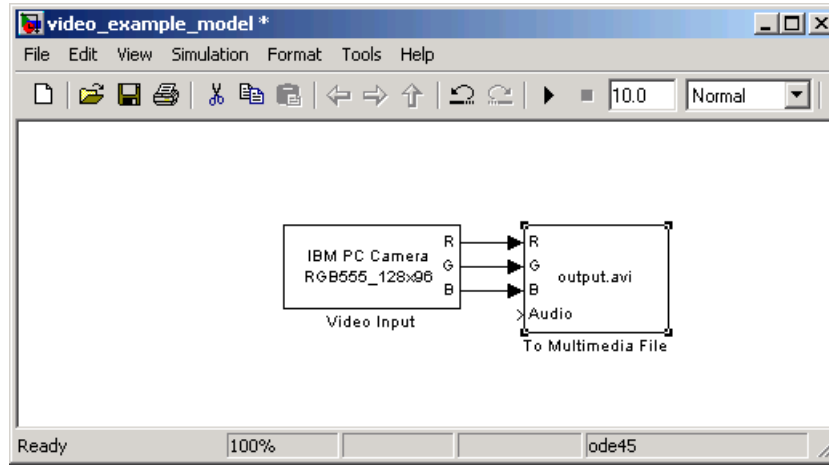
Open the Video and Image Processing Blockset library. In the library window, open the Sinks subsystem. From this subsystem, click the To Multimedia File block in the library and, holding the mouse button down, drag the block into the model window.



Drag Output Block to Model Window

Step 5: Connect the Blocks

Connect the three outputs from the Video Input block to the three corresponding inputs on the To Multimedia File block. (You can leave the Audio input on the To Multimedia File block unconnected.) One quick way to make all three connections at once is to select the Video Input block, press and hold the Control key, and then click the To Multimedia File block.

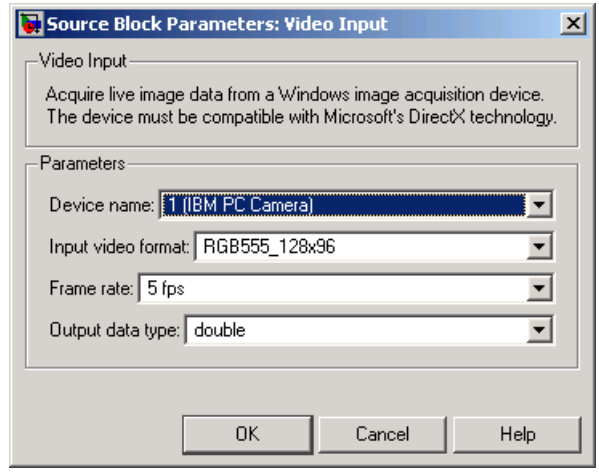


Connect the Video Input Block to the To Multimedia File Block

Step 6: Specify Video Input Block Parameter Values

To check Video Input block parameter settings, double-click the block's icon in the model window. This opens the **Block Parameters** dialog box for the Video Input block, shown in the following figure. You use the various menus in the dialog box to determine the current values of Video Input block parameters or change the values.

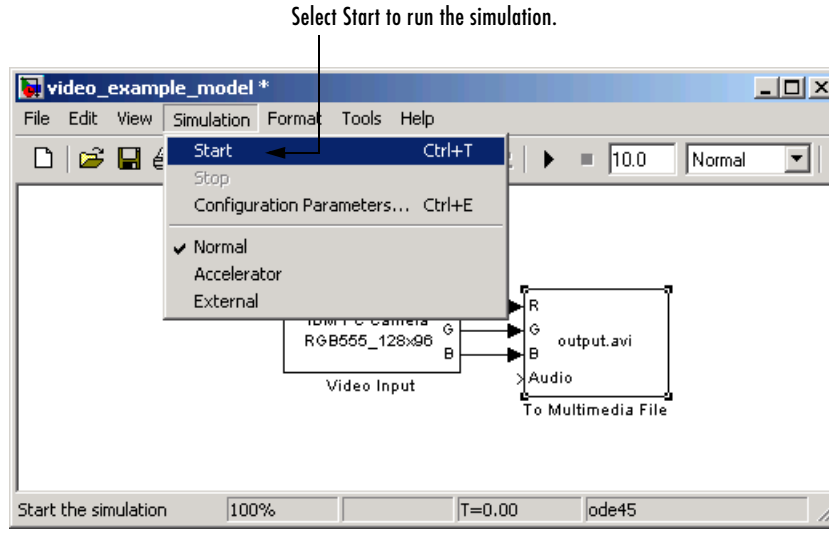
For example, using this dialog box, you can specify the device you want to use, select the video resolution you want to use with the device, or specify the rate at which frames are acquired. For more details, see the "Video Input" block reference page.



You can set parameters for any of the blocks you include in your model. For example, to specify the name of the AVI file, double-click the To Multimedia File block. Make sure that you have write permission to the directory into which the block writes the AVI file.

Step 7: Run the Simulation

To run the simulation, click the **Simulation** menu in the model window and choose the **Start** option. You can also use the **Start** button on the model window toolbar. You can use toolbar options to specify how long to run the simulation and to stop a running simulation.



While the simulation is running, the status bar at the bottom of the model window indicates the progress of the simulation. After the simulation finishes, check the directory in which you ran the simulation to verify that an AVI file was created.

8

Adding Support for Additional Hardware

The Image Acquisition Toolbox supports connections with hardware from many common vendors but it might not support the hardware you use. To add support for your hardware, you can create an adaptor using the Image Acquisition Toolbox Adaptor Kit.

Overview

The Image Acquisition Toolbox Adaptor Kit is a C++ framework that you can use to implement an adaptor. An adaptor is a dynamic link library (DLL) that implements the connection between the Image Acquisition Toolbox engine and a device driver via the vendor's SDK API. When you use the Adaptor Kit framework, you can take advantage of many prepackaged toolbox features such as disk logging, multiple triggering modes, and a standardized interface to the image acquisition device.

After you create your adaptor DLL and register it with the toolbox using the `imaqregister` function, you can create a video input object to connect with a device through your adaptor. In this way, adaptors enable the dynamic loading of support for hardware without requiring recompilation and linking of the toolbox.

To build an adaptor requires familiarity with C++, knowledge of the application programming interface (API) provided by the manufacturer of your hardware, and familiarlity with Image Acquisition Toolbox concepts, functionality, and terminology. To learn more about creating an adaptor, read the Image Acquisition Toolbox Adaptor Kit User's Guide. For detailed information about the adaptor kit framework classes, see the *Image Acquisition Toolbox Adaptor Kit Class Reference*, which is available in

```
<matlabroot>\toolbox\imaq\imaqadaptors\kit\doc\adaptorkit.chm
```

where `<matlabroot>` represents your MATLAB installation directory.

9

Troubleshooting

This chapter provides information about solving common problems you might encounter with the Image Acquisition Toolbox and the video acquisition hardware it supports.

Overview

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 the vendors and categories of devices supported by the Image Acquisition Toolbox.

- Coreco Imaging, Inc.
- Data Translation, Inc.
- Digital cameras that support the IIDC 1394-based Digital Camera (DCAM) Specification, developed by the 1394 Trade Organization
- Matrox, Inc.
- Generic Windows video acquisition devices

If you are encountering problems with the preview window, see "Troubleshooting a Video Preview Window" on page 9-18.

Troubleshooting Coreco Hardware

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

1. Verify that your image acquisition hardware is functioning properly.

   For Coreco devices, run the application that came with your hardware, the IFC Camera Configurator, and verify that you can view a live video stream from your camera.

2. Verify that the toolbox can locate your camera file, if you are using a camera file to configure the device. Make sure that your camera file appears in the **List of Cameras** in the Coreco IFC Camera Configurator.

3. 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 Coreco hardware and is not guaranteed to work with any other versions.

   - Find out the driver version you are using on your system. To learn how to get this information, see "Determining the Driver Version for Coreco Devices" on page 9-4.
   - Verify that the version is compatible with the Image Acquisition Toolbox. For the latest driver information, check 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 Coreco Web site (`www.imaging.com`) to download the latest drivers.

Determining the Driver Version for Coreco Devices

To determine the Coreco IFC Library version you are using, view the release notes for the driver. You can access the release notes through the Windows **Start** menu.

1. Click the **Start** button.
2. On the **Start** menu, select **Programs**.
3. From the **Programs** menu, select the **IFC** link.
4. On the **IFC** menu, select the IFC release notes.

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.
   - Find out the driver version you are using on your system.
   - Verify that the version is compatible with the Image Acquisition Toolbox. For the latest driver information, check 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.

3. Install the Data Translation Software Development Kit (SDK).

   If the `imaqhwinfo` function does not return the driver for a Data Translation frame grabber, or the `imaqhwinfo` function or `videoinput` functions return an error message about a missing DLL (`olfg32.dll`), you may need to install additional files from the Imaging Omni CD.

   By default, when you install drivers for your Data Translation frame grabber, the installation program may not install all the files the device drivers need. The additional files needed by the device driver are part of the SDK installation, not the device driver installation. If you get error messages about missing files, insert the Imaging Omni CD into your CD-ROM drive and install the SDK.

Troubleshooting DCAM IEEE 1394 (FireWire) Hardware

If you are having trouble using the Image Acquisition Toolbox with an IEEE 1394 (FireWire) camera, using the toolbox's `dcam` adaptor, perform these troubleshooting steps, in the order specified.

1. Verify that your IEEE 1394 (FireWire) camera is plugged into the IEEE 1394 (FireWire) port on your computer and is powered up.

2. Verify that your IEEE 1394 (FireWire) camera can be accessed through the `dcam` adaptor.
   - Make sure the camera is compliant with the IIDC 1394-based Digital Camera (DCAM) specification. Vendors typically include this information in documentation that comes with the camera. If your digital camera is not DCAM compliant, you might be able to use the `winvideo` adaptor. See "Troubleshooting Windows Video Hardware" on page 9-15 for information.
   - Make sure the camera outputs data in uncompressed format. Cameras that output data in Digital Video (DV) format, such as digital camcorders, cannot use the `dcam` adaptor. To access these devices, use the `winvideo` adaptor. See "Troubleshooting Windows Video Hardware" on page 9-15 for information.
   - Make sure you specified the `dcam` adaptor when you created the video input object. Some IEEE 1394 (FireWire) cameras can be accessed through either the `dcam` or `winvideo` adaptors. If you can connect to your camera from the toolbox but cannot access some camera features, such as hardware triggering, you might be accessing the camera through a DirectX driver. See "Creating a Video Input Object" on page 3-10 for more information about specifying adaptors.

3. Verify that your IEEE 1394 (FireWire) camera is using the Carnegie Mellon University (CMU) DCAM driver.

Note The toolbox only supports connections to IEEE 1394 (FireWire) DCAM-compliant devices using the CMU DCAM driver. The toolbox is not compatible with any other vendor-supplied driver, even if the driver is DCAM compliant.

To verify this, run the demo application provided by CMU, `1394CameraDemo.exe`. This demo application is among the files you extract from the CMU driver archive file when you install the CMU DCAM driver — see "Installing and Configuring the CMU DCAM Driver" on page 9-8. To learn how to run the demo application, see "Running the CMU Camera Demo Application" on page 9-10.

- If the demo application recognizes the camera, the camera is set up to use the CMU DCAM driver and is ready for use by the toolbox.
- If the demo application does not recognize the camera, install the CMU DCAM driver. See "Installing and Configuring the CMU DCAM Driver" on page 9-8 for instructions.
- If the demo application recognizes your camera, but the toolbox still does not, verify that the camera complies with the minimum DCAM specification version for the camera and the minimum DCAM CMU driver version required by the toolbox. For the latest information about supported hardware, visit the Image Acquisition Toolbox product page at The MathWorks Web site (`www.mathworks.com/products/imaq`). To learn how to get version information, see "Determining the DCAM Driver Version" on page 9-10.

Installing and Configuring the CMU DCAM Driver

The Image Acquisition Toolbox supports acquiring data from IEEE 1394 (FireWire) cameras that support the IIDC 1394-based Digital Camera (DCAM) specification. To use a DCAM compliant camera, you must use the DCAM driver created by Carnegie Mellon University (CMU) to connect to these devices. This section describes

- "Installing the Driver" on page 9-8
- "Configuring the DCAM Driver" on page 9-9

Note The CMU DCAM driver is the only DCAM driver supported by the toolbox. You cannot use vendor-supplied drivers, even if they are compliant with the DCAM specification.

Installing the Driver

To install the CMU DCAM driver on your system, follow this procedure.

1. Obtain the CMU DCAM driver files. The Image Acquisition Toolbox includes the CMU DCAM archive file, `1394camera62.zip`, in the directory

   ```
   $MATLAB\toolbox\imaq\imaqextern\drivers\win32\dcam
   ```

   where `$MATLAB` represents the name of your MATLAB installation directory.

   You can also download the DCAM driver directly from CMU. Go to the Web site `www-2.cs.cmu.edu/~iwan/1394` and click the download link.

2. Extract the files from the archive.

   Double-click the CMU DCAM zip archive file, `1394camera62.zip`, and extract the files to any convenient directory. You will specify this directory name later when you use the Windows **Update Device Driver Wizard** during configuration. When unzipping the distribution, select the **Use folder names** option to extract the files into the correct directories.

Configuring the DCAM Driver

After extracting the files from the zip archive, follow these instructions to replace your camera's vendor-supplied driver with the CMU DCAM driver.

1. Make sure your camera is connected to the IEEE 1394 (FireWire) port on your computer.

2. Open the **System Properties** control panel to update the device driver for your IEEE 1394 camera. The exact method to open this control panel varies with different versions of the Windows operating system. One way to open the **System Properties** control panel is to right-click the **My Computer** icon on your desktop and select **Properties** from this menu.

3. In the **System Properties** dialog box, click the **Hardware** tab and then click the **Device Manager** button.

4. In the **Device Manager** dialog box, expand the **Imaging devices** entry and right-click the device you intend to use. In the menu displayed, select **Properties**.

5. In the **Properties** dialog box, click the **Driver** tab, and then click the **Update Drivers** button. This launches the **Update Device Driver Wizard.** When the wizard asks what you want to do, select the **Display a list of the known drivers** option. Do not select the **Search** option. When the wizard asks you to select the manufacturer and model of your hardware, click the **Have Disk** button. In the **Install From Disk** dialog box, specify the name of the directory in which you stored the driver files you extracted from the CMU archive, including the file named `1394camera.inf`.

Note You can ignore the warnings about the driver's not being digitally signed.

---

6. Click **Finish** to update the driver.

7. After configuration is complete, verify that your camera is now listed in the **Device Manager** as `CMU 1394 Digital Camera Device`.

8. Restart your computer to ensure that the camera uses the new driver.

Determining the DCAM Driver Version

To determine the version of the DCAM driver, go to the directory in which you extracted the CMU DCAM driver files and right-click the DCAM library file (`1394camera.dll`). In the context menu, select **Properties**. In the **Properties** control panel, click the **Version** tab.

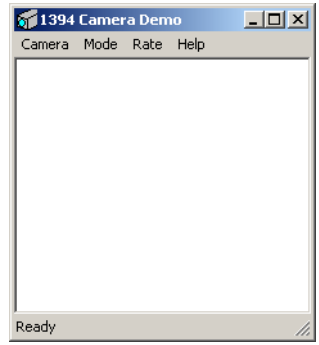
Running the CMU Camera Demo Application

The Carnegie Mellon University (CMU) DCAM driver distribution includes a camera demo application, named `1394CameraDemo.exe`. The demo application is among the files you extracted from the CMU DCAM archive file — see "Installing and Configuring the CMU DCAM Driver" on page 9-8.

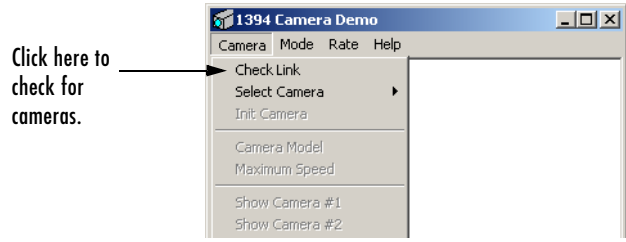
You can use this demo application to verify whether your camera is using the CMU DCAM driver. The following describes the step-by-step procedure you must perform to access a camera through this demo application.

1. Go to the directory into which you extracted the files from the CMU zip file. See "Installing and Configuring the CMU DCAM Driver" on page 9-8 for more information.

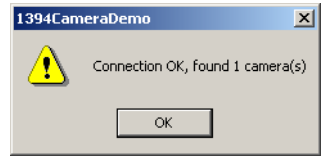
2. Double-click the `1394CameraDemo.exe` file. The application opens a window on your display, shown in the following figure.



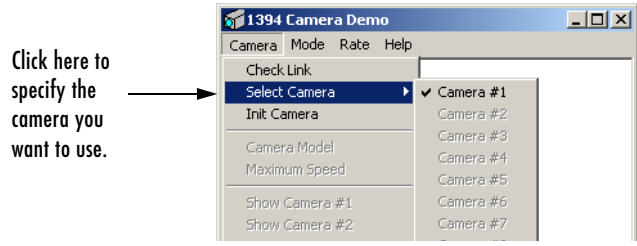
3. From the **Camera** menu, choose the **Check Link** option. This option causes the demo application to look for DCAM-compatible cameras that are available through the IEEE 1394 (FireWire) connection.



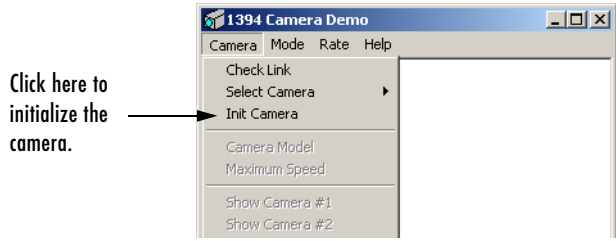
The demo application displays the results of this search in a pop-up message box. In the following example, the demo application found a camera. Click **OK** to continue.



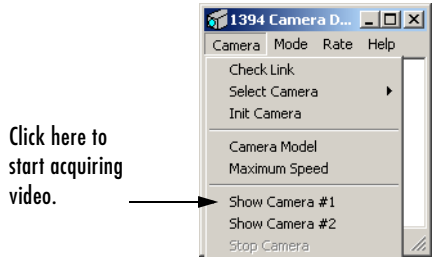
4 From the **Camera** menu, choose the **Select Camera** option and select the camera you want to use. The **Select Camera** option is not enabled until after the **Check Link** option has successfully found cameras.



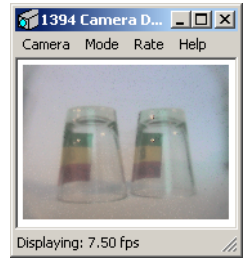
5 From the **Camera** menu, choose the **Init Camera** option. In this step, the demo application checks the values of various camera properties. The demo application might resize itself to fit the video format of the specified camera.



6 From the **Camera** menu, select the **Show Camera #1** option to start acquiring video.



The demo application starts displaying live video in the window.



7 To exit, select **Stop Camera** from the Camera menu and then click **Exit**.

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 the driver version you are using on your system. To learn how to get this information, see "Determining the Driver Version for Matrox Devices" on page 9-14.
   - Verify that the version is compatible with the Image Acquisition Toolbox. For the latest driver information, check the list of supported drivers on the Image Acquisition Toolbox product page at The MathWorks Web site (`www.mathworks.com/products/imaq`).

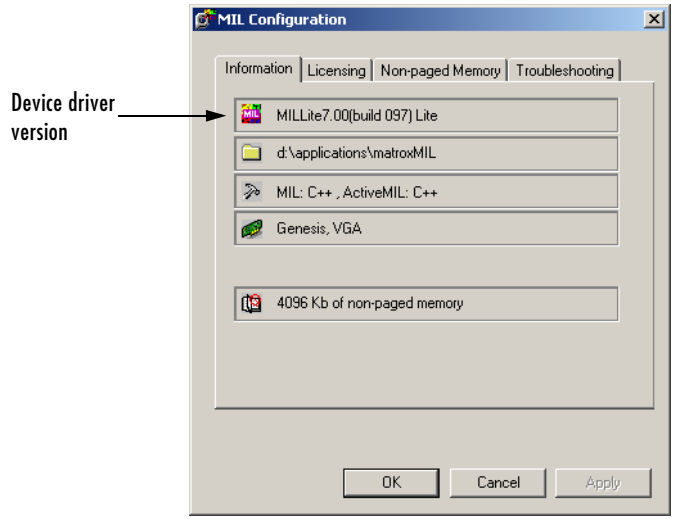
   If you discover that you are using an unsupported driver, visit the Matrox Web site (`www.matrox.com`) to download the latest drivers.

Determining the Driver Version for Matrox Devices

To determine the 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.



Matrox MIL Configuration Utility

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 `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.

   - Find out the driver version you are using on your system. 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.
   - Verify that the version is compatible with the Image Acquisition Toolbox. For the latest driver information, check 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.

3 Make sure you have the correct version of Microsoft DirectX installed on your computer. 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. To learn how to get this information, see "Determining the Microsoft DirectX Version".
- Verify that the version is compatible with the Image Acquisition Toolbox. For the latest version information, check the Image Acquisition Toolbox product page at The MathWorks Web site (`www.mathworks.com/products/imaq`).

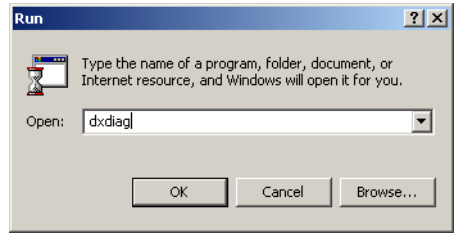
If you discover that you are using an unsupported version, visit the Microsoft DirectX Web site (`www.microsoft.com/directx/`) for the latest version of DirectX.

Determining the Microsoft DirectX Version

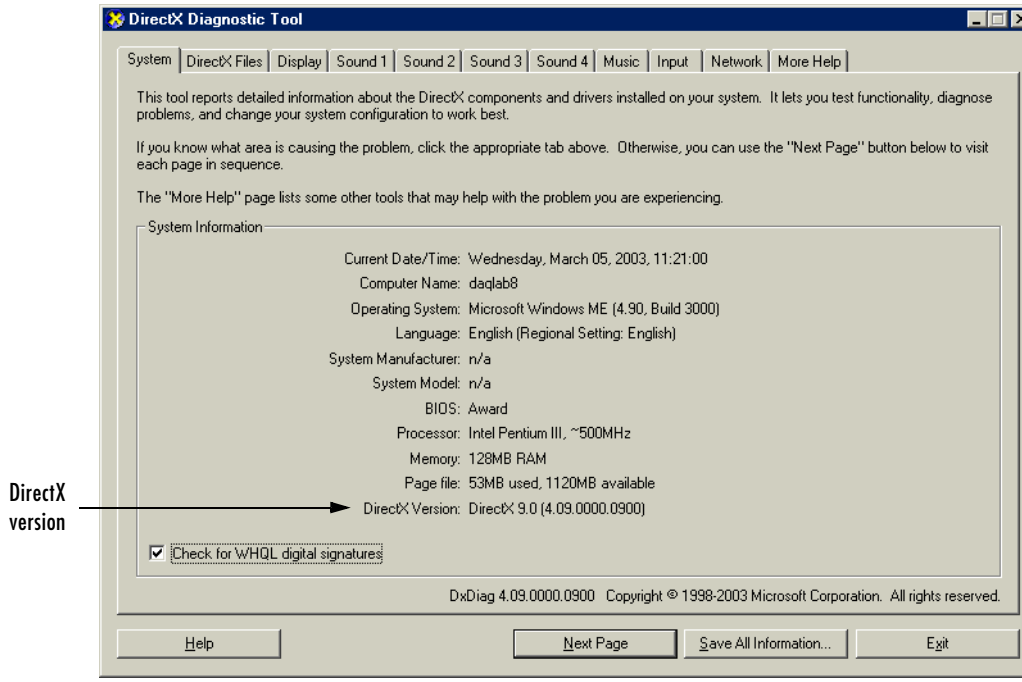
To determine the version of Microsoft DirectX you are using, run the DirectX Diagnostic Tool. You can access this software through the Windows **Start** button.

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 Diagnostic Tool

Troubleshooting a Video Preview Window

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

Problem	Possible Solutions
Video Preview window stops running.	• 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.
Video Preview window displays blank, gray window.	• 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.
Video Preview window displays dropped frames message.	• 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`.

10

Function Reference

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

Getting Command-Line Function Help (p. 10-2)	Describes how to get help for toolbox functions at the command line
Functions — Categorical List (p. 10-3)	Contains a series of tables that provide brief descriptions of Image Acquisition Toolbox functions, arranged by category
Functions — Alphabetical List (p. 10-6)	Contains individual reference pages for each function, arranged alphabetically

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 — Categorical List

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 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
`islogging`	Determine if video input object is currently logging data
`isrunning`	Determine if video input object is currently running
`isvalid`	Determine if image acquisition object is a valid object
`load`	Load image acquisition objects into the workspace
`obj2mfile`	Convert video input object into an M-file that can be used to reconstruct the object.

`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 or logging

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 preview window
`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
`stoppreview`	Stop previewing video data

Functions — Alphabetical List

This section provides detailed information about all the functions in the Image Acquisition Toolbox. The function descriptions are arranged alphabetically by function name.

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`

closepreview

Purpose

Close Video Preview window

Syntax

```
closepreview(obj)
closepreview
```

Description

`closepreview(obj)` stops the image acquisition object `obj` from previewing and, if the default Video Preview window was used, closes the window.

`closepreview` stops all image acquisition objects from previewing and, for all image acquisition objects that used the default Video Preview window, closes the windows.

Note If the preview window was created with a user specified image object handle as the target, `closepreview` does not close the figure window..

See Also

`preview`, `stoppreview`

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`

disp

Purpose

Display method for image acquisition objects

Syntax

```
obj
disp(obj)
```

Description

`obj` displays summary information for image acquisition object `obj`.

`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')
```

```
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 ST
                           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`

flushdata

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 data from the memory buffer that was acquired during the oldest trigger executed. `TriggerRepeat` must be greater than 0 and `FramesPerTrigger` must not be set to `inf`.

See Also

`getdata`, `imaqhelp`, `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 of the property specified by *`PropertyName`* for image acquisition object `obj`. Use the `get(obj)` syntax to view a list of all the properties supported by a particular image acquisition object.

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`

getdata

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 color 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`* is one of the text strings in the following table that specify the data type used to store the returned data.

Type String	Data Type
`'uint8'`	Unsigned 8-bit integer
`'uint16'`	Unsigned 16-bit integer

Type String	Data Type
'uint32'	Unsigned 32-bit integer
'single'	Single precision
'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* is one of the text strings in the following table that specify the MATLAB format of data.

Format String	Description
'numeric'	Returns data as an H-by-W-by-B-by-F array. This is the default format if none is specified.
'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 *n*th frame since the `start` command was issued
`'RelativeFrame'`	Number identifying the *n*th 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`

getselectedsource

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`

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`

Purpose

Find image acquisition objects

Syntax

```
imaqfind
out = imaqfind
out = imaqfind(PropertyName, Value, PropertyName2, Value2,...)
out = imaqfind(S)
out = imaqfind(obj, PropertyName, Value, PropertyName2, Value2,...)
```

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(PropertyName, Value, PropertyName2, Value2,...)` returns a cell array, `out`, of image acquisition objects whose property names and property values match those passed as arguments. You can specify the property name/property value pairs in a cell array. You can use a mixture of strings, structures, and cell arrays. Use the `get` function to determine the list of properties supported by an image acquisition object.

`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, PropertyName, Value, PropertyName2, Value2,...)` 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 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'`. 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'`. Use the `get` function to determine the exact spelling of a property value.

Example

These examples illustrate a variety of `imaqfind` syntaxes.

```
obj1 = videoinput('matrox', 1, 'M_RS170','Tag', 'FrameGrabber');

obj2 = videoinput('winvideo', 1, 'RGB24_320x240', 'Tag',
'Webcam');

out1 = imaqfind('Type', 'videoinput')

out2 = imaqfind('Tag', 'FrameGrabber')

out3 = imaqfind({'Type', 'Tag'}, {'videoinput', 'Webcam'})
```

See Also

`get`, `videoinput`

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 specified by the text string *`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 the function or property specified by the text string *`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');

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`

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 `out`, a structure 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 `out`, a structure that contains information about the adaptor specified by the text string *`adaptorname`*. 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 field specified by the text string *`field`* for the adaptor specified by the text string *`adaptorname`*. The argument 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 `out`, a structure containing information about the device specified by the numeric 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 `out`, a structure that contains information about the specified image acquisition object `obj`. The information returned includes the adaptor name, device name, video resolution, native data type, and device driver name and version. 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 information in the field specified by `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: '7.0.4 (R14SP2)'
          ToolboxName: 'Image Acquisition Toolbox'
       ToolboxVersion: '1.8 (R14SP2)
```

This example returns information about all the devices accessible through a particular adaptor.

```
info = imaqhwinfo('winvideo')
info =

       AdaptorDllName: [1x73 char]
    AdaptorDllVersion: '1.8 (R14SP2)'
          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
              NativeDataType: 'uint8'
                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`

imaqmem

Purpose

Limit memory or display memory usage for the Image Acquisition Toolbox

Syntax

```
mem = imaqmem
imaqmem(field)
imaqmem(limit)
```

Description

`mem = imaqmem` returns a structure 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 addressed 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 the toolbox is first used or queried.
`FrameMemoryUsed`	Number of bytes currently allocated by the Image Acquisition Toolbox

`imaqmem(`*`field`*`)` returns information for the field specified by the text string *`field`*.

`imaqmem(limit)` configures the frame memory limit, in bytes, for the Image Acquisition Toolbox. `limit` is used to determine the maximum amount of memory the toolbox can use for logging image frames.

Note Configuring the frame memory 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.

Example

Use `imaqmem` to get information about system memory.

```
imaqmem

ans =

          MemoryLoad: 85
           TotalPhys: 263766016
           AvailPhys: 37306368
       TotalPageFile: 643878912
       AvailPageFile: 391446528
        TotalVirtual: 2.1474e+009
        AvailVirtual: 1.6307e+009
    FrameMemoryLimit: 38313984
     FrameMemoryUsed: 0
```

Retrieve information about a specific field returned by `imaqmem`.

```
memlimit = imaqmem('FrameMemoryLimit')

memlimit =

    38313984
```

Specify the amount of memory available for the toolbox to log image frames (`FrameMemoryLimit`).

```
imaqmem(30000000)

ans =

          MemoryLoad: 85
           TotalPhys: 263766016
           AvailPhys: 37634048
       TotalPageFile: 643878912
       AvailPageFile: 391479296
        TotalVirtual: 2.1474e+009
        AvailVirtual: 1.6307e+009
    FrameMemoryLimit: 30000000
     FrameMemoryUsed: 0
```

See Also

`flushdata`, `getdata`, `videoinput`

Purpose

Display a sequence of image frames as a montage

Syntax

```
imaqmontage(frames)
imaqmontage(obj)
imaqmontage(...,CLIM)
h = imaqmontage(...)
```

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.

`h = imaqmontage(...)` returns a handle to an image object.

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.

You can use `imaqreset` 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 a video input object is logging

Syntax

```
bool = islogging(obj)
```

Description

`bool = islogging(obj)` returns `true` if the video input object `obj` is logging data, otherwise `false`. A video input object is logging if the value of its `Logging` property is set to `'on'`.

If `obj` is an array of video input objects, `bool` is a logical array where each element in `bool` represents the corresponding element in `obj`. If an object in `obj` is logging data, `islogging` sets the corresponding element in `bool` to `true`, otherwise `false`. If any of the video input objects in `obj` is invalid, `islogging` returns an error.

Example

Create a video input object.

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

To put the video input object in a logging state, start acquiring data. The example acquires 50 frames to increase the amount of time that the object remains in logging state.

```
set(vid,'FramesPerTrigger',50)
start(vid)
```

When the call to the start function returns, and the object is still acquiring data, use `islogging` to check the state of the object.

```
bool = islogging(vid)
bool =

   1
```

Create a second video input object.

```
vid2 = videoinput('winvideo');
```

Start one of the video input objects again, such as `vid`, and use `islogging` to determine which of the two objects is logging.

```
start(vid)
bool = islogging([vid vid2])

bool =

     1     0
```

See Also

`isrunning`, `isvalid`, `videoinput`

Properties

`Logging`, `LoggingMode`

Purpose

Determine if a video input object is running

Syntax

```
bool = isrunning(obj)
```

Description

`bool = isrunning(obj)` returns `true` if the video input object `obj` is running, otherwise `false`. A video input object is running if the value of its `Running` property is set to `'on'`.

If `obj` is an array of video input objects, `bool` is a logical array where each element in `bool` represents the corresponding element in `obj`. If an object in `obj` is running, the `isrunning` function sets the corresponding element in `bool` to `true`, otherwise `false`. If any of the video input objects in `obj` is invalid, `isrunning` returns an error.

Example

Create a video input object, configure a manual trigger, and then start the object. This puts the object in running state.

```
vid = videoinput('winvideo');
triggerconfig(vid,'manual')
start(vid)
```

Use `isrunning` to check the state of the object.

```
bool = isrunning(vid)
bool =
   1
```

Create a second video input object.

```
vid2 = videoinput('winvideo');
```

Use `isrunning` to determine which of the two objects is running.

```
bool = isrunning([vid vid2])
bool =
     1     0
```

See Also

`islogging`, `isvalid`, `start`, `stop`, `videoinput`

Properties

`Running`

isvalid

Purpose

Determine if an image acquisition object is associated with an image acquisition device

Syntax

```
bool = isvalid(obj)
```

Description

`bool = isvalid(obj)` returns `true` if the video input object `obj` is valid, otherwise `false`. An object is an invalid image acquisition object if it is no longer associated with any hardware; that is, the object was deleted using the `delete` function. If this is the case, `obj` should be cleared from the workspace.

If `obj` is an array of video input objects, `bool` is a logical array where each element in `bool` represents the corresponding element in `obj`. If an object in `obj` is valid, the `isvalid` function sets the corresponding element in `bool` to `true`, otherwise `false`.

See Also

`delete`, `imaqfind`, `videoinput`

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 specified by `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`. 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 when loaded. 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`

obj2mfile

Purpose

Convert video input objects to MATLAB code

Syntax

```
obj2mfile(obj,filename)
obj2mfile(obj,filename,syntax)
obj2mfile(obj,filename,syntax,mode)
obj2mfile(obj,filename,syntax,mode,reuse)
```

Description

`obj2mfile(obj,filename)` converts the video input object `obj` into an M-file with the name specified by `filename`. The M-file contains the MATLAB code required to create the object and set its properties. `obj` can be a single video input object or an array of objects.

The `obj2mfile` function simplifies the process of restoring an object with specific property settings and can be used to create video input objects. `obj2mfile` also creates and configures the video source object associated with the video input object.

If `filename` does not specify an extension or if it has an extension other than the MATLAB M-file extension (`.m`), `obj2mfile` appends `.m` to the end of `filename`. To recreate `obj`, execute the M-file by calling `filename`.

If the `UserData` property of the object is set, or if any of the callback properties is set to a cell array or to a function handle, `obj2mfile` writes the data stored in those properties to a MAT-file. `obj2mfile` gives the MAT-file the same name as the M-file, but uses the `.mat` filename extension. `obj2mfile` creates the MAT-file in the same directory as the M-file.

Note `obj2mfile` does not restore the values of read-only properties. For example, if an object is saved with a `Logging` property set to `'on'`, the object is recreated with a `Logging` property set to `'off'` (the default value). Use the `propinfo` function to determine if a property is read only.

`obj2mfile(obj,filename,`*`syntax`*`)` converts `obj` to the equivalent MATLAB code where *`syntax`* specifies how `obj2mfile` assigns values to properties of the object. `syntax` can be either of the following text strings. The default value is enclosed in braces ({}).

String	Description
`{'set'}`	`obj2mfile` uses the `set` function when specifying property values.
`'dot'`	`obj2mfile` uses subscripted assignment (dot notation) when specifying property values.

`obj2mfile(obj,filename,`*`syntax`*`,`*`mode`*`)` converts `obj` to the equivalent MATLAB code where `mode` specifies which properties are configured. `mode` can be either of the following strings. The default value is enclosed in braces ({}).

String	Description
`{'modified'}`	Configure writable properties that are not set to their default values.
`'all'`	Configure all writable properties. `obj2mfile` does not restore the values of read-only properties.

Note that `obj2mfile(obj,filename,`*`mode`*`)` is a valid syntax. If the *`syntax`* argument is not specified, `obj2mfile` uses the default value.

`obj2mfile(obj,filename,`*`syntax`*`,`*`mode`*`,`*`reuse`*`)` converts `obj` to the equivalent MATLAB code where *`reuse`* specifies whether `obj2mfile` searches for a reusable video input object or creates a new one. *`reuse`* can be either of the following strings. The default value is enclosed in braces ({}).

String	Description
`{'reuse'}`	Find and modify an existing object, if the existing object is associated with the same adaptor and the values of the `DeviceID`, `VideoFormat`, and `Tag` properties match the object being created. If no matching object can be found, `obj2mfile` creates a new object.
`'create'`	Create a new object regardless of whether there are reusable objects.

Note that `obj2mfile(obj,filename,`*`reuse`*`)` is a valid syntax. If the *`syntax`* and *`mode`* arguments are not specified, `obj2mfile` uses their default values.

Example

Create a video input object.

```
vidobj = videoinput('winvideo', 1, 'RGB24_640x480');
```

Configure several properties of the video input object.

```
set(vidobj, 'FramesPerTrigger', 100);
set(vidobj, 'FrameGrabInterval', 2);
set(vidobj, 'Tag', 'CAM1');
```

Retrieve the selected video source object associated with the video input object.

```
src = getselectedsource(vidobj);
```

Configure the properties of the video source object.

```
set(src, 'Contrast', 85);
set(src, 'Saturation', 125);
```

Save the video input object.

```
obj2mfile(vidobj, 'myvidobj.m', 'set', 'modified');
```

Delete the object and clear it from the workspace.

```
delete(vidobj);
clear vidobj;
```

Execute the M-file to recreate the object. Note that `obj2mfile` creates and configures the associated video source object as well.

```
vidObj = myvidobj;
```

See Also

`getselectedsource`, `imaqhelp`, `propinfo`, `set`, `videoinput`

peekdata

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.

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`

preview

Purpose

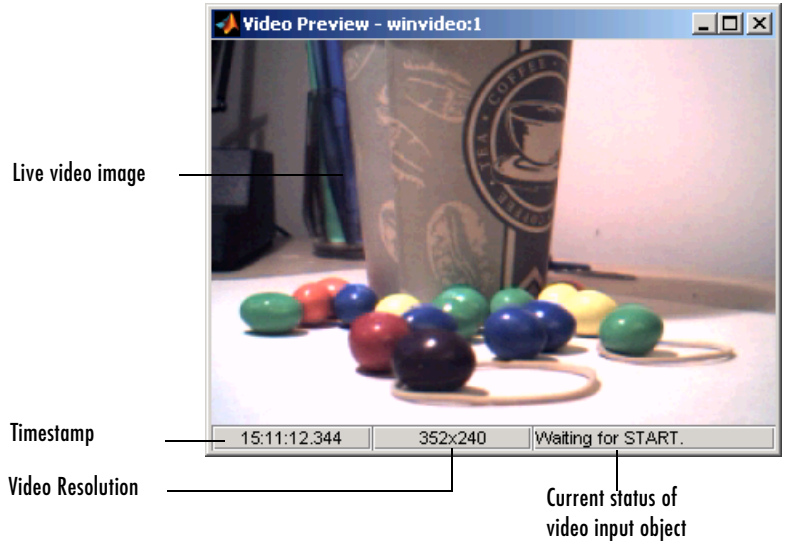
Display preview of live video data

Syntax

```
preview(obj)
preview(obj,himage)
himage = preview(...)
```

Description

`preview(obj)` creates a Video Preview window that displays live video data for video input object `obj`. The window also displays the timestamp and video resolution of each frame, and the current status of `obj`. The Video Preview window displays the video data at 100% magnification (one screen pixel represents one image pixel). The size of the preview image is determined by the value of the video input object `ROIPosition` property.



Components of a Video Preview Window

The Video Preview window displays image data in its native color space, indicated by the default value of the video input object `ReturnedColorSpace` property. To determine this value, use `propinfo`.

The Video Preview window remains active until it is either stopped using `stoppreview` or closed using `closepreview`. If you delete the object, by calling

`delete(obj)`, the Video Preview window stops previewing and closes automatically.

`preview(obj,himage)` displays live video data for video input object `obj` in the image object specified by the handle `himage`. `preview` scales the image data to fill the entire area of the image object but does not modify the values of any image object properties. Use this syntax to preview video data in a custom GUI of your own design (see Examples).

`himage = preview(...)` returns `himage`, a handle to the image object containing the previewed data. To obtain a handle to the figure window containing the image object, use the `ancestor` function. For more information about using image objects, see `image`. See the Custom Update Function section for more information about the image object returned.

Image data is previewed in its native color space, indicated by the default value of the `ReturnedColorSpace` property of the video input object. To determine the default value for the `ReturnedColorSpace` property, use `propinfo`.

Notes

The behavior of the Video Preview window depends on the video input object's current state and trigger configuration.

Object State	Preview Window Behavior
`Running=off`	Displays 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.)
`Running=on`	If `TriggerType` is set to `immediate` or `manual`, the Video Preview window continues to update the image displayed. If `TriggerType` is set to `hardware`, the Video Preview window stops updating the image displayed until a trigger occurs.
`Logging=on`	Video Preview window might drop some data frames, but this will not affect the frames logged to memory or disk.

Custom Update Function

`preview` creates application-defined data for the image object, `himage`, assigning it the name `'UpdatePreviewWindowFcn'` and setting its value to an empty array (`[]`). You can configure the value of the `'UpdatePreviewWindowFcn'` application data and retrieve its value using the MATLAB `setappdata` and `getappdata` functions, respectively.

If you set the value of `'UpdatePreviewWindowFcn'` to a function handle, `preview` invokes the function every time a new image frame is available. You can use this function to define custom processing of the previewed image data. When `preview` invokes the function handle you specify, it passes three arguments to your function:

- `obj` — The video input object being previewed
- `event` — An event structure containing image frame information. For more information, see below.
- `himage` — A handle to the image object that is being updated

The event structure contains the following fields:

Field	Description
`Data`	Current image frame specified as an H-by-W-by-B matrix where H and W are the image height and width, respectively, as specified in the `ROIPosition` property, and B is the number of color bands, as specified in the `NumberOfBands` property.
`Resolution`	String specifying current image width and height, as defined by the `ROIPosition` property.
`Status`	String describing the current acquisition status of the video input object.
`Timestamp`	String specifying the timestamp associated with the current image frame.

See Also

`ancestor`, `closepreview`, `image`, `imaqhelp`, `stoppreview`

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: • `'always'`— Property cannot be configured. • `'whileRunning'`— Property cannot be configured while `Running` is set to on. • `'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`

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 text 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`

set

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. Use the `set(obj)` syntax to get a list of all the properties for a particular image acquisition object that can be set.

`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 specified by the text string *`PropertyName`* to the value specified by `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`

start

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`

stop

Purpose

Stop video input object

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`

Purpose

Stop previewing video data

Syntax

```
stoppreview(obj)
```

Description

`stoppreview(obj)` stops the previewing of video data from image acquisition object `obj`.

To restart previewing, call `preview` again.

Example

Create a video input object and open a Video Preview window.

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

Stop previewing video data.

```
stoppreview(vid);
```

Restart previewing.

```
preview(vid)
```

See Also

`closepreview`, `preview`

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)` configures the value of the `TriggerType` property of the video input object `obj` to the value specified by the text string *`type`*. For a list of valid `TriggerType` values, use `triggerinfo(obj)`. *`type`* must specify a unique trigger configuration.

`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.

`triggerconfig(obj,type,condition)` configures the values of the `TriggerType` and `TriggerCondition` properties of the video input object `obj` to the values specified by the text strings *`type`* and *`condition`*. For a list of valid `TriggerType` and `TriggerCondition` values, use `triggerinfo(obj)`. *`type`* and *`condition`* must specify a unique trigger configuration.

`triggerconfig(obj,type,condition,source)` configures the values of the `TriggerType`, `TriggerCondition`, and `TriggerSource` properties of the video input object `obj` to the values specified by the text strings *`type`*, *`condition`*, and *`source`*, respectively. For a list of valid `TriggerType`, `TriggerCondition`, and `TriggerSource` values, use `triggerinfo(obj)`.

`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);
```

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 about available trigger configurations

Syntax

```
triggerinfo(obj)
triggerinfo(obj,type)
config = triggerinfo(...)
```

Description

`triggerinfo(obj)` displays all available trigger configurations for the 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 the video input object `obj`. To get a list of valid *`type`* values for a particular image acquisition object, use `triggerinfo(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.

1 Create a video input object.

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

2 Get information about the available trigger configurations for this object.

```
config = triggerinfo(vid)

config =

1x2 struct array with fields:
    TriggerType
    TriggerCondition
    TriggerSource
```

3 View one of the trigger configurations returned by `triggerinfo`.

```
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, format,P1,V1,P2,V2,...)
```

Description

`obj = videoinput(adaptorname)` constructs the video input object `obj`. A video input object represents the connection between MATLAB and particular image acquisition device. *`adaptorname`* is a text string that specifies the name of the adaptor used to communicate with the device. Use the `imaqhwinfo` function to determine the adaptors available on your system.

`obj = videoinput(adaptorname,deviceID)` constructs a video input object `obj`, where `deviceID` is a numeric scalar value that identifies a particular device available through the specified adaptor, *`adaptorname`*. Use the `imaqhwinfo(adaptorname)` syntax to determine the devices available through the specified adaptor. If `deviceID` is not specified, the first available device ID is used. 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,format)` constructs a video input object, where *`format`* is a text string that specifies a particular video format supported by the device or the full path of a device configuration file (also known as a camera file).

To get a list of the formats supported by a particular device, view the `DeviceInfo` structure for the device that is returned by the `imaqhwinfo` function. Each `DeviceInfo` structure contains a `SupportedFormats` field. If `format` is not specified, the device's default format is used.

When the video input object is created, its `VideoFormat` field contains the format name or device configuration file that you specify.

`obj = videoinput(adaptorname,deviceID,format,P1,V1,...)` 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 name and 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
```

Remarks

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.

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`

Purpose

Wait for the image acquisition object to stop running or logging

Syntax

```
wait(obj)
wait(obj,waittime)
wait(obj,waittime,state)
```

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 `obj` is not running or is an invalid object, `wait` returns immediately. The `wait` function can be useful when you want to guarantee that data is acquired before another task is performed.

`wait(obj,waittime)` blocks the MATLAB command line until the video input object or array of objects `obj` stops running or until `waittime` seconds have expired, whichever comes first. By default, `waittime` is set to the value of the object's `Timeout` property.

`wait(obj,waittime,state)` blocks the MATLAB command line until the video input object or array of objects `obj` stops running or logging, or until `waittime` seconds have expired, whichever comes first. `state` can be either of the following text strings. The default value is enclosed in braces ({}).

State	Description
`{'running'}`	Blocks until the value of the object's `Running` property is `'off'`.
`'logging'`	Blocks until the value of the object's `Logging` property is `'off'`.

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 or logging 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`

11

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 – Categorical List

This chapter contains brief descriptions of all the properties of the video input object 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.

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

`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
`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
`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.

`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

Properties – Alphabetical List

This section provides detailed information about all the properties of the video input object and the video source object. The property descriptions are arranged alphabetically by property name.

DeviceID

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: '7.0 (R14)'
          ToolboxName: 'Image Acquisition Toolbox'
       ToolboxVersion: '1.5 (R14)'
```

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: '1.5 (R14)'
          AdaptorName: 'winvideo'
            DeviceIDs: {[1]}
           DeviceInfo: [1x1 struct]
```

See Also

Functions

`imaqhwinfo`, `videoinput`

Properties

`Name`

DiskLogger

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 `[]`.

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`

DiskLoggerFrameCount

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`

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`

EventLog

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 6-7.

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`

FrameGrabInterval

Purpose

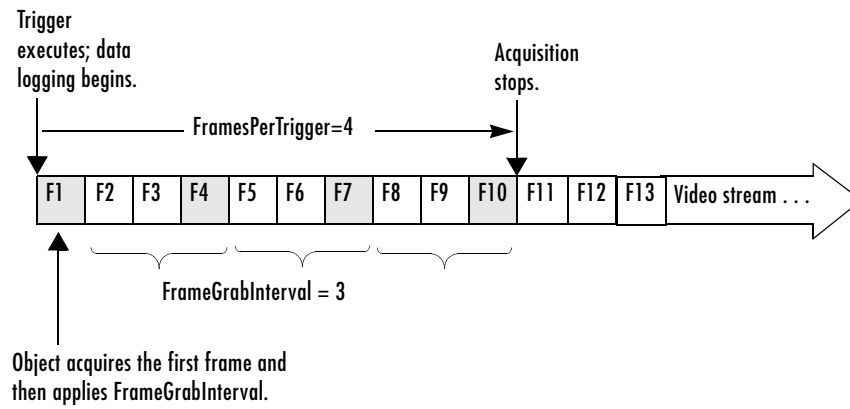
Specify how often to acquire a frame from the video stream

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.

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`

FramesAcquired

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`

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`

FramesAcquiredFcnCount

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`

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`

FramesPerTrigger

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`

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`

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`, `islogging`, `peekdata`, `stop`, `trigger`

Properties

`LoggingMode`, `Running`

LoggingMode

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 specify the amount of memory available for image frames.

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`

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`

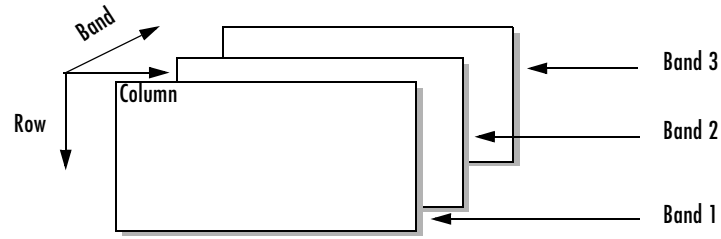
NumberOfBands

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
```

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`

Parent

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`

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`

ReturnedColorSpace

Purpose

Specify the color space used in MATLAB

Description

The `ReturnedColorSpace` property specifies the color space you want the toolbox to use when it returns image data 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'`	MATLAB grayscale color space
`'rgb'`	MATLAB RGB color space
`'YCbCr'`	MATLAB YCbCr color space. This is often referred to as YUV.

Characteristics

Access	Read/write
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`

Purpose

Specify the region-of-interest (ROI) window

Description

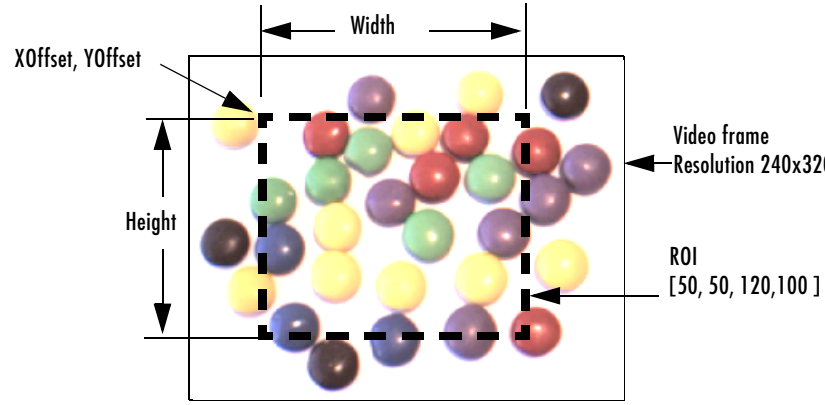
The `ROIPosition` property specifies the region-of-interest acquisition window. The ROI window defines the actual size of the frame logged by the toolbox, measured with respect to the top left corner of an image frame.

`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`.



ROIPosition

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`

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 is 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`

Selected

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);
```

View its `Name` and `Selected` properties.

```
src.SourceName

ans =

input1


src.Selected

ans =

on
```

See Also

Functions

`getselectedsource`

Properties

`SelectedSourceName`

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(vid,'SelectedSourceName');
```

See Also

Functions

`set`

Properties

`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`

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`

StartFcn

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.

The `StartFcn` callback executes synchronously. The toolbox does not set the object's `Running` property to `'on'` until the callback function finishes executing. If the callback function encounters an error, the object never starts running.

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`

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.

The `StopFcn` callback executes synchronously. Under most circumstances, the image acquisition object will be stopped and the `Running` property will be set to `'off'` by the time the M-file completes execution.

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`

Tag

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`

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`

TimerFcn

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`

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`

TriggerCondition

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`

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`

TriggerFrameDelay

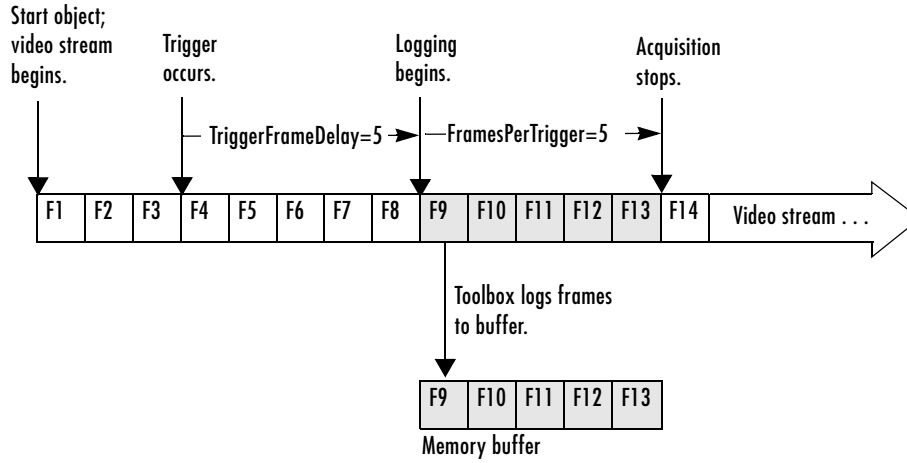
Purpose

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`

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`

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`

TriggerSource

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`

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`

Type

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`

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`

VideoFormat

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`.

```
info = imaqhwinfo('winvideo')

info =

       AdaptorDllName: [1x73 char]
    AdaptorDllVersion: '1.5 (R14)'
          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`

VideoResolution

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`

12

Block Reference

Blocks — Categorical List

The Image Acquisition Toolbox contains one block for use with Simulink: the Video Input block.

Blocks — Alphabetical List

This section provides detailed information about all the block in the Image Acquisition Blockset.

Video Input

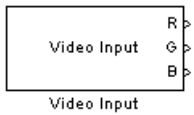
Purpose

Connect to image acquisition device

Library

Image Acquisition Toolbox

Description

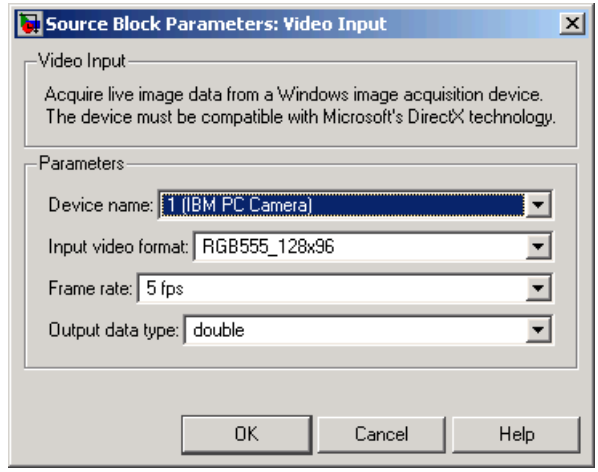


The Video Input block opens, initializes, configures, and controls an acquisition device. The opening, initializing, and configuration occur once, at the start of the model's execution. During the model's run-time, the block buffers image data, delivering one image frame for each simulation time step.

The block has no input ports. The block has three output ports, corresponding to the uncompressed red, green, and blue color bands.

Note The Video Input block supports only Windows video devices compatible with DirectX.

Dialog Box



Device Name

The image acquisition device to which you want to connect. The items on the list vary, depending on which devices you have connected to your system.

Input video format

The video formats supported by the device. This list varies with each device.

Frame rate

The speed at which frames are delivered to the block, expressed as frames-per-second (fps).

Output data type

The image data type used when the block outputs frames. This data type indicates how image frames are stored internally.

Index

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W