**Revision History**

<table>
<thead>
<tr>
<th>Date</th>
<th>Printing Type</th>
<th>Revision Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>May 1999</td>
<td>First printing</td>
<td>New for Version 1</td>
</tr>
<tr>
<td>November 2000</td>
<td>Second printing</td>
<td>Revised for Version 2 (Release 12)</td>
</tr>
<tr>
<td>June 2001</td>
<td>Third printing</td>
<td>Revised for Version 2.1 (Release 12.1)</td>
</tr>
<tr>
<td>July 2002</td>
<td>Online only</td>
<td>Revised for Version 2.2 (Release 13)</td>
</tr>
<tr>
<td>June 2004</td>
<td>Online only</td>
<td>Revised for Version 2.5 (Release 14)</td>
</tr>
<tr>
<td>October 2004</td>
<td>Online only</td>
<td>Revised for Version 2.5.1 (Release 14SP1)</td>
</tr>
<tr>
<td>March 2005</td>
<td>Online only</td>
<td>Revised for Version 2.6 (Release 14SP2)</td>
</tr>
<tr>
<td>September 2005</td>
<td>Online only</td>
<td>Revised for Version 2.7 (Release 14SP3)</td>
</tr>
<tr>
<td>October 2005</td>
<td>Reprint</td>
<td>Version 2.1 (Notice updated)</td>
</tr>
<tr>
<td>November 2005</td>
<td>Online only</td>
<td>Revised for Version 2.8 (Release 14SP3+)</td>
</tr>
<tr>
<td>March 2006</td>
<td>Fourth printing</td>
<td>Revised for Version 2.8.1 (Release 2006a)</td>
</tr>
<tr>
<td>September 2006</td>
<td>Online only</td>
<td>Revised for Version 2.9 (Release 2006b)</td>
</tr>
<tr>
<td>March 2007</td>
<td>Online only</td>
<td>Revised for Version 2.10 (Release 2007a)</td>
</tr>
<tr>
<td>May 2007</td>
<td>Fifth printing</td>
<td>Minor revision for Version 2.10</td>
</tr>
<tr>
<td>September 2007</td>
<td>Online only</td>
<td>Revised for Version 2.11 (Release 2007b)</td>
</tr>
</tbody>
</table>
Introduction to Data Acquisition

What Is Data Acquisition Toolbox? .......................... 1-3
Understanding The Data Acquisition Toolbox ............ 1-3
Exploring the Toolbox ..................................... 1-4
Supported Hardware ........................................... 1-4

Anatomy of a Data Acquisition Experiment ............. 1-5
System Setup .................................................. 1-5
Calibration ....................................................... 1-5
Trials ............................................................... 1-6

Data Acquisition System ..................................... 1-7
Overview ......................................................... 1-7
Data Acquisition Hardware ................................ 1-9
Sensors ............................................................. 1-11
Signal Conditioning .......................................... 1-15
The Computer .................................................. 1-17
Software .......................................................... 1-17

Analog Input Subsystem ................................. 1-20
Function of the Analog Input Subsystem ............... 1-20
Sampling .......................................................... 1-20
Quantization ...................................................... 1-24
Channel Configuration ...................................... 1-28
Transferring Data from Hardware to System Memory ... 1-31

Making Quality Measurements ......................... 1-34
What Do You Measure? ...................................... 1-34
Accuracy and Precision .................................... 1-34
Noise ............................................................... 1-38
Matching the Sensor Range and A/D Converter Range ... 1-40
How Fast Should a Signal Be Sampled? ................. 1-40

Getting Command-Line Function Help .................. 1-45
Getting Started Using Data Acquisition Toolbox

Installation Information ............................................. 2-2
   Before You Install .............................................. 2-2
   Toolbox Installation ............................................ 2-2
   Hardware and Driver Installation ............................... 2-3

Toolbox Components ................................................... 2-4
   Toolbox Components: Information and Interaction ........... 2-4
   M-File Functions .................................................. 2-6
   Data Acquisition Engine ......................................... 2-6
   Hardware Driver Adaptor ......................................... 2-9
   Supported Hardware .............................................. 2-9
   Unsupported Hardware ........................................... 2-11

Accessing Your Hardware .............................................. 2-12
   Connecting to Your Hardware ................................. 2-12
   Acquiring Data ................................................... 2-12
   Outputting Data ................................................... 2-13
   Reading and Writing Digital Values ............................ 2-14
   Acquiring Data in a Loop ......................................... 2-16

Understanding the Toolbox Capabilities ........................... 2-17
   Contents M-File .................................................... 2-17
   Documentation Examples ......................................... 2-17
   Quick Reference Guide ........................................... 2-18
   Demos ............................................................... 2-18

Examining Your Hardware Resources ................................ 2-19
   Using the daqhwinfo Function .................................. 2-19
   General Toolbox Information .................................... 2-19
   Adaptor-Specific Information .................................... 2-20
   Device Object Information ....................................... 2-21

Getting Help ............................................................ 2-23
# Data Acquisition Session

## Understanding the Data Acquisition Session
- Overview ........................................ 3-2
- Example: The Data Acquisition Session .............. 3-3

## Creating a Device Object
- Understanding Device Objects ........................ 3-5
- Creating an Array of Device Objects .................. 3-6
- Where Do Device Objects Exist? .................. 3-7

## Hardware Channels or Lines
- Adding Channels and Lines .......................... 3-9
- Mapping Hardware Channel IDs to MATLAB Indices ... 3-10

## Configuring and Returning Properties
- Overview ........................................ 3-13
- Property Types .................................... 3-13
- Returning Property Names and Property Values ........ 3-15
- Configuring Property Values .................. 3-19
- Specifying Property Names ........................ 3-21
- Default Property Values .......................... 3-21
- The Property Inspector ............................ 3-22

## Acquiring and Outputting Data
- Device Object States ............................ 3-23
- Starting the Device Object .................. 3-24
- Logging or Sending Data .......................... 3-24
- Stopping the Device Object .................. 3-25

## Cleaning Up ................................... 3-27
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampling Rate</td>
<td>5-5</td>
</tr>
<tr>
<td>Channel Skew</td>
<td>5-6</td>
</tr>
<tr>
<td>Managing Acquired Data</td>
<td>5-8</td>
</tr>
<tr>
<td>Analog Input Data Management Properties</td>
<td>5-8</td>
</tr>
<tr>
<td>Previewing Data</td>
<td>5-8</td>
</tr>
<tr>
<td>Rules for Using peekdata</td>
<td>5-9</td>
</tr>
<tr>
<td>Extracting Data from the Engine</td>
<td>5-11</td>
</tr>
<tr>
<td>Returning Time Information</td>
<td>5-16</td>
</tr>
<tr>
<td>Configuring Analog Input Triggers</td>
<td>5-19</td>
</tr>
<tr>
<td>Analog Input Trigger Properties</td>
<td>5-19</td>
</tr>
<tr>
<td>Defining a Trigger: Trigger Types and Conditions</td>
<td>5-20</td>
</tr>
<tr>
<td>Executing the Trigger</td>
<td>5-25</td>
</tr>
<tr>
<td>Trigger Delays</td>
<td>5-26</td>
</tr>
<tr>
<td>Repeating Triggers</td>
<td>5-30</td>
</tr>
<tr>
<td>How Many Triggers Occurred?</td>
<td>5-35</td>
</tr>
<tr>
<td>When Did the Trigger Occur?</td>
<td>5-36</td>
</tr>
<tr>
<td>Device-Specific Hardware Triggers</td>
<td>5-37</td>
</tr>
<tr>
<td>Events and Callbacks</td>
<td>5-45</td>
</tr>
<tr>
<td>Understanding Events and Callbacks</td>
<td>5-45</td>
</tr>
<tr>
<td>Event Types</td>
<td>5-45</td>
</tr>
<tr>
<td>Recording and Retrieving Event Information</td>
<td>5-48</td>
</tr>
<tr>
<td>Creating and Executing Callback Functions</td>
<td>5-52</td>
</tr>
<tr>
<td>Examples: Using Callback Properties and Functions</td>
<td>5-54</td>
</tr>
<tr>
<td>Linearly Scaling the Data: Engineering Units</td>
<td>5-57</td>
</tr>
<tr>
<td>Analog Input Engineering Units Properties</td>
<td>5-57</td>
</tr>
<tr>
<td>Example: Performing a Linear Conversion</td>
<td>5-59</td>
</tr>
<tr>
<td>Linear Conversion with Asymmetric Data</td>
<td>5-60</td>
</tr>
</tbody>
</table>

**Analog Output**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Getting Started with Analog Output</td>
<td>6-2</td>
</tr>
<tr>
<td>Creating an Analog Output Object</td>
<td>6-2</td>
</tr>
<tr>
<td>Adding Channels to an Analog Output Object</td>
<td>6-3</td>
</tr>
<tr>
<td>Configuring Analog Output Properties</td>
<td>6-5</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Outputting Data</td>
<td>6-7</td>
</tr>
<tr>
<td>Analog Output Examples</td>
<td>6-9</td>
</tr>
<tr>
<td>Evaluating the Analog Output Object Status</td>
<td>6-12</td>
</tr>
<tr>
<td><strong>Managing Output Data</strong></td>
<td>6-16</td>
</tr>
<tr>
<td>The Analog Output Subsystem</td>
<td>6-16</td>
</tr>
<tr>
<td>Queuing Data with putdata</td>
<td>6-16</td>
</tr>
<tr>
<td>Example: Queuing Data with putdata</td>
<td>6-18</td>
</tr>
<tr>
<td><strong>Configuring Analog Output Triggers</strong></td>
<td>6-20</td>
</tr>
<tr>
<td>Analog Output Trigger Properties</td>
<td>6-20</td>
</tr>
<tr>
<td>Defining a Trigger: Trigger Types</td>
<td>6-21</td>
</tr>
<tr>
<td>Executing the Trigger</td>
<td>6-22</td>
</tr>
<tr>
<td>How Many Triggers Occurred?</td>
<td>6-22</td>
</tr>
<tr>
<td>When Did the Trigger Occur?</td>
<td>6-23</td>
</tr>
<tr>
<td>Device-Specific Hardware Triggers</td>
<td>6-24</td>
</tr>
<tr>
<td><strong>Events and Callbacks</strong></td>
<td>6-26</td>
</tr>
<tr>
<td>Understanding Events and Callbacks</td>
<td>6-26</td>
</tr>
<tr>
<td>Event Types</td>
<td>6-26</td>
</tr>
<tr>
<td>Recording and Retrieving Event Information</td>
<td>6-29</td>
</tr>
<tr>
<td>Examples: Using Callback Properties and Callback Functions</td>
<td>6-32</td>
</tr>
<tr>
<td><strong>Linearly Scaling the Data</strong></td>
<td>6-35</td>
</tr>
<tr>
<td>Engineering Units</td>
<td>6-35</td>
</tr>
<tr>
<td>Example: Performing a Linear Conversion</td>
<td>6-36</td>
</tr>
<tr>
<td><strong>Starting Multiple Device Objects</strong></td>
<td>6-38</td>
</tr>
</tbody>
</table>

## Digital Input/Output

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital I/O Objects</td>
<td>7-3</td>
</tr>
<tr>
<td>Creating a Digital I/O Object</td>
<td>7-3</td>
</tr>
<tr>
<td>The Parallel Port</td>
<td>7-4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adding Lines to a Digital I/O Object</td>
<td>7-6</td>
</tr>
</tbody>
</table>
Using the Addline Function ........................................ 7-6
Line and Port Characteristics ..................................... 7-8
Referencing Individual Hardware Lines ....................... 7-12

Writing and Reading Digital I/O Line Values ............... 7-15
  Writing Digital Values ...................................... 7-15
  Reading Digital Values .................................... 7-17
  Example: Writing and Reading Digital Values ............. 7-18

Generating Timer Events ....................................... 7-20
  Overview .................................................. 7-20
  Timer Events ............................................ 7-20
  Starting and Stopping a Digital I/O Object ............... 7-21
  Example: Generating Timer Events ......................... 7-22

Evaluating the Digital I/O Object Status .................. 7-24
  Running Property ....................................... 7-24
  The Display Summary .................................. 7-24

Saving and Loading the Session

8

Saving and Loading Device Objects ......................... 8-2
  Saving Device Objects to an M-File ....................... 8-2
  Saving Device Objects to a MAT-File ..................... 8-4

Logging Information to Disk .................................. 8-5
  Analog Input Logging Properties ......................... 8-5
  Specifying a Filename ................................... 8-6
  Retrieving Logged Information ......................... 8-7
  Example: Logging and Retrieving Information .......... 8-9
## Contents

**Oscilloscope Overview** ........................................... 9-3
- Opening the Oscilloscope .................................. 9-3
- Hardware Configuration .................................... 9-4

**Displaying Channels** ........................................ 9-6
- Creating a Display ....................................... 9-6
- Creating Additional Displays ......................... 9-7
- Configuring Display Properties ...................... 9-9
- Math and Reference Channels ......................... 9-10
- Removing Channel Displays .......................... 9-13

**Channel Data and Properties** ................................ 9-15
- Scaling the Channel Data ............................... 9-15
- Configuring Channel Properties .................... 9-16

**Triggering the Oscilloscope** .................................. 9-19
- Acquisition Types ..................................... 9-19
- Trigger Types ......................................... 9-19
- Configuring Trigger Properties .................... 9-21

**Making Measurements** ....................................... 9-22
- Predefined Measurement ............................... 9-22
- Defining a Measurement ................................ 9-23
- Defining a New Measurement Type .................. 9-25
- Configuring Measurement Properties .............. 9-26

**Exporting Data** ............................................. 9-29
- Channels ............................................... 9-29
- Measurements ........................................... 9-30

**Saving and Loading the Oscilloscope Configuration** .......... 9-32
Using the Data Acquisition Blocks in Simulink

10

Overview .................................................. 10-2

Opening the Data Acquisition Block Library .......... 10-3
  Using the daqlib Command from MATLAB .......... 10-3
  Using the Simulink Library Browser ................. 10-3

Building Simulink Models to Acquire Data from a
  Device .................................................. 10-6
  Data Acquisition Toolbox Library .................... 10-6
  Example: Bringing Analog Data into a Model ........ 10-6

Functions — By Category

11

Creating Device Objects ............................... 11-2

Adding Channels and Lines ............................ 11-3

Getting and Setting Properties ....................... 11-4

Executing the Object .................................. 11-5

Working with Data ..................................... 11-6

Getting Information and Help ......................... 11-7

General Purpose ....................................... 11-8
National Instruments ........................................ 15-5

Parallel Port .................................................. 15-5

Sound Card .................................................... 15-6
Getting Command-Line Property Help ................. 15-6

Device-Specific Properties — Alphabetical List

16

Blocks — Alphabetical List

17

Troubleshooting Your Hardware

A

Advantech Hardware ..................................... A-3
What Driver Are You Using? .............................. A-3
Is Your Hardware Functioning Properly? .............. A-3

Agilent Technologies Hardware ....................... A-5
Is Your Hardware Functioning Properly? .............. A-5

Measurement Computing Hardware .................. A-7
What Driver Are You Using? .............................. A-7
Is Your Hardware Functioning Properly? .............. A-7

National Instruments Hardware ...................... A-9
NI-DAQmx Versus Traditional NI-DAQ Drivers .... A-9
What Driver Are You Using? .............................. A-10
Is Your Hardware Functioning Properly? .............. A-10
Introduction to Data Acquisition

Before you set up any data acquisition system, you should understand the physical quantities you want to measure, the characteristics of those physical quantities, the appropriate sensor to use, and the appropriate data acquisition hardware to use.

The purpose of this chapter is to provide you with some general guidelines about making measurements with a data acquisition system. The information provided should assist you in understanding the above considerations, and understanding the specification sheet associated with your hardware. The sections are as follows.

- **What Is Data Acquisition Toolbox? (p. 1-3)**
  - Description of the toolbox and the kinds of tasks it can perform

- **Anatomy of a Data Acquisition Experiment (p. 1-5)**
  - Tasks you perform for each new data acquisition experiment

- **Data Acquisition System (p. 1-7)**
  - Typical components that compose a data acquisition system

- **Analog Input Subsystem (p. 1-20)**
  - Hardware subsystem that converts (digitizes) real-world sensor signals into numbers your computer can read

- **Making Quality Measurements (p. 1-34)**
  - Maximizing precision and accuracy, minimizing noise, and matching the sensor range to the hardware range
| Getting Command-Line Function Help (p. 1-45) | Getting command-line function help using the daqhelp function |
| Selected Bibliography (p. 1-46)             | Resources for additional information |
What Is Data Acquisition Toolbox?

In this section...

“Understanding The Data Acquisition Toolbox” on page 1-3
“Exploring the Toolbox” on page 1-4
“Supported Hardware” on page 1-4

Understanding The Data Acquisition Toolbox

Data Acquisition Toolbox is a collection of M-file functions and a MEX-file (shared library) built on the MATLAB® technical computing environment. The toolbox also includes several dynamic link libraries (DLLs) called adaptors, which enable you to interface with specific hardware. The toolbox provides you with these main features:

• A framework for bringing live, measured data into MATLAB using PC-compatible, plug-in data acquisition hardware
• Support for analog input (AI), analog output (AO), and digital I/O (DIO) subsystems including simultaneous analog I/O conversions
• Support for these popular hardware vendors/devices:
  - Advantech boards that use the Advantech Device Manager
  - Agilent Technologies E1432A/33A/34A VXI modules
  - Keithley boards that use DriverLINX drivers
  - Measurement Computing Corporation (ComputerBoards) boards
  - National Instruments boards that use Traditional NI-DAQ or NI-DAQmx software (except SCXI)
  - Parallel ports LPT1-LPT3
  - Windows sound cards

  Additionally, you can use the Data Acquisition Toolbox Adaptor Kit to interface unsupported hardware devices to the toolbox.

• Event-driven acquisitions
Exploring the Toolbox

A list of the toolbox functions is available to you by typing

```matlab
daq
```

You can view the code for any function by typing

```matlab
type function_name
```

You can view the help for any function by typing

```matlab
daqhelp function_name
```

You can change the way any toolbox function works by copying and renaming the M-file, then modifying your copy. You can also extend the toolbox by adding your own M-files, or by using it in combination with other products such as Signal Processing Toolbox or Instrument Control Toolbox.

The MathWorks provides several related products that are especially relevant to the kinds of tasks you can perform with Data Acquisition Toolbox. For more information about any of these products, see [http://www.mathworks.com/products/daq/related.jsp](http://www.mathworks.com/products/daq/related.jsp).

Supported Hardware

The list of hardware that Data Acquisition Toolbox supports can change in each release, since hardware support is frequently added. The MathWorks Web site is the best place to check for the most up-to-date listing.

To see the full list of hardware that the toolbox supports, visit the Data Acquisition Toolbox product page at [www.mathworks.com/products/daq](http://www.mathworks.com/products/daq) and click the Supported Hardware link.
Anatomy of a Data Acquisition Experiment

In this section...

| “System Setup” on page 1-5 |
| “Calibration” on page 1-5 |
| “Trials” on page 1-6 |

System Setup

The first step in any data acquisition experiment is to install the hardware and software. Hardware installation consists of plugging a board into your computer or installing modules into an external chassis. Software installation consists of loading hardware drivers and application software onto your computer. After the hardware and software are installed, you can attach your sensors.

Calibration

After the hardware and software are installed and the sensors are connected, the data acquisition hardware should be calibrated. Calibration consists of providing a known input to the system and recording the output. For many data acquisition devices, calibration can be easily accomplished with software provided by the vendor.
Trials
After the hardware is set up and calibrated, you can begin to acquire data. You might think that if you completely understand the characteristics of the signal you are measuring, then you should be able to configure your data acquisition system and acquire the data.

In the real world however, your sensor might be picking up unacceptable noise levels and require shielding, or you might need to run the device at a higher rate, or perhaps you need to add an antialias filter to remove unwanted frequency components.

These real-world effects act as obstacles between you and a precise, accurate measurement. To overcome these obstacles, you need to experiment with different hardware and software configurations. In other words, you need to perform multiple data acquisition trials.
Data Acquisition System

In this section...

“Overview” on page 1-7
“Data Acquisition Hardware” on page 1-9
“Sensors” on page 1-11
“Signal Conditioning” on page 1-15
“The Computer” on page 1-17
“Software” on page 1-17

Overview

As a user of MATLAB and Data Acquisition Toolbox, you are interested in measuring and analyzing physical phenomena. The purpose of any data acquisition system is to provide you with the tools and resources necessary to do so.

You can think of a data acquisition system as a collection of software and hardware that connects you to the physical world. A typical data acquisition system consists of these components.

<table>
<thead>
<tr>
<th>Components</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data acquisition hardware</td>
<td>At the heart of any data acquisition system lies the data acquisition hardware. The main function of this hardware is to convert analog signals to digital signals, and to convert digital signals to analog signals.</td>
</tr>
<tr>
<td>Sensors and actuators (transducers)</td>
<td>Sensors and actuators can both be transducers. A transducer is a device that converts input energy of one form into output energy of another form. For example, a microphone is a sensor that converts sound energy (in the form of pressure) into electrical energy, while a loudspeaker is an actuator that converts electrical energy into sound energy.</td>
</tr>
</tbody>
</table>
Signal conditioning hardware

Sensor signals are often incompatible with data acquisition hardware. To overcome this incompatibility, the signal must be conditioned. For example, you might need to condition an input signal by amplifying it or by removing unwanted frequency components. Output signals might need conditioning as well. However, only input signal conditioning is discussed in this chapter.

Computer

The computer provides a processor, a system clock, a bus to transfer data, and memory and disk space to store data.

Software

Data acquisition software allows you to exchange information between the computer and the hardware. For example, typical software allows you to configure the sampling rate of your board, and acquire a predefined amount of data.

The data acquisition components, and their relationship to each other, are shown below.
The figure depicts the two important features of a data acquisition system:

- Signals are input to a sensor, conditioned, converted into bits that a computer can read, and analyzed to extract meaningful information.

  For example, sound level data is acquired from a microphone, amplified, digitized by a sound card, and stored in MATLAB for subsequent analysis of frequency content.

- Data from a computer is converted into an analog signal and output to an actuator.

  For example, a vector of data in MATLAB is converted to an analog signal by a sound card and output to a loudspeaker.

**Data Acquisition Hardware**

Data acquisition hardware is either internal and installed directly into an expansion slot inside your computer, or external and connected to your
computer through an external cable. For example, VXI modules are installed in an external VXI chassis, and data is transferred between MATLAB and the module using an external link such as FireWire (IEEE 1394).

At the simplest level, data acquisition hardware is characterized by the subsystems it possesses. A subsystem is a component of your data acquisition hardware that performs a specialized task. Common subsystems include

- Analog input
- Analog output
- Digital input/output
- Counter/timer

Hardware devices that consist of multiple subsystems, such as the one depicted below, are called multifunction boards.

**Analog Input Subsystems**

Analog input subsystems convert real-world analog input signals from a sensor into bits that can be read by your computer. Perhaps the most important of all the subsystems commonly available, they are typically multichannel devices offering 12 or 16 bits of resolution.

Analog input subsystems are also referred to as AI subsystems, A/D converters, or ADCs. Analog input subsystems are discussed in detail beginning in “Analog Input Subsystem” on page 1-20.
Analog Output Subsystems
Analog output subsystems convert digital data stored on your computer to a real-world analog signal. These subsystems perform the inverse conversion of analog input subsystems. Typical acquisition boards offer two output channels with 12 bits of resolution, with special hardware available to support multiple channel analog output operations.

Analog output subsystems are also referred to as AO subsystems, D/A converters, or DACs.

Digital Input/Output Subsystems
Digital input/output (DIO) subsystems are designed to input and output digital values (logic levels) to and from hardware. These values are typically handled either as single bits or lines, or as a port, which typically consists of eight lines.

While most popular data acquisition cards include some digital I/O capability, it is usually limited to simple operations, and special dedicated hardware is often necessary for performing advanced digital I/O operations.

Counter/Timer Subsystems
Counter/timer (C/T) subsystems are used for event counting, frequency and period measurement, and pulse train generation.

Sensors
A sensor converts the physical phenomena of interest into a signal that is input into your data acquisition hardware. There are two main types of sensors based on the output they produce: digital sensors and analog sensors.

Digital sensors produce an output signal that is a digital representation of the input signal, and has discrete values of magnitude measured at discrete times. A digital sensor must output logic levels that are compatible with the digital receiver. Some standard logic levels include transistor-transistor logic (TTL) and emitter-coupled logic (ECL). Examples of digital sensors include switches and position encoders.
Analog sensors produce an output signal that is directly proportional to the input signal, and is continuous in both magnitude and in time. Most physical variables such as temperature, pressure, and acceleration are continuous in nature and are readily measured with an analog sensor. For example, the temperature of an automobile cooling system and the acceleration produced by a child on a swing all vary continuously.

The sensor you use depends on the phenomena you are measuring. Some common analog sensors and the physical variables they measure are listed below.

**Table 1-1  Common Analog Sensors**

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Physical Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer</td>
<td>Acceleration</td>
</tr>
<tr>
<td>Microphone</td>
<td>Pressure</td>
</tr>
<tr>
<td>Pressure gauge</td>
<td>Pressure</td>
</tr>
<tr>
<td>Resistive temperature device (RTD)</td>
<td>Temperature</td>
</tr>
<tr>
<td>Strain gauge</td>
<td>Force</td>
</tr>
<tr>
<td>Thermocouple</td>
<td>Temperature</td>
</tr>
</tbody>
</table>

When choosing the best analog sensor to use, you must match the characteristics of the physical variable you are measuring with the characteristics of the sensor. The two most important sensor characteristics are

- The sensor output
- The sensor bandwidth

**Sensor Output**

The output from a sensor can be an analog signal or a digital signal, and the output variable is usually a voltage although some sensors output current.
**Current Signals.** Current is often used to transmit signals in noisy environments because it is much less affected by environmental noise. The full scale range of the current signal is often either 4-20 mA or 0-20 mA. A 4-20 mA signal has the advantage that even at minimum signal value, there should be a detectable current flowing. The absence of this indicates a wiring problem.

Before conversion by the analog input subsystem, the current signals are usually turned into voltage signals by a current-sensing resistor. The resistor should be of high precision, perhaps 0.03% or 0.01% depending on the resolution of your hardware. Additionally, the voltage signal should match the signal to an input range of the analog input hardware. For 4-20 mA signals, a 50 ohm resistor will give a voltage of 1 V for a 20 mA signal by Ohm’s law.

**Voltage Signals.** The most commonly interfaced signal is a voltage signal. For example, thermocouples, strain gauges, and accelerometers all produce voltage signals. There are three major aspects of a voltage signal that you need to consider:

- **Amplitude**
  
  If the signal is smaller than a few millivolts, you might need to amplify it. If it is larger than the maximum range of your analog input hardware (typically ±10 V), you will have to divide the signal down using a resistor network.

  The amplitude is related to the sensitivity (resolution) of your hardware. Refer to “Accuracy and Precision” on page 1-34 for more information about hardware sensitivity.

- **Frequency**
  
  Whenever you acquire data, you should decide the highest frequency you want to measure.

  The highest frequency component of the signal determines how often you should sample the input. If you have more than one input, but only one analog input subsystem, then the overall sampling rate goes up in proportion to the number of inputs. Higher frequencies might be present as noise, which you can remove by filtering the signal before it is digitized.

  If you sample the input signal at least twice as fast as the highest frequency component, then that signal will be uniquely characterized. However, this
rate might not mimic the waveform very closely. For a rapidly varying signal, you might need a sampling rate of roughly 10 to 20 times the highest frequency to get an accurate picture of the waveform. For slowly varying signals, you need only consider the minimum time for a significant change in the signal.

The frequency is related to the bandwidth of your measurement. Bandwidth is discussed in the next section.

- **Duration**

  How long do you want to sample the signal for? If you are storing data to memory or to a disk file, then the duration determines the storage resources required. The format of the stored data also affects the amount of storage space required. For example, data stored in ASCII format takes more space than data stored in binary format.

**Sensor Bandwidth**

In a real-world data acquisition experiment, the physical phenomena you are measuring has expected limits. For example, the temperature of your automobile’s cooling system varies continuously between its low limit and high limit. The temperature limits, as well as how rapidly the temperature varies between the limits, depends on several factors including your driving habits, the weather, and the condition of the cooling system. The expected limits might be readily approximated, but there are an infinite number of possible temperatures that you can measure at a given time. As explained in “Quantization” on page 1-24, these unlimited possibilities are mapped to a finite set of values by your data acquisition hardware.

The bandwidth is given by the range of frequencies present in the signal being measured. You can also think of bandwidth as being related to the rate of change of the signal. A slowly varying signal has a low bandwidth, while a rapidly varying signal has a high bandwidth. To properly measure the physical phenomena of interest, the sensor bandwidth must be compatible with the measurement bandwidth.

You might want to use sensors with the widest possible bandwidth when making any physical measurement. This is the one way to ensure that the basic measurement system is capable of responding linearly over the full range of interest. However, the wider the bandwidth of the sensor, the
more you must be concerned with eliminating sensor response to unwanted frequency components.

**Signal Conditioning**

Sensor signals are often incompatible with data acquisition hardware. To overcome this incompatibility, the sensor signal must be conditioned. The type of signal conditioning required depends on the sensor you are using. For example, a signal might have a small amplitude and require amplification, or it might contain unwanted frequency components and require filtering. Common ways to condition signals include

- Amplification
- Filtering
- Electrical isolation
- Multiplexing
- Excitation source

**Amplification**

Low-level – less than around 100 millivolts – usually need to be amplified. High-level signals might also require amplification depending on the input range of the analog input subsystem.

For example, the output signal from a thermocouple is small and must be amplified before it is digitized. Signal amplification allows you to reduce noise and to make use of the full range of your hardware thereby increasing the resolution of the measurement.

**Filtering**

Filtering removes unwanted noise from the signal of interest. A noise filter is used on slowly varying signals such as temperature to attenuate higher frequency signals that can reduce the accuracy of your measurement.

Rapidly varying signals such as vibration often require a different type of filter known as an antialiasing filter. An antialiasing filter removes undesirable higher frequencies that might lead to erroneous measurements.
**Electrical Isolation**

If the signal of interest contains high-voltage transients that could damage the computer, then the sensor signals should be electrically isolated from the computer for safety purposes.

You can also use electrical isolation to make sure that the readings from the data acquisition hardware are not affected by differences in ground potentials. For example, when the hardware device and the sensor signal are each referenced to ground, problems occur if there is a potential difference between the two grounds. This difference can lead to a ground loop, which might lead to erroneous measurements. Using electrically isolated signal conditioning modules eliminates the ground loop and ensures that the signals are accurately represented.

**Multiplexing**

A common technique for measuring several signals with a single measuring device is multiplexing.

Signal conditioning devices for analog signals often provide multiplexing for use with slowly changing signals such as temperature. This is in addition to any built-in multiplexing on the DAQ board. The A/D converter samples one channel, switches to the next channel and samples it, switches to the next channel, and so on. Because the same A/D converter is sampling many channels, the effective sampling rate of each individual channel is inversely proportional to the number of channels sampled.

You must take care when using multiplexers so that the switched signal has sufficient time to settle. Refer to “Noise” on page 1-38 for more information about settling time.

**Excitation Source**

Some sensors require an excitation source to operate. For example, strain gauges, and resistive temperature devices (RTDs) require external voltage or current excitation. Signal conditioning modules for these sensors usually provide the necessary excitation. RTD measurements are usually made with a current source that converts the variation in resistance to a measurable voltage.
**The Computer**

The computer provides a processor, a system clock, a bus to transfer data, and memory and disk space to store data.

The processor controls how fast data is accepted by the converter. The system clock provides time information about the acquired data. Knowing that you recorded a sensor reading is generally not enough. You also need to know when that measurement occurred.

Data is transferred from the hardware to system memory via dynamic memory access (DMA) or interrupts. DMA is hardware controlled and therefore extremely fast. Interrupts might be slow because of the latency time between when a board requests interrupt servicing and when the computer responds. The maximum acquisition rate is also determined by the computer’s bus architecture. Refer to “How Are Acquired Samples Clocked?” on page 1-27 for more information about DMA and interrupts.

**Software**

Regardless of the hardware you are using, you must send information to the hardware and receive information from the hardware. You send configuration information to the hardware such as the sampling rate, and receive information from the hardware such as data, status messages, and error messages. You might also need to supply the hardware with information so that you can integrate it with other hardware and with computer resources. This information exchange is accomplished with software.

There are two kinds of software:

- Driver software
- Application software
For example, suppose you are using Data Acquisition Toolbox with a National Instruments AT-MIO-16E-1 board and its associated NI-DAQ driver. The relationship between you, the driver software, the application software, and the hardware is shown below.

The diagram illustrates that you supply information to the hardware, and you receive information from the hardware.

**Driver Software**

For data acquisition device, there is associated driver software that you must use. Driver software allows you to access and control the capabilities of your hardware. Among other things, basic driver software allows you to

- Bring data on to and get data off of the board
- Control the rate at which data is acquired
- Integrate the data acquisition hardware with computer resources such as processor interrupts, DMA, and memory
- Integrate the data acquisition hardware with signal conditioning hardware
- Access multiple subsystems on a given data acquisition board
- Access multiple data acquisition boards
**Application Software**

Application software provides a convenient front end to the driver software. Basic application software allows you to

- Report relevant information such as the number of samples acquired
- Generate events
- Manage the data stored in computer memory
- Condition a signal
- Plot acquired data

With some application software, you can also perform analysis on the data. MATLAB and Data Acquisition Toolbox provide you with these capabilities and more.
Analog Input Subsystem

In this section...

“Function of the Analog Input Subsystem” on page 1-20
“Sampling” on page 1-20
“Quantization” on page 1-24
“Channel Configuration” on page 1-28
“Transferring Data from Hardware to System Memory” on page 1-31

Function of the Analog Input Subsystem

Many data acquisition hardware devices contain one or more subsystems that convert (digitize) real-world sensor signals into numbers your computer can read. Such devices are called analog input subsystems (AI subsystems, A/D converters, or ADCs). After the real-world signal is digitized, you can analyze it, store it in system memory, or store it to a disk file.

The function of the analog input subsystem is to sample and quantize the analog signal using one or more channels. You can think of a channel as a path through which the sensor signal travels. Typical analog input subsystems have eight or 16 input channels available to you. After data is sampled and quantized, it must be transferred to system memory.

Analog signals are continuous in time and in amplitude (within predefined limits). Sampling takes a “snapshot” of the signal at discrete times, while quantization divides the voltage (or current) value into discrete amplitudes. Sampling, quantization, channel configuration, and transferring data from hardware to system memory are discussed next.

Sampling

Sampling takes a snapshot of the sensor signal at discrete times. For most applications, the time interval between samples is kept constant (for example, sample every millisecond) unless externally clocked.

For most digital converters, sampling is performed by a sample and hold (S/H) circuit. An S/H circuit usually consists of a signal buffer followed by an
Analog Input Subsystem

electronic switch connected to a capacitor. The operation of an S/H circuit follows these steps:

1 At a given sampling instant, the switch connects the buffer and capacitor to an input.

2 The capacitor is charged to the input voltage.

3 The charge is held until the A/D converter digitizes the signal.

4 For multiple channels connected (multiplexed) to one A/D converter, the previous steps are repeated for each input channel.

5 The entire process is repeated for the next sampling instant.

A multiplexer, S/H circuit, and A/D converter are illustrated in the next section.

Hardware can be divided into two main categories based on how signals are sampled: scanning hardware, which samples input signals sequentially, and simultaneous sample and hold (SS/H) hardware, which samples all signals at the same time. These two types of hardware are discussed below.

**Scanning Hardware**

Scanning hardware samples a single input signal, converts that signal to a digital value, and then repeats the process for every input channel used. In other words, each input channel is sampled sequentially. A scan occurs when each input in a group is sampled once.
As shown below, most data acquisition devices have one A/D converter that is multiplexed to multiple input channels.

Therefore, if you use multiple channels, those channels cannot be sampled simultaneously and a time gap exists between consecutive sampled channels. This time gap is called the channel skew. You can think of the channel skew as the time it takes the analog input subsystem to sample a single channel.

Additionally, the maximum sampling rate your hardware is rated at typically applies for one channel. Therefore, the maximum sampling rate per channel is given by the formula

\[
\text{Maximum sampling rate per channel} = \frac{\text{Maximum board rate}}{\text{Number of channels scanned}}
\]
Typically, you can achieve this maximum rate only under ideal conditions. In practice, the sampling rate depends on several characteristics of the analog input subsystem including the settling time and the gain, as well as the channel skew. The sample period and channel skew for a multichannel configuration using scanning hardware is shown below.

If you cannot tolerate channel skew in your application, you must use hardware that allows simultaneous sampling of all channels. Simultaneous sample and hold hardware is discussed in the next section.

**Simultaneous Sample and Hold Hardware**

Simultaneous sample and hold (SS/H) hardware samples all input signals at the same time and holds the values until the A/D converter digitizes all the signals. For high-end systems, there can be a separate A/D converter for each input channel.

For example, suppose you need to simultaneously measure the acceleration of multiple accelerometers to determine the vibration of some device under test. To do this, you must use SS/H hardware because it does not have a channel skew. In general, you might need to use SS/H hardware if your sensor signal changes significantly in a time that is less than the channel skew, or if you need to use a transfer function or perform a frequency domain correlation.
Introduction to Data Acquisition

The sample period for a multichannel configuration using SS/H hardware is shown below. Note that there is no channel skew.

Quantization

As discussed in the previous section, sampling takes a snapshot of the input signal at an instant of time. When the snapshot is taken, the sampled analog signal must be converted from a voltage value to a binary number that the computer can read. The conversion from an infinitely precise amplitude to a binary number is called quantization.

During quantization, the A/D converter uses a finite number of evenly spaced values to represent the analog signal. The number of different values is determined by the number of bits used for the conversion. Most modern converters use 12 or 16 bits. Typically, the converter selects the digital value that is closest to the actual sampled value.
The figure below shows a 1 Hz sine wave quantized by a 3 bit A/D converter.

The number of quantized values is given by $2^3 = 8$, the largest representable value is given by $111 = 2^2 + 2^1 + 2^0 = 7.0$, and the smallest representable value is given by $000 = 0.0$.

**Quantization Error**
There is always some error associated with the quantization of a continuous signal. Ideally, the maximum quantization error is ±0.5 least significant bits (LSBs), and over the full input range, the average quantization error is zero.
As shown below, the quantization error for the previous sine wave is calculated by subtracting the actual signal from the quantized signal.

![Quantization Error Graph]

### Input Range and Polarity

The *input range* of the analog input subsystem is the span of input values for which a conversion is valid. You can change the input range by selecting a different *gain* value. For example, National Instruments’ AT-MIO-16E-1 board has eight gain values ranging from 0.5 to 100. Many boards include a programmable gain amplifier that allows you to change the device gain through software.

When an input signal exceeds the valid input range of the converter, an *overrange* condition occurs. In this case, most devices saturate to the largest representable value, and the converted data is almost definitely incorrect. The gain setting affects the precision of your measurement — the higher (lower) the gain value, the lower (higher) the precision. Refer to “How Are Range, Gain, and Measurement Precision Related?” on page 1-37 for more information about how input range, gain, and precision are related to each other.
Analog Input Subsystem

An analog input subsystem can typically convert both unipolar signals and bipolar signals. A unipolar signal contains only positive values and zero, while a bipolar signal contains positive values, negative values, and zero.

Unipolar and bipolar signals are depicted below. Refer to the figure in “Quantization” on page 1-24 for an example of a unipolar signal.

In many cases, the signal polarity is a fixed characteristic of the sensor and you must configure the input range to match this polarity.

As you can see, it is crucial to understand the range of signals expected from your sensor so that you can configure the input range of the analog input subsystem to maximize resolution and minimize the chance of an overrange condition.

**How Are Acquired Samples Clocked?**

Samples are acquired from an analog input subsystem at a specific rate by a clock. Like any timing system, data acquisition clocks are characterized their resolution and accuracy. Timing resolution is defined as the smallest time interval that you can accurately measure. The timing accuracy is affected by clock jitter. Jitter arises when a clock produces slightly different values for a given time interval.

For any data acquisition system, there are typically three clock sources that you can use: the onboard data acquisition clock, the computer clock, or an
external clock. Data Acquisition Toolbox supports all of these clock sources, depending on the requirements of your hardware.

**Onboard Clock.** The onboard clock is typically a timer chip on the hardware board that is programmed to generate a pulse stream at the desired rate. The onboard clock generally has high accuracy and low jitter compared to the computer clock. You should always use the onboard clock when the sampling rate is high, and when you require a fixed time interval between samples. The onboard clock is referred to as the *internal clock* in this guide.

**Computer Clock.** The computer (PC) clock is used for boards that do not possess an onboard clock. The computer clock is less accurate and has more jitter than the onboard clock, and is generally limited to sampling rates below 500 Hz. The computer clock is referred to as the *software clock* in this guide.

**External Clock.** An external clock is often used when the sampling rate is low and not constant. For example, an external clock source is often used in automotive applications where samples are acquired as a function of crank angle.

**Channel Configuration**

You can configure input channels in one of these two ways:

- Differential
- Single-ended

Your choice of input channel configuration might depend on whether the input signal is *floating* or *grounded*.

A floating signal uses an isolated ground reference and is not connected to the building ground. As a result, the input signal and hardware device are not connected to a common reference, which can cause the input signal to exceed the valid range of the hardware device. To circumvent this problem, you must connect the signal to the onboard ground of the device. Examples of floating signal sources include ungrounded thermocouples and battery devices.

A grounded signal is connected to the building ground. As a result, the input signal and hardware device are connected to a common reference. Examples of
grounded signal sources include nonisolated instrument outputs and devices that are connected to the building power system.

**Note** For more information about channel configuration, refer to your hardware documentation.

**Differential Inputs**

When you configure your hardware for differential input, there are two signal wires associated with each input signal — one for the input signal and one for the reference (return) signal. The measurement is the difference in voltage between the two wires, which helps reduce noise and any voltage that is common to both wires.

As shown below, the input signal is connected to the positive amplifier socket (labeled +) and the return signal is connected to the negative amplifier socket (labeled -). The amplifier has a third connector that allows these signals to be referenced to ground.

![Diagram of Differential Inputs](image)

National Instruments recommends that you use differential inputs under any of these conditions:

- The input signal is low level (less than 1 volt).
- The leads connecting the signal are greater than 10 feet.
- The input signal requires a separate ground-reference point or return signal.
• The signal leads travel through a noisy environment.

**Single-Ended Inputs**

When you configure your hardware for single-ended input, there is one signal wire associated with each input signal, and each input signal is connected to the same ground. Single-ended measurements are more susceptible to noise than differential measurements because of differences in the signal paths.

As shown below, the input signal is connected to the positive amplifier socket (labeled +) and the ground is connected to the negative amplifier socket (labeled -).

![Diagram of single-ended input configuration]

National Instruments suggests that you can use single-ended inputs under any of these conditions:

• The input signal is high level (greater than 1 volt).
• The leads connecting the signal are less than 10 feet.
• The input signal can share a common reference point with other signals.

You should use differential input connectors for any input signal that does not meet the preceding conditions. You can configure many National Instruments boards for two different types of single-ended connections:

• Referenced single-ended (RSE) connection

  The RSE configuration is used for floating signal sources. In this case, the hardware device itself provides the reference ground for the input signal.
• Nonreferenced single-ended (NRSE) connection

The NRSE input configuration is used for grounded signal sources. In this case, the input signal provides its own reference ground and the hardware device should not supply one.

Refer to your National Instruments hardware documentation for more information about RSE and NRSE connections.

**Transferring Data from Hardware to System Memory**

The transfer of acquired data from the hardware to system memory follows these steps:

1. Acquired data is stored in the hardware’s first-in first-out (FIFO) buffer.

2. Data is transferred from the FIFO buffer to system memory using interrupts or DMA.

These steps happen automatically. Typically, all that’s required from you is some initial configuration of the hardware device when it is installed.

**FIFO Buffer**

The FIFO buffer is used to temporarily store acquired data. The data is temporarily stored until it can be transferred to system memory. The process of transferring data into and out of an analog input FIFO buffer is given below:

1. The FIFO buffer stores newly acquired samples at a constant sampling rate.

2. Before the FIFO buffer is filled, the software starts removing the samples. For example, an interrupt is generated when the FIFO is half full, and signals the software to extract the samples as quickly as possible.

3. Because servicing interrupts or programming the DMA controller can take up to a few milliseconds, additional data is stored in the FIFO for future retrieval. For a larger FIFO buffer, longer latencies can be tolerated.

4. The samples are transferred to system memory via the system bus (for example, PCI bus or AT bus). After the samples are transferred, the software is free to perform other tasks until the next interrupt occurs. For
example, the data can be processed or saved to a disk file. As long as the average rates of storing and extracting data are equal, acquired data will not be missed and your application should run smoothly.

**Interrupts**

The slowest but most common method to move acquired data to system memory is for the board to generate an interrupt request (IRQ) signal. This signal can be generated when one sample is acquired or when multiple samples are acquired. The process of transferring data to system memory via interrupts is given below:

1. When data is ready for transfer, the CPU stops whatever it is doing and runs a special interrupt handler routine that saves the current machine registers, and then sets them to access the board.

2. The data is extracted from the board and placed into system memory.

3. The saved machine registers are restored, and the CPU returns to the original interrupted process.

The actual data move is fairly quick, but there is a lot of overhead time spent saving, setting up, and restoring the register information. Therefore, depending on your specific system, transferring data by interrupts might not be a good choice when the sampling rate is greater than around 5 kHz.

**DMA**

Direct memory access (DMA) is a system whereby samples are automatically stored in system memory while the processor does something else. The process of transferring data via DMA is given below:

1. When data is ready for transfer, the board directs the system DMA controller to put it into in system memory as soon as possible.

2. As soon as the CPU is able (which is usually very quickly), it stops interacting with the data acquisition hardware and the DMA controller moves the data directly into memory.

3. The DMA controller gets ready for the next sample by pointing to the next open memory location.
4 The previous steps are repeated indefinitely, with data going to each open memory location in a continuously circulating buffer. No interaction between the CPU and the board is needed.

Your computer supports several different DMA channels. Depending on your application, you can use one or more of these channels. For example, simultaneous input and output with a sound card requires one DMA channel for the input and another DMA channel for the output.
Making Quality Measurements

What Do You Measure?
For most data acquisition applications, you need to measure the signal produced by a sensor at a specific rate.

In many cases, the sensor signal is a voltage level that is proportional to the physical phenomena of interest (for example, temperature, pressure, or acceleration). If you are measuring slowly changing (quasi-static) phenomena like temperature, a slow sampling rate usually suffices. If you are measuring rapidly changing (dynamic) phenomena like vibration or acoustic measurements, a fast sampling rate is required.

To make high-quality measurements, you should follow these rules:

- Maximize the precision and accuracy
- Minimize the noise
- Match the sensor range to the A/D range

Accuracy and Precision
Whenever you acquire measured data, you should make every effort to maximize its accuracy and precision. The quality of your measurement depends on the accuracy and precision of the entire data acquisition system, and can be limited by such factors as board resolution or environmental noise.

In general terms, the accuracy of a measurement determines how close the measurement comes to the true value. Therefore, it indicates the correctness
of the result. The *precision* of a measurement reflects how exactly the result is determined without reference to what the result means. The *relative precision* indicates the uncertainty in a measurement as a fraction of the result.

For example, suppose you measure a table top with a meter stick and find its length to be 1.502 meters. This number indicates that the meter stick (and your eyes) can resolve distances down to at least a millimeter. Under most circumstances, this is considered to be a fairly precise measurement with a relative precision of around 1/1500. However, suppose you perform the measurement again and obtain a result of 1.510 meters. After careful consideration, you discover that your initial technique for reading the meter stick was faulty because you did not read it from directly above. Therefore, the first measurement was not accurate.

Precision and accuracy are illustrated below.

For analog input subsystems, accuracy is usually limited by calibration errors while precision is usually limited by the A/D converter. Accuracy and precision are discussed in more detail below.
Accuracy

Accuracy is defined as the agreement between a measured quantity and the true value of that quantity. Every component that appears in the analog signal path affects system accuracy and performance. The overall system accuracy is given by the component with the worst accuracy.

For data acquisition hardware, accuracy is often expressed as a percent or a fraction of the least significant bit (LSB). Under ideal circumstances, board accuracy is typically ±0.5 LSB. Therefore, a 12 bit converter has only 11 usable bits.

Many boards include a programmable gain amplifier, which is located just before the converter input. To prevent system accuracy from being degraded, the accuracy and linearity of the gain must be better than that of the A/D converter. The specified accuracy of a board is also affected by the sampling rate and the settling time of the amplifier. The settling time is defined as the time required for the instrumentation amplifier to settle to a specified accuracy. To maintain full accuracy, the amplifier output must settle to a level given by the magnitude of 0.5 LSB before the next conversion, and is on the order of several tenths of a millisecond for most boards.

Settling time is a function of sampling rate and gain value. High rate, high gain configurations require longer settling times while low rate, low gain configurations require shorter settling times.

Precision

The number of bits used to represent an analog signal determines the precision (resolution) of the device. The more bits provided by your board, the more precise your measurement will be. A high precision, high resolution device divides the input range into more divisions thereby allowing a smaller detectable voltage value. A low precision, low resolution device divides the input range into fewer divisions thereby increasing the detectable voltage value.

The overall precision of your data acquisition system is usually determined by the A/D converter, and is specified by the number of bits used to represent the analog signal. Most boards use 12 or 16 bits. The precision of your measurement is given by
Making Quality Measurements

Precision = one part in \(2^{\text{number of bits}}\)

The precision in volts is given by

\[
\text{Precision} = \text{voltage range}/2^{\text{number of bits}}
\]

For example, if you are using a 12 bit A/D converter configured for a 10 volt range, then

\[
\text{Precision} = 10 \text{ volts}/2^{12}
\]

This means that the converter can detect voltage differences at the level of 0.00244 volts (2.44 mV).

**How Are Range, Gain, and Measurement Precision Related?**

When you configure the input range and gain of your analog input subsystem, the end result should maximize the measurement resolution and minimize the chance of an overrange condition. The actual input range is given by the formula

\[
\text{Actual input range} = \text{Input range}/\text{Gain}
\]

The relationship between gain, actual input range, and precision for a unipolar and bipolar signal having an input range of 10 V is shown below.
Table 1-2  Relationship Between Input Range, Gain, and Precision

<table>
<thead>
<tr>
<th>Input Range</th>
<th>Gain</th>
<th>Actual Input Range</th>
<th>Precision (12 Bit A/D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 to 10 V</td>
<td>1.0</td>
<td>0 to 10 V</td>
<td>2.44 mV</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>0 to 5 V</td>
<td>1.22 mV</td>
</tr>
<tr>
<td></td>
<td>5.0</td>
<td>0 to 2 V</td>
<td>0.488 mV</td>
</tr>
<tr>
<td></td>
<td>10.0</td>
<td>0 to 1 V</td>
<td>0.244 mV</td>
</tr>
<tr>
<td>-5 to 5 V</td>
<td>0.5</td>
<td>-10 to 10 V</td>
<td>4.88 mV</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>-5 to 5 V</td>
<td>2.44 mV</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>-2.5 to 2.5 V</td>
<td>1.22 mV</td>
</tr>
<tr>
<td></td>
<td>5.0</td>
<td>-1.0 to 1.0 V</td>
<td>0.488 mV</td>
</tr>
<tr>
<td></td>
<td>10.0</td>
<td>-0.5 to 0.5 V</td>
<td>0.244 mV</td>
</tr>
</tbody>
</table>

As shown in the table, the gain affects the precision of your measurement. If you select a gain that decreases the actual input range, then the precision increases. Conversely, if you select a gain that increases the actual input range, then the precision decreases. This is because the actual input range varies but the number of bits used by the A/D converter remains fixed.

**Note** With Data Acquisition Toolbox, you do not have to specify the range and gain. Instead, you simply specify the actual input range desired.

**Noise**

Noise is considered to be any measurement that is not part of the phenomena of interest. Noise can be generated within the electrical components of the input amplifier (internal noise), or it can be added to the signal as it travels down the input wires to the amplifier (external noise). Techniques that you can use to reduce the effects of noise are described below.
Removing Internal Noise
Internal noise arises from thermal effects in the amplifier. Amplifiers typically generate a few microvolts of internal noise, which limits the resolution of the signal to this level. The amount of noise added to the signal depends on the bandwidth of the input amplifier.

To reduce internal noise, you should select an amplifier with a bandwidth that closely matches the bandwidth of the input signal.

Removing External Noise
External noise arises from many sources. For example, many data acquisition experiments are subject to 60 Hz noise generated by AC power circuits. This type of noise is referred to as pick-up or hum, and appears as a sinusoidal interference signal in the measurement circuit. Another common interference source is fluorescent lighting. These lights generate an arc at twice the power line frequency (120 Hz).

Noise is added to the acquisition circuit from these external sources because the signal leads act as aerials picking up environmental electrical activity. Much of this noise is common to both signal wires. To remove most of this common-mode voltage, you should

- Configure the input channels in differential mode. Refer to “Channel Configuration” on page 1-28 for more information about channel configuration.
- Use signal wires that are twisted together rather than separate.
- Keep the signal wires as short as possible.
- Keep the signal wires as far away as possible from environmental electrical activity.

Filtering
Filtering also reduces signal noise. For many data acquisition applications, a low-pass filter is beneficial. As the name suggests, a low-pass filter passes the lower frequency components but attenuates the higher frequency components. The cut-off frequency of the filter must be compatible with the frequencies present in the signal of interest and the sampling rate used for the A/D conversion.
A low-pass filter that’s used to prevent higher frequencies from introducing distortion into the digitized signal is known as an antialiasing filter if the cut-off occurs at the Nyquist frequency. That is, the filter removes frequencies greater than one-half the sampling frequency. These filters generally have a sharper cut-off than the normal low-pass filter used to condition a signal. Antialiasing filters are specified according to the sampling rate of the system and there must be one filter per input signal.

**Matching the Sensor Range and A/D Converter Range**

When sensor data is digitized by an A/D converter, you must be aware of these two issues:

- The expected range of the data produced by your sensor. This range depends on the physical phenomena you are measuring and the output range of the sensor.
- The range of your A/D converter. For many devices, the hardware range is specified by the gain and polarity.

You should select the sensor and hardware ranges such that the maximum precision is obtained, and the full dynamic range of the input signal is covered.

For example, suppose you are using a microphone with a dynamic range of 20 dB to 140 dB and an output sensitivity of 50 mV/Pa. If you are measuring street noise in your application, then you might expect that the sound level never exceeds 80 dB, which corresponds to a sound pressure magnitude of 200 mPa and a voltage output from the microphone of 10 mV. Under these conditions, you should set the input range of your data acquisition card for a maximum signal amplitude of 10 mV, or a little more.

**How Fast Should a Signal Be Sampled?**

Whenever a continuous signal is sampled, some information is lost. The key objective is to sample at a rate such that the signal of interest is well characterized and the amount of information lost is minimized.

If you sample at a rate that is too slow, then signal aliasing can occur. Aliasing can occur for both rapidly varying signals and slowly varying signals. For
example, suppose you are measuring temperature once a minute. If your
acquisition system is picking up a 60-Hz hum from an AC power supply, then
that hum will appear as constant noise level if you are sampling at 30 Hz.

Aliasing occurs when the sampled signal contains frequency components
greater than one-half the sampling rate. The frequency components could
originate from the signal of interest in which case you are undersampling and
should increase the sampling rate. The frequency components could also
originate from noise in which case you might need to condition the signal
using a filter. The rule used to prevent aliasing is given by the Nyquist
theorem, which states that

- An analog signal can be uniquely reconstructed, without error, from
samples taken at equal time intervals.
- The sampling rate must be equal to or greater than twice the highest
frequency component in the analog signal. A frequency of one-half the
sampling rate is called the Nyquist frequency.

However, if your input signal is corrupted by noise, then aliasing can still
occur.
For example, suppose you configure your A/D converter to sample at a rate of 4 samples per second (4 S/s or 4 Hz), and the signal of interest is a 1 Hz sine wave. Because the signal frequency is one-fourth the sampling rate, then according to the Nyquist theorem, it should be completely characterized. However, if a 5 Hz sine wave is also present, then these two signals cannot be distinguished. In other words, the 1 Hz sine wave produces the same samples as the 5 Hz sine wave when the sampling rate is 4 S/s. This situation is shown below.

In a real-world data acquisition environment, you might need to condition the signal by filtering out the high frequency components.

Even though the samples appear to represent a sine wave with a frequency of one-fourth the sampling rate, the actual signal could be any sine wave with a frequency of

\[(n \pm 0.25) \times (\text{Sampling rate})\]
where \( n \) is zero or any positive integer. For this example, the actual signal could be at a frequency of 3 Hz, 5 Hz, 7 Hz, 9 Hz, and so on. The relationship \( 0.25 \times \text{Sampling rate} \) is called the alias of a signal that may be at another frequency. In other words, aliasing occurs when one frequency assumes the identity of another frequency.

If you sample the input signal at least twice as fast as the highest frequency component, then that signal might be uniquely characterized, but this rate would not mimic the waveform very closely. As shown below, to get an accurate picture of the waveform, you need a sampling rate of roughly 10 to 20 times the highest frequency.

As shown in the top figure, the low sampling rate produces a sampled signal that appears to be a triangular waveform. As shown in the bottom figure, a higher fidelity sampled signal is produced when the sampling rate is higher. In the latter case, the sampled signal actually looks like a sine wave.
How Can Aliasing Be Eliminated?
The primary considerations involved in antialiasing are the sampling rate of the A/D converter and the frequencies present in the sampled data. To eliminate aliasing, you must

- Establish the useful bandwidth of the measurement.
- Select a sensor with sufficient bandwidth.
- Select a low-pass anti-aliasing analog filter that can eliminate all frequencies exceeding this bandwidth.
- Sample the data at a rate at least twice that of the filter’s upper cutoff frequency.
**Getting Command-Line Function Help**

To get command-line function help, you should use the `daqhelp` function. For example, to get help for the `addchannel` function, type

```
daqhelp addchannel
```

Alternatively, you can use the `help` command.

```
help addchannel
```

However, Data Acquisition Toolbox provides “overloaded” versions of several MATLAB functions. That is, it provides toolbox-specific implementations of these functions using the same function name. To get command-line help for an overloaded toolbox function using the `help` command, you must supply one of two possible class directories to `help`:

```
help daqdevice/function_name
help daqchild/function_name
```

Note that the same help information is returned regardless of the class directory specified.

For example, Data Acquisition Toolbox provides an overloaded version of the `delete` function. To obtain help for the MATLAB version of this function, type

```
help delete
```

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 daqdevice/delete.m
help daqchild/delete.m
```

So, to obtain help on the toolbox version of this function, type

```
help daqdevice/delete
```
Selected Bibliography


Getting Started Using Data Acquisition Toolbox

This chapter provides the information you need to get started with Data Acquisition Toolbox. The sections are as follows.

- Installation Information (p. 2-2)  How to determine whether the toolbox is installed on your system
- Toolbox Components (p. 2-4)  The M-files and hardware driver adaptors that compose the toolbox
- Accessing Your Hardware (p. 2-12)  Examples that show you how to acquire data, output data, and read and write digital values
- Understanding the Toolbox Capabilities (p. 2-17)  Resources to help you understand the toolbox capabilities including demos and documentation examples
- Examining Your Hardware Resources (p. 2-19)  Return hardware-related information visible to the toolbox including the installed adaptors and the syntax for creating device objects
- Getting Help (p. 2-23)  Get help using the Help browser, M-file help, and other methods
Installation Information

In this section...

“Before You Install” on page 2-2
“Toolbox Installation” on page 2-2
“Hardware and Driver Installation” on page 2-3

Before You Install

To acquire live, measured data into the MATLAB environment, or to output
data from the MATLAB environment, you must install these components:

- MATLAB
- Data Acquisition Toolbox
- A supported data acquisition device — For a complete listing of all
  supported devices, visit the Data Acquisition Toolbox section of the
- Software such as drivers and support libraries, as required by your data
  acquisition device

Toolbox Installation

To determine if Data Acquisition Toolbox is installed on your system, type

\texttt{ver}

at the MATLAB prompt. MATLAB displays information about the versions
of MATLAB you are running, including a list of installed add-on products
and their version numbers. Check the list to see if Data Acquisition Toolbox
appears. For information about installing the toolbox, see the MATLAB
Installation Guide for your platform.

If you experience installation difficulties and have Web access, look for the
license manager and installation information at the MathWorks Web site
(http://www.mathworks.com).
Hardware and Driver Installation
Installation of your hardware device, hardware drivers, and any other
device-specific software is described in the documentation provided by your
hardware vendor.

Note You need to install all necessary device-specific software provided by
your hardware vendor in addition to Data Acquisition Toolbox.
Toolbox Components

<table>
<thead>
<tr>
<th>In this section...</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Toolbox Components: Information and Interaction” on page 2-4</td>
</tr>
<tr>
<td>“M-File Functions” on page 2-6</td>
</tr>
<tr>
<td>“Data Acquisition Engine” on page 2-6</td>
</tr>
<tr>
<td>“Hardware Driver Adaptor” on page 2-9</td>
</tr>
<tr>
<td>“Supported Hardware” on page 2-9</td>
</tr>
<tr>
<td>“Unsupported Hardware” on page 2-11</td>
</tr>
</tbody>
</table>

Toolbox Components: Information and Interaction

Data Acquisition Toolbox consists of three distinct components: M-file functions, the data acquisition engine, and hardware driver adaptors. As shown below, these components allow you to pass information between MATLAB and your data acquisition hardware.
The preceding diagram illustrates how information flows from component to component. Information consists of

- **Property values**

  You can control the behavior of your data acquisition application by configuring property values. In general, you can think of a property as a characteristic of the toolbox or of the hardware driver that can be manipulated to suit your needs.
• **Data**

You can acquire data from a sensor connected to an analog input subsystem and store it in MATLAB, or output data from MATLAB to an actuator connected to an analog output subsystem. Additionally you can transfer values (1s and 0s) between MATLAB and a digital I/O subsystem.

• **Events**

An event occurs at a particular time after a condition is met and might result in one or more callbacks that you specify. Events can be generated only after you configure the associated properties. Some of the ways you can use events include initiating analysis after a predetermined amount of data is acquired, or displaying a message to the MATLAB workspace after an error occurs.

**M-File Functions**

To perform any task with your data acquisition application, you must call M-file functions from the MATLAB environment. Among other things, these functions allow you to

• Create device objects, which provide a gateway to your hardware’s capabilities and allow you to control the behavior of your application.

• Acquire or output data.

• Configure property values.

• Evaluate your acquisition status and hardware resources.

For a listing of all Data Acquisition Toolbox functions, refer to Chapter 12, “Functions — Alphabetical List”. You can also display all the toolbox functions by typing

```matlab
help daq
```

**Data Acquisition Engine**

The data acquisition engine (or just engine) is a MEX-file (shared library that is executable within MATLAB) that

• Stores the device objects and associated property values that control your data acquisition application
• Controls the synchronization of events
• Controls the storage of acquired or queued data

While the engine performs these tasks, you can use MATLAB for other tasks such as analyzing acquired data. In other words, the engine and MATLAB are asynchronous. The relationship between acquiring data, outputting data, and data flow is described below.

**Flow of Acquired Data**

Acquiring data means that data is flowing from your hardware device into the data acquisition engine where it is temporarily stored in memory. The data is stored temporarily because it can be overwritten. The rate at which the data is overwritten depends on several factors including the available memory, the rate at which data is acquired, and the number of hardware channels from which data is acquired.

The stored data is not automatically available in the MATLAB workspace. Instead, you must explicitly extract data from the engine using the `getdata` function.

The flow of acquired data consists of these two independent steps:

1. Data acquired from the hardware is stored in the engine.
2. Data is extracted from the engine and stored in MATLAB, or output to a disk file.

These two steps are illustrated below.
Flow of Output Data

Outputting data means that data is flowing from the data acquisition engine to the hardware device. However, before data is output, you must queue it in the engine with the `putdata` function. The amount of data that you can queue depends on several factors including the available memory, the number of hardware channels to which data is output, and the size of each data sample.

The flow of output data consists of these two independent steps:

1. **Data from MATLAB is queued in the engine.**
2. **Data queued in the engine is output to the hardware.**

These two steps are illustrated below.
Hardware Driver Adaptor

The hardware driver adaptor (or *adaptor*) is the interface between the data acquisition engine and the hardware driver. The adaptor’s main purpose is to pass information between MATLAB and your hardware device via its driver.

Hardware drivers are provided by your device vendor. For example, to acquire data using a National Instruments board, the appropriate version of the NI-DAQ driver must be installed on your platform. For further information about NI-DAQmx and Traditional NI-DAQ drivers, see “NI-DAQmx Versus Traditional NI-DAQ Drivers” on page A-9. Hardware drivers are not installed as part of Data Acquisition Toolbox with the exception of a special parallel port driver that allows access to the port’s protected memory addresses. Additionally, a suitable driver is usually installed on PCs that are equipped with a sound card. For the remaining supported devices, the drivers must be installed.

Supported Hardware

You can obtain most adaptors either from The MathWorks or from the device vendors. See the Data Acquisition Toolbox supported hardware page at www.mathworks.com/products/daq/supportedio.html for the list of vendors whose hardware the toolbox supports, and for information about how to obtain...
an adaptor. Data Acquisition Toolbox provides the adaptors listed below. The
name of the vendor is also listed in the table.

**Adaptor Provided by the Data Acquisition Device**

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Adaptor Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advantech</td>
<td>advantech</td>
</tr>
<tr>
<td>Keithley</td>
<td>keithley</td>
</tr>
<tr>
<td>Measurement Computing</td>
<td>mcc</td>
</tr>
<tr>
<td>National Instruments</td>
<td>nidaq</td>
</tr>
<tr>
<td>Parallel port</td>
<td>parallel</td>
</tr>
<tr>
<td>VXI Technology</td>
<td>hpe1432</td>
</tr>
<tr>
<td>Windows sound cards</td>
<td>winsound</td>
</tr>
</tbody>
</table>

**Note** Additional vendors not listed in this table are listed in the supported
hardware page at www.mathworks.com/products/daq/supportedio.html. This
page contains a comprehensive list of vendors whose hardware the toolbox
supports and it provides information on how to obtain an adaptor.

**Note** Keithley and VXI Technology adaptors will be deprecated in a
future version of the toolbox. If you create a Data Acquisition Toolbox
object for the keithley or hp1432 adaptors in release R2007a or
later, you will receive a warning stating that these adaptors will be
removed in a future release. See the supported hardware page at

As described in “Examining Your Hardware Resources” on page 2-19, you can
list the installed adaptor names with the daqhwinfo function.
Unsupported Hardware

Refer to the Data Acquisition Toolbox supported hardware page at www.mathworks.com/products/daq/supportedio.html for the list of vendors whose hardware the toolbox supports, and for information about how to obtain an adaptor. If the device you are using is not listed on this page, you can do one of the following:

• Contact the device vendor to request them to develop an interface to the toolbox. Refer them to the supported hardware page at www.mathworks.com/products/daq/supportedio.html for a list of currently supported hardware and for information about contacting The MathWorks.

• Search for your device on the MathWorks support page at www.mathworks.com/support/ to see if a solution is listed for using your non-supported device. Such solutions are typically available for devices that the next release of the Data Acquisition Toolbox will support.

• Create the interface yourself. To interface unsupported hardware devices to the toolbox, use the Data Acquisition Toolbox Adaptor Kit installed with the toolbox. For more information about the adaptor kit, read the Data Acquisition Toolbox Adaptor Kit documentation.

• Hire a consultant to write the interface or a systems integrator to build the system. For a potential list of consultants or systems integrators, go to the Third Party Products and Services page at www.mathworks.com/connections.

• Consider using hardware that the Data Acquisition Toolbox already supports.
Accessing Your Hardware

<table>
<thead>
<tr>
<th>In this section...</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Connecting to Your Hardware” on page 2-12</td>
</tr>
<tr>
<td>“Acquiring Data” on page 2-12</td>
</tr>
<tr>
<td>“Outputting Data” on page 2-13</td>
</tr>
<tr>
<td>“Reading and Writing Digital Values” on page 2-14</td>
</tr>
<tr>
<td>“Acquiring Data in a Loop” on page 2-16</td>
</tr>
</tbody>
</table>

Connecting to Your Hardware

Perhaps the most effective way to get started with Data Acquisition Toolbox is to connect to your hardware, and input or output data.

Each example illustrates a typical data acquisition session. The data acquisition session comprises all the steps you are likely to take when acquiring or outputting data using a supported hardware device. You should keep these steps in mind when constructing your own data acquisition applications.

Note that the analog input and analog output examples use a sound card, while the digital I/O example uses a National Instruments PCI-6024E board. If you are using a different supported hardware device, you should modify the adaptor name and the device ID supplied to the creation function as needed.

If you want detailed information about any functions that are used, refer to Chapter 12, “Functions — Alphabetical List”. If you want detailed information about any properties that are used, refer to Chapter 14, “Base Properties — Alphabetical List”.

Acquiring Data

If you have a sound card installed, you can run the following example, which acquires one second of data from two analog input hardware channels, and then plots the acquired data.
You should modify this example to suit your specific application needs. If you want detailed information about acquiring data, refer to Chapter 5, “Doing More with Analog Input”.

1 Create a device object — Create the analog input object ai for a sound card.

   ai = analoginput('winsound');

2 Add channels — Add two hardware channels to ai.

   addchannel(ai,1:2);

3 Configure property values — Configure the sampling rate to 44.1 kHz and collect 1 second of data (44,100 samples) for each channel.

   set(ai,'SampleRate',44100)
   set(ai,'SamplesPerTrigger',44100)

4 Acquire data — Start the acquisition. When all the data is acquired, ai automatically stops executing.

   start(ai)
   wait(ai,2)
   data = getdata(ai);
   plot(data)

5 Clean up — When you no longer need ai, you should remove it from memory and from the MATLAB workspace.

   delete(ai)
   clear ai

### Outputting Data

If you have a sound card installed, you can run the following example, which outputs 1 second of data to two analog output hardware channels.

You should modify this example to suit your specific application needs. If you want detailed information about outputting data, refer to Chapter 6, “Analog Output”.

1 **Create a device object** — Create the analog output object ao for a sound card.

\[
\text{ao} = \text{analogoutput('winsound');}
\]

2 **Add channels** — Add two hardware channels to ao.

\[
\text{addchannel(ao,1:2)};
\]

3 **Configure property values** — Configure the sampling rate to 44.1 kHz for each channel.

\[
\text{set(ao,'SampleRate',44100)}
\]

4 **Output data** — Create 1 second of output data, and queue the data in the engine for eventual output to the analog output subsystem. You must queue one column of data for each hardware channel added.

\[
\text{data = sin(linspace(0,2*pi*500,44100)' );}
\]

\[
\text{putdata(ao,[data data])}
\]

Start the output. When all the data is output, ao automatically stops executing.

\[
\text{start(ao)}
\]

5 **Clean up** — When you no longer need ao, you should remove it from memory and from the MATLAB workspace.

\[
\text{delete(ao)}
\]

\[
\text{clear ao}
\]

**Reading and Writing Digital Values**

If you have a supported National Instruments board with at least eight digital I/O lines, you can run the following example, which outputs digital values, and then reads back those values.

You should modify this example to suit your specific application needs. If you want detailed information about reading and writing digital values, refer to Chapter 7, “Digital Input/Output”.
1 Create a device object — Create the digital I/O object \texttt{dio} for a National Instruments PCI-6024E board with hardware ID 1.

\begin{verbatim}
dio = digitalio('nidaq','Dev1');
\end{verbatim}

2 Add lines — Add eight hardware lines to \texttt{dio}, and configure them for output.

\begin{verbatim}
addline(dio,0:7,'out');
\end{verbatim}

3 Read and write values — Create an array of output values, and write the values to the digital I/O subsystem. Note that reading and writing digital I/O line values typically does not require that you configure specific property values.

\begin{verbatim}
pval = [1 1 1 1 0 1 0 1];
putvalue(dio,pval)
gval = getvalue(dio);
\end{verbatim}

4 Clean up — When you no longer need \texttt{dio}, you should remove it from memory and from the MATLAB workspace.

\begin{verbatim}
delete(dio)
clear dio
\end{verbatim}

\textbf{Note} Digital line values are usually not transferred at a specific rate. Although some specialized boards support clocked I/O, Data Acquisition Toolbox does not support this functionality.
Acquiring Data in a Loop

To make multiple acquisitions using a single analog input object, create a single object and execute the acquisition in a loop. Delete the object at the end of the loop.

```matlab
% Create the object outside of the loop. ai = analoginput('nidaq', 'Dev1'); addchannel(ai, 0); % Execute acquisition. for ii = 1:num_iterations start(ai); wait(ai, 2) data = getdata(ai); plot(data); end % Delete the object out of the loop. delete(ai) clear ai
```

If you are creating the object within the loop, you must delete the object within the loop as well.

```matlab
% Execute acquisition. for ii = 1:num_iterations % Create the object within the loop. ai = analoginput('nidaq', 'Dev1'); addchannel(ai, 0); start(ai); wait(ai, 2) data = getdata(ai); plot(data); % Delete the object within the loop. delete(ai) end clear ai
```
Understanding the Toolbox Capabilities

<table>
<thead>
<tr>
<th>In this section...</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Contents M-File” on page 2-17</td>
</tr>
<tr>
<td>“Documentation Examples” on page 2-17</td>
</tr>
<tr>
<td>“Quick Reference Guide” on page 2-18</td>
</tr>
<tr>
<td>“Demos” on page 2-18</td>
</tr>
</tbody>
</table>

**Contents M-File**

The Contents M-file lists the toolbox functions and demos. You can display this information by typing

```matlab
help daq
```

**Documentation Examples**

This guide provides detailed examples that show you how to acquire or output data. These examples are collected in the index.

Some examples are constructed as mini-applications that illustrate one or two important features of the toolbox and serve as templates so you can see how to build applications that suit your specific needs. These examples are included as toolbox M-files and are treated as demos. You can list all Data Acquisition Toolbox demos by typing

```matlab
help daqdemos
```

All documentation example M-files begin with `daqdoc`. To run an example, type the M-file name at the command line. Note that most analog input (AI) and analog output (AO) examples are written for sound cards. To use these examples with your hardware device, you should modify the adaptor name and the device ID supplied to the creation function as needed.

Additionally, most documentation examples are written for clocked subsystems. However, some supported hardware devices — particularly Measurement Computing devices — do not possess onboard clocks. If the AI or AO subsystem of your hardware device does not have an onboard clock, then these examples will not work. To use the documentation examples, you can
• Input single values using the `getsample` function, or output single values using the `putsample` function.

• Configure the `ClockSource` property to `Software`.

**Quick Reference Guide**

The Quick Reference Guide provides a complete overview of the toolbox capabilities, functions, and properties. You might find it useful to print this guide and keep it handy when using the toolbox. You can access this guide through the Help browser.

**Demos**

The toolbox includes a large collection of tutorial demos, which you can access through the Help browser `Demos` pane. Use the following command to open the Help browser to the toolbox demos:

```matlab
demo toolbox 'Data Acquisition'
```

Note that the analog input and analog output tutorials require that you have a sound card installed. The digital I/O tutorials require that you have a supported National Instruments board with digital I/O capabilities.
Examining Your Hardware Resources

In this section...

“Using the daqhwinfo Function” on page 2-19
“General Toolbox Information” on page 2-19
“Adaptor-Specific Information” on page 2-20
“Device Object Information” on page 2-21

Using the daqhwinfo Function

You can examine the data acquisition hardware resources visible to the toolbox with the daqhwinfo function. Hardware resources include installed boards, hardware drivers, and adaptors. The information returned by daqhwinfo depends on the supplied arguments, and is divided into three categories described in this section.

If you configure hardware parameters using a vendor tool such as National Instruments’ Measurement and Automation Explorer or Measurement Computings’ InstaCal, daqhwinfo will return this configuration information. For example, if you configure your Measurement Computing device for 16 single-ended channels using InstaCal, daqhwinfo returns this configuration. However, the toolbox does not preserve configuration information that is not directly associated with your hardware. For example, channel name information is not preserved. Refer to Appendix A, “Troubleshooting Your Hardware” for more information about using vendor tools.

General Toolbox Information

To display general information about Data Acquisition Toolbox

```matlab
out = daqhwinfo
out =
    ToolboxName: 'Data Acquisition Toolbox'
    ToolboxVersion: '2.2 (R13)'
    MATLABVersion: '6.5 (R13)'
    InstalledAdaptors: {4x1 cell}
```
The `InstalledAdaptors` field lists the hardware driver adaptors installed on your system. To display the installed adaptors:

```matlab
out.InstalledAdaptors
ans =
    'mcc'
    'nidaq'
    'parallel'
    'winsound'
```

This information tells you that an adaptor is available for Measurement Computing and National Instruments devices, parallel ports, and sound cards.

**Note** The list of installed adaptors might be different for your platform. Toolbox adaptors are available to you only if the associated hardware driver is installed.

### Adaptor-Specific Information

To display hardware information for a particular vendor, you must supply the adaptor name as an argument to `daqhwinfo`. The supported vendors and adaptor names are given in . For example, to display hardware information for the `winsound` adaptor:

```matlab
out = daqhwinfo('winsound')
out =
    AdaptorDllName: 'd:\v6\toolbox\daq\daq\private\mwwinsound.dll'
    AdaptorDllVersion: 'Version 2.2  (R13) 01-Jul-2002'
    AdaptorName: 'winsound'
    BoardNames: {'AudioPCI Record'}
    InstalledBoardIds: {'0'}
    ObjectConstructorName:{'analoginput('winsound',0)'}[1x26 char]
```

The `ObjectConstructorName` field lists the subsystems supported by the installed sound cards, and the syntax for creating a device object associated with a given subsystem. To display the device object constructor names available for the AudioPCI Record board:
This information tells you that the sound card supports analog input and analog output objects. To create an analog input object for the sound card

```matlab
ai = analoginput('winsound');
```

To create an analog output object for the sound card

```matlab
ao = analogoutput('winsound');
```

### Device Object Information

To display hardware information for a specific device object, you supply the device object as an argument to `daqhwinfo`. The hardware information for the analog input object `ai` created in the preceding section is given below.

```matlab
out = daqhwinfo(ai)
```

```plaintext
out =

    AdaptorName: 'winsound'
       Bits: 16
         Coupling: {'AC Coupled'}
      DeviceName: 'AudioPCI Record'
DifferentialIDs: []
         Gains: []
           ID: '0'
    InputRanges: [-1 1]
MaxSampleRate: 44100
MinSampleRate: 8000
NativeDataType: 'int16'
          Polarity: {'Bipolar'}
       SampleType: 'SimultaneousSample'
SingleEndedIDs: [1 2]
   SubsystemType: 'AnalogInput'
     TotalChannels: 2
VendorDriverDescription: 'Windows Multimedia Driver'
VendorDriverVersion: '5.0'
```
Among other things, this information tells you that the minimum sampling rate is 8 kHz, the maximum sampling rate is 44.1 kHz, and there are two hardware channels that you can add to the analog input object.

Alternatively, you can return hardware information via the Workspace browser by right-clicking a device object, and selecting **Explore > Display Hardware Info** from the context menu.
Getting Help

The daqhelp Function
You can use the `daqhelp` function to

- Display command-line help for functions and properties
- List all the functions and properties associated with a specific device object

A device object need not exist for you to obtain this information. For example, to display all the functions and properties associated with an analog input object, as well as the constructor M-file help

```
daqhelp analoginput
```

To display help for the `SampleRate` property

```
daqhelp SampleRate
```

You can also display help for an existing device object. For example, to display help for the `BitsPerSample` property for an analog input object associated with a sound card

```
ai = analoginput('winsound');
out = daqhelp(ai,'BitsPerSample');
```

Alternatively, you can display help via the Workspace browser by right-clicking a device object, and selecting Explore > DAQ Help from the context menu.

The propinfo Function
You can use the `propinfo` function to return the characteristics of Data Acquisition Toolbox properties. For example, you can find the default value for any property using this function. `propinfo` returns a structure containing the fields shown below.
Table 2-1  propinfo Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>The property data type. Possible values are callback, any, double, and string.</td>
</tr>
<tr>
<td>Constraint</td>
<td>The type of constraint on the property value. Possible values are callback, bounded, enum, and none.</td>
</tr>
<tr>
<td>ConstraintValue</td>
<td>The property value constraint. The constraint can be a range of valid values or a list of valid string values.</td>
</tr>
<tr>
<td>DefaultValue</td>
<td>The property default value.</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>Indicates when the property is read-only. Possible values are always, never, and whileRunning.</td>
</tr>
<tr>
<td>DeviceSpecific</td>
<td>If the property is device-specific, a 1 is returned. If a 0 is returned, the property is supported for all device objects of a given type.</td>
</tr>
</tbody>
</table>

For example, to return the characteristics for all the properties associated with the analog input object ai created in the preceding section

```matlab
AIIinfo = propinfo(ai);
```

The characteristics for the TriggerType property are displayed below.

```matlab
AIIinfo.TriggerType
ans =
    Type: 'string'
    Constraint: 'enum'
    ConstraintValue: {'Manual' 'Immediate' 'Software'}
    DefaultValue: 'Immediate'
    ReadOnly: 'whileRunning'
    DeviceSpecific: 0
```
This information tells you that

- The property value data type is a string.
- The property value is constrained as an enumerated list of values.
- The three possible property values are Manual, Immediate and Software.
- The default value is Immediate.
- The property is read-only while the device object is running.
- The property is supported for all analog input objects.
Data Acquisition Session

The data acquisition session consists of all the steps you are likely to take when acquiring or outputting data. These steps are described in the following sections.

Understanding the Data Acquisition Session (p. 3-2)  Description of the data acquisition session including a brief example
Creating a Device Object (p. 3-5)  Create a MATLAB object that represents the hardware subsystem
Hardware Channels or Lines (p. 3-9)  Add hardware channels or hardware lines to the device object
Configuring and Returning Properties (p. 3-13)  Define the device object behavior by assigning values to properties
Acquiring and Outputting Data (p. 3-23)  Execute the device object and acquire or output data using the previously added channels
Cleaning Up (p. 3-27)  Remove the device object from memory and from the workspace
Understanding the Data Acquisition Session

<table>
<thead>
<tr>
<th>In this section...</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Overview” on page 3-2</td>
</tr>
<tr>
<td>“Example: The Data Acquisition Session” on page 3-3</td>
</tr>
</tbody>
</table>

Overview

The data acquisition session consists of all the steps you are likely to take when acquiring or outputting data. These steps are

1. **Create a device object** — You create a device object using the `analoginput`, `analogoutput`, or `digitalio` creation function. Device objects are the basic toolbox elements you use to access your hardware device.

2. **Add channels or lines** — After a device object is created, you must add channels or lines to it. Channels are added to analog input and analog output objects, while lines are added to digital I/O objects. Channels and lines are the basic hardware device elements with which you acquire or output data.

3. **Configure properties** — To establish the device object behavior, you assign values to properties using the `set` function or dot notation.

   You can configure many of the properties at any time. However, some properties are configurable only when the device object is not running. Conversely, depending on your hardware settings and the requirements of your application, you might be able to accept the default property values and skip this step.

4. **Queue data** (analog output only) — Before you can output analog data, you must queue it in the engine with the `putdata` function.

5. **Start acquisition or output of data** — To acquire or output data, you must execute the device object with the `start` function. Acquisition and output occur in the background, while MATLAB continues executing. You can execute other MATLAB commands while the acquisition is occurring, and then wait for the acquisition or output to complete.
6 **Wait for the acquisition or output to complete** — You can continue working in MATLAB while the toolbox is acquiring or outputting data. (For more information, see Chapter 5, “Doing More with Analog Input”.) However, in many cases, you simply want to wait for the acquisition or output to complete before continuing. Use the `wait` function to pause MATLAB until the acquisition is complete.

7 **Extract your acquired data** (analog input only) — After data is acquired, you must extract it from the engine with the `getdata` function.

8 **Clean up** — When you no longer need the device object, you should remove it from memory using the `delete` function, and remove it from the MATLAB workspace using the `clear` command.

The data acquisition session is used in many of the documentation examples included in this guide. Note that the fourth step is treated differently for digital I/O objects because they do not store data in the engine. Therefore, only analog input and analog output objects are discussed in this section.

### Example: The Data Acquisition Session

This example illustrates the basic steps you take during a data acquisition session using an analog input object. You can run this example by typing `daqdoc3_1` at the MATLAB command line.

1 **Create a device object** — Create the analog input object `AI` for a sound card. The installed adaptors and hardware IDs are found with `daqhwinfo`.

   ```matlab
   AI = analoginput('winsound');
   ``%
   ```matlab
   AI = analoginput('nidaq','Dev1');
   ``%
   ```matlab
   AI = analoginput('mcc',1);
   ```

2 **Add channels** — Add two channels to `AI`.

   ```matlab
   addchannel(AI,1:2);
   ``%
   ```matlab
   addchannel(AI,0:1); % For NI and MCC
   ```

3 **Configure property values** — Configure the sampling rate to 11.025 kHz and define a 2 second acquisition.

   ```matlab
   set(AI,'SampleRate',11025)
   ```
set(AI,'SamplesPerTrigger',22050)

4 Start acquisition — Before the start function is issued, you might want to begin inputting data from a microphone or a CD player.

    start(AI)

5 Wait for the acquisition or output to complete — Pause MATLAB until either the acquisition completes or 3 seconds have elapsed (whichever comes first). If 3 seconds elapse, an error occurs.

    wait(AI,3);

6 Extract the acquired data from the engine and plot results

    data = getdata(AI);

    Plot the data and label the figure axes.

    plot(data)
    xlabel('Samples')
    ylabel('Signal (Volts)')

7 Clean up — When you no longer need AI, you should remove it from memory and from the MATLAB workspace.

    delete(AI)
    clear AI
Creating a Device Object

Understanding Device Objects

Device objects are the toolbox components you use to access your hardware device. They provide a gateway to the functionality of your hardware, and allow you to control the behavior of your data acquisition application. Each device object is associated with a specific hardware subsystem.

To create a device object, you call M-file functions called object creation functions (or object constructors). These M-files are implemented using the object-oriented programming capabilities provided by MATLAB, which are described in MATLAB Classes and Objects in the MATLAB Programming documentation. The device object creation functions are listed below.

Device Object Creation Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>analoginput</td>
<td>Create an analog input object.</td>
</tr>
<tr>
<td>analogoutput</td>
<td>Create an analog output object.</td>
</tr>
<tr>
<td>digitalio</td>
<td>Create a digital I/O object.</td>
</tr>
</tbody>
</table>

Before you can create a device object, the associated hardware driver adaptor must be registered. Adaptor registration occurs automatically. However, if for some reason an adaptor is not automatically registered, then you must do so manually with the daqregister function. Refer to “Registering the Hardware Driver Adaptor” on page A-20 for more information.

You can find out how to create device objects for a particular vendor and subsystem with the ObjectConstructorName field of the daqhwinfo function. For example, to find out how to create an analog input object for an installed
National Instruments board, you supply the appropriate adaptor name to `daqhwinfo`.

```matlab
out = daqhwinfo('nidaq');
out.ObjectConstructorName(:)
an =
    'analoginput('nidaq','Dev1')'
    'analogoutput('nidaq','Dev1')'
    'digitalio('nidaq','Dev1')'
```

The constructor syntax tells you that you must supply the adaptor name and the hardware ID to the `analoginput` function

```matlab
ai = analoginput('nidaq','Dev1');
```

The association between device objects and hardware subsystems is shown below.

---

**Creating an Array of Device Objects**

In MATLAB, you can create an array from existing variables by concatenating those variables together. The same is true for device objects. For example, suppose you create the analog input object `ai` and the analog output object `ao` for a sound card:

```matlab
ai = analoginput('winsound');
```
ao = analogoutput('winsound');

You can now create a device object array consisting of ai and ao using the usual MATLAB syntax. To create the row array x:

\[
x = [ai \ ao]\]

Index: Subsystem: Name:
1 Analog Input winsound0-AI
2 Analog Output winsound0-AO

To create the column array y:

\[
y = [ai; ao];
\]

Note that you cannot create a matrix of device objects. For example, you cannot create the matrix

\[
z = [ai \ ao; ai \ ao];
\]

??? Error using ==> analoginput/vertcat
Only a row or column vector of device objects can be created.

Depending on your application, you might want to pass an array of device objects to a function. For example, using one call to the set function, you can configure both ai and ao to the same property value.

\[
set(x,'SampleRate',44100)
\]

Refer to Chapter 12, “Functions — Alphabetical List” to see which functions accept a device object array as an input argument.

**Where Do Device Objects Exist?**

When you create a device object, it exists in both the MATLAB workspace and the data acquisition engine. For example, suppose you create the analog input object ai for a sound card and then make a copy of ai.

\[
ai = analoginput('winsound');
newai = ai;
\]
The copied device object `newai` is identical to the original device object `ai`. You can verify this by setting a property value for `ai` and returning the value of the same property from `newai`.

```matlab
set(ai,'SampleRate',22050);
get(newai,'SampleRate')
ans =
    22050
```

As shown below, `ai` and `newai` return the same property value because they both reference the same device object in the data acquisition engine.

If you delete either the original device object or a copy, then the engine device object is also deleted. In this case, you cannot use any copies of the device object that remain in the workspace because they are no longer associated with any hardware. Device objects that are no longer associated with hardware are called `invalid objects`. The example below illustrates this situation.

```matlab
delete(ai);
newai
newai =
    Invalid Data Acquisition object.
This object is not associated with any hardware and should be removed from your workspace using CLEAR.
```

You should remove invalid device objects from the workspace with the `clear` command.
Hardware Channels or Lines

In this section...

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Adding Channels and Lines” on page 3-9</td>
</tr>
<tr>
<td>“Mapping Hardware Channel IDs to MATLAB Indices” on page 3-10</td>
</tr>
</tbody>
</table>

Adding Channels and Lines

Channels and lines are the basic hardware device elements with which you acquire or output data.

After you create a device object, you must add channels or lines to it. Channels are added to analog input and analog output objects, while lines are added to digital I/O objects. The channels added to a device object constitute a channel group, while the lines added to a device object constitute a line group.

The functions associated with adding channels or lines to a device object are listed below.

Table 3-1  Functions Associated with Adding Channels or Lines

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addchannel</td>
<td>Add hardware channels to an analog input or analog output object.</td>
</tr>
<tr>
<td>addline</td>
<td>Add hardware lines to a digital I/O object.</td>
</tr>
<tr>
<td>addmuxchannel</td>
<td>Add channels when using a National Instruments AMUX-64T multiplexer. This applies only to Traditional NI-DAQ boards.</td>
</tr>
</tbody>
</table>

For example, to add two channels to an analog input object associated with a sound card, you must supply the appropriate hardware channel identifiers (IDs) to addchannel.

```matlab
ai = analoginput('winsound');
addchannel(ai,1:2)
```
Note You cannot acquire or output data with a device object that does not contain channels or lines. Similarly, you cannot acquire or output data with channels or lines that are not contained by a device object.

You can think of a device object as a channel or line container that reflects the common functionality of a particular device. The common functionality of a device applies to all channels or lines that it contains. For example, the sampling rate of an analog input object applies to all channels contained by that object. In contrast, the channels and lines contained by the device object reflect the functionality of a particular channel or line. For example, you can configure the input range (gain and polarity) on a per-channel basis.

The relationship between an analog input object and the channels it contains is shown below.

![Diagram of Analog Input Object]

For digital I/O objects, the diagram would look the same except that lines would be substituted for channels.

**Mapping Hardware Channel IDs to MATLAB Indices**

When you add channels to a device object, the resulting channel group consists of a mapping between hardware channel IDs and MATLAB indices.
Hardware channel IDs are numeric values defined by the hardware vendor that uniquely identify a channel. For National Instruments and Measurement Computing hardware, the channel IDs are “zero-based” (begin at zero). For Agilent Technologies hardware and sound cards, the channel IDs are “one-based” (begin at one). However, when you reference channels, you use the MATLAB indices and not the hardware IDs. Given this, you should keep in mind that MATLAB is one-based. You can return the vendor’s hardware IDs with the `daqhwinfo` function.

For example, suppose you create the analog input object `ai` for a National Instruments board and you want to add the first three differential channels.

```matlab
ai = analoginput('nidaq','Dev1');
```

To return the hardware IDs, supply the device object to `daqhwinfo`, and examine the `DifferentialIDs` field.

```matlab
out = daqhwinfo(ai);
out.DifferentialIDs
ans =
    0    1    2    3    4    5    6    7
```

The first three differential channels have IDs 0, 1, and 2, respectively.

```matlab
addchannel(ai,0:2);
```

The index assigned to a hardware channel depends on the order in which you add it to the device object. In the above example, the channels are automatically assigned the MATLAB indices 1, 2, and 3, respectively. You can change the hardware channels associated with the MATLAB indices using the `HwChannel` property. For example, to swap the order of the second and third hardware channels,

```matlab
ai.Channel(2).HwChannel = 2;
ai.Channel(3).HwChannel = 1;
```
The original and modified index assignments are shown below.

<table>
<thead>
<tr>
<th>Hardware channel ID</th>
<th>MATLAB index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

**Note** If you are using scanning hardware, then the MATLAB indices define the scan order; index 1 is sampled first, index 2 is sampled second, and so on.

For digital I/O objects, the diagram would look the same except that lines would be substituted for channels.
Configuring and Returning Properties

In this section...

| “Overview” on page 3-13 |
| “Property Types” on page 3-13 |
| “Returning Property Names and Property Values” on page 3-15 |
| “Configuring Property Values” on page 3-19 |
| “Specifying Property Names” on page 3-21 |
| “Default Property Values” on page 3-21 |
| “The Property Inspector” on page 3-22 |

Overview

You define and evaluate the behavior of your data acquisition application with device object properties. You define your application behavior by assigning values to properties with the `set` function or the dot notation. You evaluate your application configuration and status by displaying property values with the `get` function or the dot notation.

Property Types

Data Acquisition Toolbox properties are divided into two main types:

- **Common properties** — Common properties apply to every channel or line contained by a device object.

- **Channel/Line properties** — Channel/line properties are configured for individual channels or lines.

The relationship between an analog input object, the channels it contains, and their properties is shown below.
For digital I/O objects, the diagram would look the same except that lines would be substituted for channels.

Common properties and channel/line properties are subdivided into these two categories:

- **Base properties** — Base properties apply to all supported hardware subsystems of a given type, such as analog input. For example, the SampleRate property is supported for all analog input subsystems regardless of the vendor.

- **Device-specific properties** — Device-specific properties apply only to specific hardware devices. For example, the BitsPerSample property is supported only for sound cards. Note that base properties can have device-specific values. For example, the InputType property has a different set of values for each supported hardware vendor.

The relationship between common properties, channel/line properties, base properties, and device-specific properties is shown below.
For a complete description of all properties, refer to Chapter 14, “Base Properties — Alphabetical List” or Chapter 16, “Device-Specific Properties — Alphabetical List”.

**Returning Property Names and Property Values**

Once the device object is created, you can use the `set` function to return all configurable properties to a variable or to the command line. Additionally, if a property has a finite set of string values, then `set` also returns these values. You can use the `get` function to return one or more properties and their current values to a variable or to the command line.

The syntax used to return common and channel/line properties is described below. The examples are based on the analog input object `ai` created for a sound card and containing two channels.

```matlab
ai = analoginput('winsound');
addchannel(ai,1:2);
```
Common Properties

To return all configurable common property names and their possible values for a device object, you must supply the device object to set. For example, all configurable common properties for ai are shown below. The base properties are listed first, followed by the device-specific properties.

```
set(ai)
BufferingConfig
BufferingMode: [ {Auto} | Manual ]
Channel
ChannelSkew
ChannelSkewMode: [ {None} ]
ClockSource: [ {Internal} ]
DataMissedFcn
InputOverRangeFcn
InputType: [ {AC-Coupled} ]
LogFileName
LoggingMode: [ Disk | {Memory} | Disk&Memory ]
LogToDiskMode: [ {Overwrite} | Index ]
ManualTriggerHwOn: [ {Start} | Trigger ]
Name
RuntimeErrorFcn
SampleRate
SamplesAcquiredFcn
SamplesAcquiredFcnCount
SamplesPerTrigger
StartFcn
StopFcn
Tag
Timeout
TimerFcn
TimerPeriod
TriggerFcn
TriggerChannel
TriggerCondition: [ {None} ]
TriggerConditionValue
TriggerDelay
TriggerDelayUnits: [ {Seconds} | Samples ]
TriggerRepeat
TriggerType: [ Manual | {Immediate} | Software ]
```
UserData

WINCSOUND specific properties:
BitsPerSample
StandardSampleRates: [ Off | {On} ]

To return all common properties and their current values for a device object, you must supply the device object to get. For example, all common properties for ai are shown below. The base properties are listed first, followed by the device-specific properties.

get(ai)
BufferingConfig = [512 30]
BufferingMode = Auto
Channel = [2x1 aichannel]
ChannelSkew = 0
ChannelSkewMode = None
ClockSource = Internal
DataMissedFcn = @daqcallback
EventLog = []
InitialTriggerTime = [0 0 0 0 0 0]
InputOverRangeFcn =
InputType = AC-Coupled
LogFileName = logfile.daq
Logging = Off
LoggingMode = Memory
LogToDiskMode = Overwrite
ManualTriggerHwOn = Start
Name = winsound0-AI
Running = Off
RuntimeErrorFcn = @daqcallback
SampleRate = 8000
SamplesAcquired = 0
SamplesAcquiredFcn =
SamplesAcquiredFcnCount = 1024
SamplesAvailable = 0
SamplesPerTrigger = 8000
StartFcn =
StopFcn =
Tag =
Timeout = 1
TimerFcn =
TimerPeriod = 0.1
TriggerFcn =
TriggerChannel = [1x0 aichannel]
TriggerCondition = None
TriggerConditionValue = 0
TriggerDelay = 0
TriggerDelayUnits = Seconds
TriggerRepeat = 0
TriggersExecuted = 0
TriggerType = Immediate
Type = Analog Input
UserData = []

WINSOUND specific properties:
BitsPerSample = 16
StandardSampleRates = On

To display the current value for one property, you supply the property name to get.

get(ai,'SampleRate')
ans =
     8000

To display the current values for multiple properties, you include the property names as elements of a cell array.

get(ai,{'StandardSampleRates','Running'})
ans =
     'On'    'Off'

You can also use the dot notation to display a single property value.

ai.TriggerType
ans =
   Immediate
Channel and Line Properties
To return all configurable channel (line) property names and their possible values for a single channel (line) contained by a device object, you must use the Channel (Line) property. For example, to display the configurable channel properties for the first channel contained by ai,

```matlab
set(ai.Channel(1))
ChannelName
HwChannel
InputRange
SensorRange
Units
UnitsRange
```

All channel properties and their current values for the first channel contained by ai are shown below.

```matlab
get(ai.Channel(1))
ChannelName = Left
HwChannel = 1
Index = 1
InputRange = [-1 1]
NativeOffset = 1.5259e-005
NativeScaling = 3.0518e-005
Parent = [1x1 analoginput]
SensorRange = [-1 1]
Type = Channel
Units = Volts
UnitsRange = [-1 1]
```

As described in the preceding section, you can also return values for a specified number of channel properties with the `get` function or the dot notation.

Configuring Property Values
You configure property values with the `set` function or the dot notation. In practice, you can configure many of the properties at any time while the device object exists. However, some properties are not configurable while the object is running. Use the `propinfo` function, or refer to Chapter 14, “Base Properties — Alphabetical List” for information about when a property is configurable.
The syntax used to configure common and channel/line properties is described below. The examples are based on the analog input object `ai` created in “Returning Property Names and Property Values” on page 3-15.

**Common Properties**
You can configure a single property value using the `set` function

```
set(ai,'TriggerType','Manual')
```

or the dot notation

```
ai.TriggerType = 'Manual';
```

To configure values for multiple properties, you can supply multiple property name/property value pairs to `set`.

```
set(ai,'SampleRate',44100,'Name','Test1-winsound')
```

Note that you can configure only one property value at a time using the dot notation.

**Channel and Line Properties**
To configure channel (line) properties for one or more channels (lines) contained by a device object, you must use the `Channel` (Line) property.

For example, to configure the `SensorRange` property for the first channel contained by `ai`, you can use the `set` function

```
set(ai.Channel(1),'SensorRange',[-2 2])
```

or the dot notation

```
ai.Channel(1).SensorRange = [-2 2];
```

To configure values for multiple channel or line properties, you supply multiple property name/property value pairs to `set`.

```
set(ai.Channel(1),'SensorRange',[-2 2],'ChannelName','Chan1')
```

To configure multiple property values for multiple channels:

```
chs = ai.Channel(1:2);
```
set(chs,{'SensorRange','ChannelName'},{[-2 2],'Chan1';[0 4], 'Chan2'});

**Specifying Property Names**

Device object property names are presented in this guide using mixed case. While this makes the names easier to read, you can use any case you want when specifying property names. Additionally, you need use only enough letters to identify the property name uniquely, so you can abbreviate most property names. For example, you can configure the `SampleRate` property any of these ways.

```matlab
set(ai,'SampleRate',44100);
set(ai,'samplerate',44100);
set(ai,'sampler',44100);
```

However, when you include property names in an M-file, you should use the full property name. This practice can prevent problems with future releases of Data Acquisition Toolbox if a shortened name is no longer unique because of the addition of new properties.

**Default Property Values**

If you do not explicitly define a value for a property, then the default value is used. All configurable properties have default values. However, the default value for a given property might vary based on the hardware you are using. Additionally, some default values are calculated by the engine and depend on the values set for other properties. If the hardware driver adaptor specifies a default value for a property, then that value takes precedence over the value defined by the toolbox.

If a property has a finite set of string values, then the default value is enclosed by {} (curly braces). For example, the default value for the `LoggingMode` property is `Memory`.

```matlab
set(ai,'LoggingMode')
[ Disk | {Memory} | Disk&Memory ]
```

You can also use the `propinfo` function, or refer to Chapter 14, “Base Properties — Alphabetical List” or Chapter 16, “Device-Specific Properties — Alphabetical List” to find the default value for any property.
The Property Inspector

The Property Inspector is a graphical user interface (GUI) for accessing toolbox object properties. The Property Inspector is designed so you can

- Display the names and current values for object properties
- Display possible values for enumerated properties
- Configure the property values

You open the Property Inspector with the `inspect` function, or via the Workspace browser by double-clicking an object.

For example, create the analog input object `ai` for a sound card and add both hardware channels.

```matlab
ai = analoginput('winsound');
addchannel(ai,1:2);
```

Open the Property Inspector from the command line.

```matlab
inspect(ai)
```

For more information on the Property Inspector, see the `inspect` reference page.
Acquiring and Outputting Data

In this section...

“Device Object States” on page 3-23
“Starting the Device Object” on page 3-24
“Logging or Sending Data” on page 3-24
“Stopping the Device Object” on page 3-25

Device Object States

As data is being transferred between MATLAB and your hardware, you can think of the device object as being in a particular state. Two types of states are defined for Data Acquisition Toolbox:

- **Running** — For analog input objects, *running* means that data is being acquired from an analog input subsystem. However, the acquired data is not necessarily saved to memory or a disk file. For analog output objects, running means that data queued in the engine is ready to be output to an analog output subsystem.

  The running state is indicated by the Running property for both analog input and analog output objects. Running can be On or Off.

- **Logging or Sending** — For analog input objects, *logging* means that data acquired from an analog input subsystem is being stored in the engine or saved to a disk file. The logging state is indicated by the Logging property. Logging can be On or Off.

  For analog output objects, *sending* means the data queued in the engine is being output to an analog output subsystem. The sending state is indicated by the Sending property. Sending can be On or Off.

Running, Logging, and Sending are read-only properties that are automatically set to On or Off by the engine. When Running is Off, Logging and Sending must be Off. When Running is On, Logging and Sending are set to On only when a trigger occurs.
**Note** Digital I/O objects also possess a running state. However, because they do not store data in the engine, the logging and sending states do not exist.

### Starting the Device Object

You start a device object with the `start` function. For example, to start the analog input object `ai`,

```matlab
ai = analoginput('winsound')
addchannel(ai,1:2)
start(ai)
```

After `start` is issued, the `Running` property is automatically set to On, and both the device object and hardware device execute according to the configured and default property values.

While you are acquiring data with an analog input object, you can preview the data with the `peekdata` function. `peekdata` takes a snapshot of the most recent data but does not remove data from the engine. For example, to preview the most recent 500 samples acquired by each channel contained by `ai`,

```matlab
data = peekdata(ai,500);
```

Because previewing data is usually a low-priority task, `peekdata` does not guarantee that all requested data is returned. You can preview data at any time while the device object is running.

### Logging or Sending Data

While the device object is running, you can

- Log data acquired from an analog input subsystem to the engine (memory) or to a disk file.
- Output data queued in the engine to an analog output subsystem.

However, before you can log or send data, a trigger must occur. You configure an analog input or analog output trigger with the `TriggerType` property. All the examples presented in this section use the default `TriggerType` value of `Immediate`, which executes the trigger immediately after the `start` function.
is issued. For a detailed description of triggers, refer to “Configuring Analog Input Triggers” on page 5-19 or “Configuring Analog Output Triggers” on page 6-20.

**Extracting Logged Data**

When a trigger occurs for an analog input object, the **Logging** property is automatically set to **On** and data acquired from the hardware is logged to the engine or a disk file. You extract logged data from the engine with the `getdata` function. For example, to extract 500 samples for each channel contained by `ai`,

```matlab
data = getdata(ai,500);
```

getdata blocks the MATLAB command line until all the requested data is returned to the workspace. You can extract data any time after the trigger occurs.

**Sending Queued Data**

For analog output objects, you must queue data in the engine with the `putdata` function before it can be output to the hardware. For example, to queue 8000 samples in the engine for each channel contained by the analog output object `ao`

```matlab
    ao = analogoutput('winsound');
    addchannel(ao,1:2);
    data = sin(linspace(0,2*pi*500,8000))';
    putdata(ao,[data data])
```

Before the queued data can be output, you must start the analog output object.

```matlab
    start(ao)
```

When a trigger occurs, the **Sending** property is automatically set to **On** and the queued data is sent to the hardware.

**Stopping the Device Object**

An analog input (AI) or analog output (AO) object can stop under one of these conditions:
• You issue the stop function.
• The requested number of samples is acquired (AI) or sent (AO).
• A run-time hardware error occurs.
• A timeout occurs.

When the device object stops, the Running, Logging, and Sending properties are automatically set to Off. At this point, you can reconfigure the device object or immediately issue another start command using the current configuration.
Cleaning Up

When you no longer need a device object, you should clean up the MATLAB environment by removing the object from memory (the engine) and from the workspace. These are the steps you take to end a data acquisition session.

You remove device objects from memory with the `delete` function. For example, to delete the analog input object `ai` created in the preceding section:

```matlab
delete(ai)
```

A deleted device object is invalid, which means that you cannot connect it to the hardware. In this case, you should remove the object from the MATLAB workspace. To remove device objects and other variables from the MATLAB workspace, use the `clear` command.

```matlab
clear ai
```

If you use `clear` on a device object that is connected to hardware, the object is removed from the workspace but remains connected to the hardware. You can restore cleared device objects to MATLAB with the `daqfind` function.
Data Acquisition Session
Getting Started with Analog Input

Analog input (AI) subsystems convert real-world analog signals from a sensor into bits that can be read by your computer. AI subsystems are typically multichannel devices offering 12 or 16 bits of resolution. Data Acquisition Toolbox provides access to analog input devices through an analog input object.

The purpose of this chapter is to show you how to perform simple analog input tasks using just a few functions and properties. After reading this chapter, you should be able to use the toolbox to configure your own analog input session. The sections are as follows.

Creating an Analog Input Object (p. 4-3)
Create a MATLAB object that represents the analog input subsystem

Adding Channels to an Analog Input Object (p. 4-5)
Associate hardware channels with the analog input object

Configuring Analog Input Properties (p. 4-10)
Define the object behavior by assigning values to properties

Acquiring Data (p. 4-15)
Execute the object and stream data from the hardware channels to memory
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analog Input Examples (p. 4-17)</td>
<td>Examples that show you how to perform a complete data acquisition task</td>
</tr>
<tr>
<td>Evaluating the Analog Input Object Status (p. 4-25)</td>
<td>Return the values of certain properties in a convenient display format</td>
</tr>
</tbody>
</table>
Creating an Analog Input Object

You create an analog input object with the analoginput function. analoginput accepts the adaptor name and the hardware device ID as input arguments. For a list of supported adaptors, refer to . The device ID refers to the number associated with your board when it is installed. (When using NI-DAQmx, this is usually a string such as 'Dev1'.) Some vendors refer to the device ID as the device number or the board number. The device ID is optional for sound cards with an ID of 0. Use the daqhwinfo function to determine the available adaptors and device IDs.

Each analog input object is associated with one board and one analog input subsystem. For example, to create an analog input object associated with a National Instruments board with device ID 1:

    ai = analoginput('nidaq','Dev1');

The analog input object ai now exists in the MATLAB workspace. You can display the class of ai with the who command.

    whos ai
    Name      Size           Bytes  Class          Attributes
    ai        1x1            1332  analoginput object

Grand total is 52 elements using 1332 bytes

Once the analog input object is created, the properties listed below are automatically assigned values. These general purpose properties provide descriptive information about the object based on its class type and adaptor.

Table 4-1 Descriptive Analog Input Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Specify a descriptive name for the device object.</td>
</tr>
<tr>
<td>Type</td>
<td>Indicate the device object type.</td>
</tr>
</tbody>
</table>

You can display the values of these properties for ai with the get function.
get(ai,{'Name','Type'})
ans =
    'nidaq1-AI'    'Analog Input'
Adding Channels to an Analog Input Object

In this section...

“Channel Group” on page 4-5
“Referencing Individual Hardware Channels” on page 4-7
“Example: Adding Channels for a Sound Card” on page 4-8

Channel Group

After creating the analog input object, you must add hardware channels to it. As shown by the figure in “Hardware Channels or Lines” on page 3-9, you can think of a device object as a container for channels. The collection of channels contained by the device object is referred to as a channel group. As described in “Mapping Hardware Channel IDs to MATLAB Indices” on page 3-10, a channel group consists of a mapping between hardware channel IDs and MATLAB indices (see below).

When adding channels to an analog input object, you must follow these rules:

- The channels must reside on the same hardware device. You cannot add channels from different devices, or from different subsystems on the same device.
- The channels must be sampled at the same rate.

You add channels to an analog input object with the `addchannel` function. `addchannel` requires the device object and at least one hardware channel ID as input arguments. You can optionally specify MATLAB indices, descriptive channel names, and an output argument. For example, to add two hardware channels to the device object `ai` created in the preceding section:

```matlab
chans = addchannel(ai,0:1);
```

The output argument `chans` is a channel object that reflects the channel array contained by `ai`. You can display the class of `chans` with the `whos` command.

```matlab
whos chans
Name      Size     Bytes  Class
```
Getting Started with Analog Input

chans 2x1 512 aichannel object

Grand total is 7 elements using 512 bytes

You can use `chans` to easily access channels. For example, you can easily configure or return property values for one or more channels. As described in “Referencing Individual Hardware Channels” on page 4-7, you can also access channels with the `Channel` property.

Once you add channels to an analog input object, the properties listed below are automatically assigned values. These properties provide descriptive information about the channels based on their class type and ID.

**Table 4-2 Descriptive Analog Input Channel Properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HwChannel</td>
<td>Specify the hardware channel ID.</td>
</tr>
<tr>
<td>Index</td>
<td>Indicate the MATLAB index of a hardware channel.</td>
</tr>
<tr>
<td>Parent</td>
<td>Indicate the parent (device object) of a channel.</td>
</tr>
<tr>
<td>Type</td>
<td>Indicate a channel.</td>
</tr>
</tbody>
</table>

You can display the values of these properties for `chans` with the `get` function.

```matlab
get(chans,{'HwChannel','Index','Parent','Type'})
ans =
[0]    [1]    [1x1 analoginput]    'Channel'
[1]    [2]    [1x1 analoginput]    'Channel'
```

If you are using scanning hardware, then the MATLAB indices define the scan order; index 1 is sampled first, index 2 is sampled second, and so on.
The number of channels you can add to a device object depends on the specific board you are using. Some boards support adding channels in any order and adding the same channel multiple times, while other boards do not. Additionally, each channel might have its own input range, which is verified with each acquired sample. The collection of channels you add to a device object is sometimes referred to as a channel gain list or a channel gain queue. For scanning hardware, these channels define the scan order.

Referencing Individual Hardware Channels
As described in the preceding section, you can access channels with the Channel property or with a channel object. To reference individual channels, you must specify either MATLAB indices or descriptive channel names.

MATLAB Indices
Every hardware channel contained by an analog input object has an associated MATLAB index that is used to reference the channel. When adding channels with the addchannel function, index assignments can be made automatically or manually. In either case, the channel indices start at 1 and increase monotonically up to the number of channel group members.

For example, the analog input object ai created in the preceding section had the MATLAB indices 1 and 2 automatically assigned to the hardware channels 0 and 1, respectively. To manually swap the hardware channel order, you supply the appropriate index to chans and use the HwChannel property.

```matlab
chans(1).HwChannel = 1;
chans(2).HwChannel = 0;
```

Alternatively, you can use the Channel property.

```matlab
ai.Channel(1).HwChannel = 1;
ai.Channel(2).HwChannel = 0;
```

Note that you can also use addchannel to specify the required channel order.

```matlab
chans = addchannel(ai,[1 0]);
```
Descriptive Channel Names
Choosing a unique, descriptive name can be a useful way to identify and reference channels — particularly for large channel groups. You can associate descriptive names with hardware channels using the `addchannel` function. For example, suppose you want to add 16 single-ended channels to `ai`, and you want to associate the name `TrigChan` with the first channel in the group.

```matlab
ai.InputType = 'SingleEnded';
addchannel(ai,0,'TrigChan');
addchannel(ai,1:15);
```

Alternatively, you can use the `ChannelName` property.

```matlab
ai.InputType = 'SingleEnded';
addchannel(ai,0:15);
ai.Channel(1).ChannelName = 'TrigChan';
```

You can now use the channel name to reference the channel.

```matlab
ai.TrigChan.InputRange = [-10 10];
```

Example: Adding Channels for a Sound Card
Suppose you create the analog input object `ai` for a sound card.

```matlab
ai = analoginput('winsound');
```

Most sound cards have just two hardware channels that you can add. If one channel is added, the sound card is said to be in mono mode. If two channels are added, the sound card is said to be in stereo mode. However, the rules for adding these two channels differ from those of other data acquisition devices. These rules are described below.

Mono Mode
If you add one channel to `ai`, the sound card is said to be in mono mode and the channel added must have a hardware ID of 1.

```matlab
addchannel(ai,1);
```

At the software level, mono mode means that data is acquired from channel 1. At the hardware level, you generally cannot determine the actual channel configuration and data can be acquired from channel 1, channel 2, or both.
Adding Channels to an Analog Input Object

depending on your sound card. Channel 1 is automatically assigned the descriptive channel name Mono.

```matlab
ai.Channel.ChannelName
ans =
Mono
```

**Stereo Mode**

If you add two channels to `ai`, the sound card is said to be in stereo mode. You can add two channels using two calls to `addchannel` provided channel 1 is added first.

```matlab
addchannel(ai,1);
addchannel(ai,2);
```

Alternatively, you can use one call to `addchannel` provided channel 1 is specified as the first element of the hardware ID vector.

```matlab
addchannel(ai,1:2);
```

Stereo mode means that data is acquired from both hardware channels. Channel 1 is automatically assigned the descriptive name `Left` and channel 2 is automatically assigned the descriptive name `Right`.

```matlab
ai.Channel.ChannelName
ans =
'sLeft'
'Right'
```

While in stereo mode, if you want to delete one channel, then that channel must be channel 2. If you try to delete channel 1, an error is returned.

```matlab
delete(ai.Channel(2))
```

The sound card is now in mono mode.
Configuring Analog Input Properties

In this section...

| “Analog Input: Basic Properties” on page 4-10 |
| “The Sampling Rate” on page 4-11 |
| “Trigger Types” on page 4-12 |
| “The Samples to Acquire per Trigger” on page 4-13 |

Analog Input: Basic Properties

After hardware channels are added to the analog input object, you should configure property values. As described in “Configuring and Returning Properties” on page 3-13, Data Acquisition Toolbox supports two basic types of properties for analog input objects: common properties and channel properties. Common properties apply to all channels contained by the device object while channel properties apply to individual channels.

The properties you configure depend on your particular analog input application. For many common applications, there is a small group of properties related to the basic setup that you will typically use. These basic setup properties control the sampling rate, define the trigger type, and define the samples to be acquired per trigger. Analog input properties related to the basic setup are given below.

Table 4-3 Analog Input Basic Setup Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SampleRate</td>
<td>Specify the per-channel rate at which analog data is converted to digital data.</td>
</tr>
</tbody>
</table>
Table 4-3  Analog Input Basic Setup Properties (Continued)

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SamplesPerTrigger</td>
<td>Specify the number of samples to acquire for each channel group member for each trigger that occurs.</td>
</tr>
<tr>
<td>TriggerType</td>
<td>Specify the type of trigger to execute.</td>
</tr>
</tbody>
</table>

**The Sampling Rate**

You control the rate at which an analog input subsystem converts analog data to digital data with the SampleRate property. SampleRate must be specified as samples per second. For example, to set the sampling rate for each channel of your National Instruments board to 100,000 samples per second (100 kHz)

```matlab
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:1);
set(ai,'SampleRate',100000)
```

Data acquisition boards typically have predefined sampling rates that you can set. If you specify a sampling rate that does not match one of these predefined values, there are two possibilities:

- If the rate is within the range of valid values, then the engine automatically selects a valid sampling rate. The rules governing this selection process are described in the SampleRate reference pages in Chapter 14, “Base Properties — Alphabetical List”.

- If the rate is outside the range of valid values, then an error is returned.

**Note** For some sound cards, you can set the sampling rate to any value between the minimum and maximum values defined by the hardware. You can enable this feature with the StandardSampleRates property. Refer to Chapter 16, “Device-Specific Properties — Alphabetical List” for more information.

For hardware that supports simultaneous sampling of channels (sound cards and Agilent Technologies devices), the maximum sampling rate for each
channel is given by the maximum board rate. For scanning hardware (most National Instruments and Measurement Computing devices), the per-channel sampling rate is given by the maximum hardware rate divided by the number of channels contained by the device object.

After setting a value for `SampleRate`, you should find out the actual rate set by the engine.

```matlab
ActualRate = get(ai,'SampleRate');
```

Alternatively, you can use the `setverify` function, which sets a property value and returns the actual value set.

```matlab
ActualRate = setverify(ai,'SampleRate',100000);
```

You can find the range of valid sampling rates for your hardware with the `propinfo` function.

```matlab
ValidRates = propinfo(ai,'SampleRate');
ValidRates.ConstraintValue
ans =
    1.0e+005 *
     0.0000  2.0000
```

**Trigger Types**

For analog input objects, a trigger is defined as an event that initiates data logging to memory or to a disk file. Defining an analog input trigger involves specifying the trigger type with the `TriggerType` property. The `TriggerType` values that are supported for all hardware are given below.

**Table 4-4  Analog Input TriggerType Property Values**

<table>
<thead>
<tr>
<th>TriggerType Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Immediate}</td>
<td>The trigger occurs just after the start function is issued.</td>
</tr>
</tbody>
</table>
Table 4-4  Analog Input TriggerType Property Values (Continued)

<table>
<thead>
<tr>
<th>TriggerType Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual</td>
<td>The trigger occurs just after you manually issue the trigger function.</td>
</tr>
<tr>
<td>Software</td>
<td>The trigger occurs when the associated trigger condition is satisfied.</td>
</tr>
<tr>
<td></td>
<td>Trigger conditions are given by the TriggerCondition property.</td>
</tr>
</tbody>
</table>

Many devices have additional hardware trigger types, which are available to you through the `TriggerType` property. For example, to return all the trigger types for the analog input object `ai` created in the preceding section:

```matlab
set(ai,'TriggerType')
[ Manual | {Immediate} | Software | HwDigital ]
```

This information tells you that the National Instruments board also supports a hardware digital trigger. For a description of device-specific trigger types, refer to “Device-Specific Hardware Triggers” on page 5-37, or the TriggerType reference pages in Chapter 14, “Base Properties — Alphabetical List”.

**Note** Triggering can be a complicated issue and it has many associated properties. For detailed information about triggering, refer to “Configuring Analog Input Triggers” on page 5-19.

**The Samples to Acquire per Trigger**

When a trigger executes, a predefined number of samples are acquired for each channel group member and logged to the engine or a disk file. You specify the number of samples to acquire per trigger with the `SamplesPerTrigger` property.

The default value of `SamplesPerTrigger` is calculated by the engine such that 1 second of data is collected, and is based on the default value of `SampleRate`. In general, to calculate the acquisition time for each trigger, you apply the formula
acquisition time (seconds) = samples per trigger / sampling rate (in Hz)

For example, to acquire 5 seconds of data per trigger for each channel contained by ai:

```matlab
set(ai,'SamplesPerTrigger',500000)
```

To continually acquire data, you set `SamplesPerTrigger` to `inf`.

```matlab
set(ai,'SamplesPerTrigger',inf)
```

A continuous acquisition stops only if you issue the `stop` function, or an error occurs.
In this section...

“Starting the Analog Input Object” on page 4-15
“Logging Data” on page 4-15
“Stopping the Analog Input Object” on page 4-16

Starting the Analog Input Object

You start an analog input object with the `start` function. For example, to start the analog input object `ai`:

```matlab
ai = analoginput('winsound')
addchannel(ai,1:2)
start(ai)
```

After `start` is issued, the `Running` property is automatically set to `On`, and both the device object and hardware device execute according to the configured and default property values.

While you are acquiring data with an analog input object, you can preview the data with the `peekdata` function. `peekdata` takes a "snapshot" of the most recent data but does not remove data from the engine. For example, to preview the most recent 500 samples acquired by each channel contained by `ai`:

```matlab
data = peekdata(ai,500);
```

Because previewing data is usually a low-priority task, `peekdata` does not guarantee that all requested data is returned. You can preview data at any time while the device object is running. However, you cannot use `peekdata` in conjunction with hardware triggers because the device is idle until the hardware trigger is received.

Logging Data

While the analog input object is running, you can log acquired data to the engine (memory) or to a disk file. However, before you can log data a trigger must occur. You configure an analog input trigger with the `TriggerType`
property. For a detailed description of triggers, see “Configuring Analog Input Triggers” on page 5-19.

When the trigger occurs, the Logging property is automatically set to On and data acquired from the hardware is logged to the engine or a disk file. You extract logged data from the engine with the getdata function. For example, to extract all logged samples for each channel contained by ai:

```matlab
data = getdata(ai);
```

calldata blocks the MATLAB command line until all the requested data is returned to the workspace. You can extract data any time after the trigger occurs. You can also return sample-time pairs with getdata. For example, to extract 500 sample-time pairs for each channel contained by ai:

```matlab
[data,time] = getdata(ai,500);
```

time is an m-by-1 array containing relative time values for all m samples. Time is measured relative to the time the first sample is logged, and is measured continuously until the acquisition stops. You can read more detail in the getdata reference page.

You can log data to disk with the LoggingMode property. You can replay data saved to disk with the daqread function. Refer to “Logging Information to Disk” on page 8-5 for more information about LoggingMode and daqread.

**Stopping the Analog Input Object**

An analog input object can stop under one of these conditions:

- You issue the stop function.
- The requested number of samples is acquired.
- A run-time hardware error occurs.
- A timeout occurs.

When the device object stops, the Running and Logging properties are automatically set to Off. At this point, you can reconfigure the device object or immediately issue another start command using the current configuration.
Basic Steps for Acquiring Data
This section illustrates how to perform basic data acquisition tasks using analog input subsystems and Data Acquisition Toolbox. For most data acquisition applications, you must follow these basic steps:

1 Install and connect the components of your data acquisition hardware.
   At a minimum, this involves connecting a sensor to a plug-in or external data acquisition device.

2 Configure your data acquisition session. This involves creating a device object, adding channels, setting property values, and using specific functions to acquire data.

3 Analyze the acquired data using MATLAB.

Simple data acquisition applications using a sound card and a National Instruments board are given below.

Acquiring Data with a Sound Card
Suppose you must verify that the fundamental (lowest) frequency of a tuning fork is 440 Hz. To perform this task, you will use a microphone and a sound card to collect sound level data. You will then perform a fast Fourier transform (FFT) on the acquired data to find the frequency components of the tuning fork. The setup for this task is shown below.
Getting Started with Analog Input

Configuring the Data Acquisition Session

For this example, you will acquire 1 second of sound level data on one sound card channel. Because the tuning fork vibrates at a nominal frequency of 440 Hz, you can configure the sound card to its lowest sampling rate of 8000 Hz. Even at this lowest rate, you should not experience any aliasing effects because the tuning fork will not have significant spectral content above 4000 Hz, which is the Nyquist frequency. After you set the tuning fork vibrating and place it near the microphone, you will trigger the acquisition one time using a manual trigger.

You can run this example by typing daqdoc4_1 at the MATLAB command line.

1 Create a device object — Create the analog input object AI for a sound card. The installed adaptors and hardware IDs are found with daqhwinfo.

   AI = analoginput('winsound');

2 Add channels — Add one channel to AI.

   chan = addchannel(AI,1);

3 Configure property values — Assign values to the basic setup properties, and create the variables blocksize and Fs, which are used for subsequent analysis. The actual sampling rate is retrieved because it might be set by the engine to a value that differs from the specified value.

   duration = 1; %1 second acquisition
   set(AI,'SampleRate',8000)
   ActualRate = get(AI,'SampleRate');
Analog Input Examples

set(AI,'SamplesPerTrigger',duration*ActualRate)
set(AI,'TriggerType','Manual')
blocksize = get(AI,'SamplesPerTrigger');
Fs = ActualRate;

4 Acquire data — Start AI, issue a manual trigger, and extract all data from
the engine. Before trigger is issued, you should begin inputting data from
the tuning fork to the sound card.

    start(AI)
    trigger(AI)
    wait(AI,duration + 1)

The `wait` function pauses MATLAB until either the acquisition completes
or the timeout elapses (whichever comes first). If the timeout elapses, an
error occurs. Adding 1 second to the duration allows some margin for the
timeout.

    data = getdata(AI);

5 Clean up — When you no longer need AI, you should remove it from
memory and from the MATLAB workspace.

    delete(AI)
    clear AI

Analyzing the Data

For this example, analysis consists of finding the frequency components of the
tuning fork and plotting the results. To do so, the function `daqdocfft` was
created. This function calculates the FFT of data, and requires the values of
SampleRate and SamplesPerTrigger as well as data as inputs.

    [f,mag] = daqdocfft(data,Fs,blocksize);

daqdocfft outputs the frequency and magnitude of data, which you can then
plot. `daqdocfft` is shown below.

    function [f,mag] = daqdocfft(data,Fs,blocksize)
    % [F,MAG]=DAQDOCFFT(X,FS,BLOCKSIZE) calculates the FFT of X
    % using sampling frequency FS and the SamplesPerTrigger
    % provided in BLOCKSIZE
xfft = abs(fft(data));

% Avoid taking the log of 0.
index = find(xfft == 0);
xfft(index) = 1e-17;

mag = 20*log10(xfft);
mag = mag(1:floor(blocksize/2));
f = (0:length(mag)-1)*Fs/blocksize;
f = f(:);

The results are given below.

plot(f,mag)
grid on
ylabel('Magnitude (dB)')
xlabel('Frequency (Hz)')
title('Frequency Components of Tuning Fork')
The plot shows the fundamental frequency around 440 Hz and the first overtone around 880 Hz. A simple way to find actual fundamental frequency is

\[
[\text{ymax, maxindex}] = \text{max}(\text{mag}); \\
\text{maxfreq} = f(\text{maxindex}) \\
\text{maxfreq} = 441
\]

The answer is 441 Hz.

**Note** The fundamental frequency is not always the frequency component with the largest amplitude. A more sophisticated approach involves fitting the observed frequencies to a harmonic series to find the fundamental frequency.

**Acquiring Data with a National Instruments Board**

Suppose you must verify that the nominal frequency of a sine wave generated by a function generator is 1.00 kHz. To perform this task, you will input the function generator signal into a National Instruments board. You will then perform a fast Fourier transform (FFT) on the acquired data to find the nominal frequency of the generated sine wave. The setup for this task is shown below.

![Diagram](image)

**Configuring the Data Acquisition Session**

For this example, you will acquire 1 second of data on one input channel. The board is set to a sampling rate of 10 kHz, which is well above the frequency
of interest. After you connect the input signal to the board, you will trigger the acquisition one time using a manual trigger.

You can run this example by typing `daqdoc4_2` at the MATLAB command line.

1 **Create a device object** — Create the analog input object `AI` for a National Instruments board. The installed adaptors and hardware IDs are found with `daqhwinfo`.

   ```matlab
   AI = analoginput('nidaq','Dev1');
   ```

2 **Add channels** — Add one channel to `AI`.

   ```matlab
   chan = addchannel(AI,0);
   ```

3 **Configure property values** — Assign values to the basic setup properties, and create the variables `blocksize` and `Fs`, which are used for subsequent analysis. The actual sampling rate is retrieved because it might be set by the engine to a value that differs from the specified value.

   ```matlab
   duration = 1; % 1 second acquisition
   set(AI,'SampleRate',10000)
   ActualRate = get(AI,'SampleRate');
   set(AI,'SamplesPerTrigger',duration*ActualRate)
   set(AI,'TriggerType','Manual')
   blocksize = get(AI,'SamplesPerTrigger');
   Fs = ActualRate;
   ```

4 **Acquire data** — Start `AI`, issue a manual trigger, and extract all data from the engine. Before trigger is issued, you should begin inputting data from the function generator into the data acquisition board.

   ```matlab
   start(AI)
   trigger(AI)
   wait(AI,duration + 1)
   ```

   The `wait` function pauses MATLAB until either the acquisition completes or the timeout elapses (whichever comes first). If the timeout elapses, an error occurs. Adding 1 second to the duration allows some margin for the timeout.

   ```matlab
   data = getdata(AI);
   ```
5 Clean up — When you no longer need AI, you should remove it from memory and from the MATLAB workspace.

    delete(AI)
    clear AI

Analyzing the Data
For this experiment, analysis consists of finding the frequency of the input signal and plotting the results. You can find the signal frequency with daqdocfft.

    [f,mag] = daqdocfft(data,Fs,blocksize);

This function, which is shown in “Analyzing the Data” on page 4-19, calculates the FFT of data, and requires the values of SampleRate and SamplesPerTrigger as well as data as inputs. daqdocfft outputs the frequency and magnitude of data, which you can then plot.

The results are given below.

    plot(f,mag)
    grid on
    ylabel('Magnitude (dB)')
    xlabel('Frequency (Hz)')
    title('Frequency Output by Function Generator')
This plot shows the nominal frequency around 1000 Hz. A simple way to find actual frequency is shown below.

\[
[y_{\text{max}}, \text{maxindex}] = \text{max}(\text{mag});
\]

\[
\text{maxindex} = 994
\]

The answer is 994 Hz.
Evaluating the Analog Input Object Status

In this section...
“Status Properties” on page 4-25
“The Display Summary” on page 4-26

Status Properties
The properties associated with the status of your AI object allow you to evaluate

- If the device object is running
- If data is being logged to the engine or to a disk file
- How much data has been acquired
- How much data is available to be extracted from the engine

The analog input status properties are given below.

Table 4-5  Analog Input Status Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logging</td>
<td>Indicate if data is being logged to memory or to a disk file.</td>
</tr>
<tr>
<td>Running</td>
<td>Indicate if the device object is running.</td>
</tr>
<tr>
<td>SamplesAcquired</td>
<td>Indicate the number of samples acquired per channel.</td>
</tr>
<tr>
<td>SamplesAvailable</td>
<td>Indicate the number of samples available per channel in the data acquisition engine.</td>
</tr>
</tbody>
</table>

When you issue the start function, Running is automatically set to On. When the trigger executes, Logging is automatically set to On and SamplesAcquired keeps a running count of the total number of samples per channel that have been logged to the engine or a disk file. SamplesAvailable tells you how many samples per channel are available to be extracted from the engine with the getdata function.
When the requested number of samples are acquired, `SamplesAcquired` reflects this number, and both `Running` and `Logging` are automatically set to `Off`. When you extract all the samples from the engine, `SamplesAvailable` is 0.

**The Display Summary**

You can invoke the display summary by typing an AI object or a channel object at the MATLAB command line, or by excluding the semicolon when

- Creating an AI object
- Adding channels
- Configuring property values using the dot notation

You can also display summary information via the Workspace browser by right-clicking a device object and selecting **Explore > Display Summary** from the context menu.

The displayed information reflects many of the basic setup properties described in “Configuring Analog Input Properties” on page 4-10, and is designed so you can quickly evaluate the status of your data acquisition session. The display is divided into two main sections: general information and channel information.

**General Summary Information**

The general display summary includes the device object type and the hardware device name, followed by this information:

- Acquisition parameters
  - The sampling rate
  - The number of samples to acquire per trigger
  - The acquisition duration for each trigger
  - The destination for logged data
- Trigger parameters
  - The trigger type
- The number of triggers, including the number of triggers already executed
- The engine status
  - Whether the engine is logging data, waiting to start, or waiting to trigger
  - The number of samples acquired since starting
  - The number of samples available to be extracted with getdata

**Channel Summary Information**
The channel display summary includes property values associated with

- The hardware channel mapping
- The channel name
- The engineering units

The display summary for the example given in “Acquiring Data with a Sound Card” on page 4-17 before `start` is issued is shown below.

```
Display Summary of Analog Input (AI) Object Using 'AudioPCI Record'.

Acquisition Parameters: 8000 samples per second on each channel.
                        8000 samples per trigger on each channel.
                        1 sec. of data to be logged per trigger.
                        Log data to 'Memory' on trigger.

Trigger Parameters: 1 'Manual' trigger(s) on TRIGGER.

Engine status: Waiting for START.
               0 samples acquired since starting.
               0 samples available for GETDATA.

AI object contains channel(s):

Index: ChannelName: HwChannel: InputRange: SensorRange: UnitsRange: Units:
      1   'Mono'    1       [-1 1]      [-1 1]      [-1 1]     'Volts'
```
You can use the Channel property to display only the channel summary information.

AI.Channel
Doing More with Analog Input

This chapter presents the complete analog input functionality available to you with Data Acquisition Toolbox. Properties and functions are described in a way that reflects the typical procedures you will use to configure an analog input session. The sections are as follows.

Configuring and Sampling Input Channels (p. 5-2) Configure hardware characteristics related to the input channel type, the sampling rate, and the channel skew

Managing Acquired Data (p. 5-8) Preview data and extract data from memory

Configuring Analog Input Triggers (p. 5-19) Initiate the logging of acquired data to memory or to a disk file

Events and Callbacks (p. 5-45) Enhance your analog input session using events and callbacks

Linearly Scaling the Data: Engineering Units (p. 5-57) Configure engineering units properties so that output data is linearly scaled
Configuring and Sampling Input Channels

In this section...

“Properties Associated with Configuring and Sampling Input Channels” on page 5-2
“Input Channel Configuration” on page 5-2
“Sampling Rate” on page 5-5
“Channel Skew” on page 5-6

Properties Associated with Configuring and Sampling Input Channels

The hardware you are using has characteristics that satisfy your specific application needs. Some of the most important hardware characteristics determine your configuration.

Table 5-1  Analog Input Properties Related to Sampling Channels

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChannelSkew</td>
<td>Specify the time between consecutive scanned hardware channels.</td>
</tr>
<tr>
<td>ChannelSkewMode</td>
<td>Specify how the channel skew is determined.</td>
</tr>
<tr>
<td>InputType</td>
<td>Specify the analog input hardware channel configuration.</td>
</tr>
<tr>
<td>SampleRate</td>
<td>Specify the per-channel rate at which analog data is converted to digital data.</td>
</tr>
</tbody>
</table>

Input Channel Configuration

You can configure your hardware input channels with the InputType property. The device-specific values for this property are given below.
## Table 5-2 InputType Property Values

<table>
<thead>
<tr>
<th>Vendor</th>
<th>InputType Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advantech</td>
<td>Differential</td>
</tr>
<tr>
<td>Keithley</td>
<td>Differential</td>
</tr>
<tr>
<td>Measurement Computing</td>
<td>Differential</td>
</tr>
<tr>
<td>Sound Cards</td>
<td>AC-Coupled</td>
</tr>
<tr>
<td>VXI Technology</td>
<td>Differential</td>
</tr>
</tbody>
</table>

The `InputType` value determines the number of hardware channels you can add to a device object. You can return the channel IDs with the `daqhwinfo` function. For example, suppose you create the analog input object `ai` for a National Instruments board. To display the differential channel IDs:

```matlab
ai = analoginput('nidaq','Dev1');
hwinfo = daqhwinfo(ai);
hwinfo.DifferentialIDs
```

```matlab
ans =
     0    1    2    3    4    5    6    7
```

In contrast, the single-ended channel IDs would be numbered 0 through 15.

**Note** If you change the `InputType` value to decrease the number of channels contained by the analog input object, the system returns a warning and deletes all channels.

### Advantech, Keithley, and Measurement Computing Devices

For Advantech, Keithley, and Measurement Computing devices, `InputType` can be Differential or SingleEnded. Channels configured for differential input are not connected to a fixed reference such as earth, and the input
signals are measured as the difference between two terminals. Channels configured for single-ended input are connected to a common ground, and input signals are measured with respect to this ground.

**Note** Keithley and VXI Technology adaptors will be deprecated in a future version of the Toolbox. If you create a Data Acquisition Toolbox object for the ’keithley’ or ’hp1432’ adaptors in version R2007a, you will receive a warning stating that these adaptors will be removed in a future release.

**VXI Technology Devices**

For VXI Technology devices, the only valid **InputType** value is **Differential**. Channels configured for differential input are not connected to a fixed reference such as earth, and the input signals are measured as the difference between two terminals.

**National Instruments Devices**

For National Instruments devices, **InputType** can be **Differential**, **SingleEnded**, **NonReferencedSingleEnded**, or **PseudoDifferential**. Channels configured for differential input are not connected to a fixed reference such as earth, and input signals are measured as the difference between two terminals. Channels configured for single-ended input are connected to a common ground, and input signals are measured with respect to this ground. Channels configured for nonreferenced single-ended input are connected to their own ground reference, and input signals are measured with respect to this reference. The ground reference is tied to the negative input of the instrumentation amplifier. Channels configured for pseudodifferential input are all referred to a common ground but this ground is not connected to the computer ground.

The number of channels that you can add to a device object depends on the **InputType** property value. Most National Instruments boards have 16 or 64 single-ended inputs and 8 or 32 differential inputs, which are interleaved in banks of 8. This means that for a 64 channel board with single-ended inputs, you can add all 64 channels. However, if the channels are configured for differential input, you can only add channels 0-7, 16-23, 32-39, and 48-55.
**Sound Cards**
For sound cards, the only valid **InputType** value is AC-Coupled. When input channels are AC-coupled, they are connected so that constant (DC) signal levels are suppressed, and only nonzero AC signals are measured.

**Sampling Rate**
The sampling rate is defined as the per-channel rate (in samples/second) that an analog input subsystem converts analog data to digital data. You specify the sampling rate with the **SampleRate** property.

The maximum rate at which channels are sampled depends on the type of hardware you are using. If you are using simultaneous sample and hold (SS/H) hardware such as a sound card, then the maximum sampling rate for each channel is given by the maximum board rate. For example, suppose you create the analog input object `ai` for a sound card and configure it for stereo operation. If the device has a maximum rate of 48.0 kHz, then the maximum sampling rate per channel is 48.0 kHz.

```matlab
ai = analoginput('winsound');
addchannel(ai,1:2);
set(ai,'SampleRate',48000)
```

If you are using scanning hardware such as a National Instruments board, then the maximum sampling rate your hardware is rated at typically applies for one channel. Therefore, the maximum sampling rate per channel is given by the formula

\[
\text{Maximum sampling rate per channel} = \frac{\text{Maximum board rate}}{\text{Number of channels scanned}}
\]

For example, suppose you create the analog input object `ai` for a National Instruments board and add 10 channels to it. If the device has a maximum rate of 100 kHz, then the maximum sampling rate per channel is 10 kHz.

```matlab
ai = analoginput('nidaq','Dev1');
set(ai,'InputType','SingleEnded')
addchannel(ai,0:9);
set(ai,'SampleRate',10000)
```
Typically, you can achieve this maximum rate only under ideal conditions. In practice, the sampling rate depends on several characteristics of the analog input subsystem including the settling time, the gain, and the channel skew. Channel skew is discussed in “Channel Skew” on page 5-6.

**Note** Whenever the *SampleRate* value is changed, the *BufferingConfig* property value is recalculated by the engine if the *BufferingMode* property is set to Auto. Because *BufferingConfig* indicates the memory used by the engine, you should monitor this property closely.

### Channel Skew

Many data acquisition devices have one A/D converter that is multiplexed to all input channels. If you sample multiple input channels from scanning hardware, then each channel is sampled sequentially following this procedure:

1. A single input channel is sampled.
2. The analog signal is converted to a digital value.
3. The process is repeated for every input channel being used.

Because these channels cannot be sampled simultaneously, a time gap exists between consecutively sampled channels. This time gap is called the *channel skew*. The channel skew and the sample period are illustrated below.

![Diagram of channel skew and sample period](image-url)
As shown in the preceding figure, a scan occurs when all channels in a group are sampled once and the scan rate is defined as the rate at which every channel in the group is sampled. The properties associated with configuring the channel skew are given below.

### Table 5-3 Channel Skew Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChannelSkew</td>
<td>Specify the time between consecutive scanned hardware channels.</td>
</tr>
<tr>
<td>ChannelSkewMode</td>
<td>Specify how the channel skew is determined.</td>
</tr>
</tbody>
</table>

ChannelSkew and ChannelSkewMode are configurable only for scanning hardware and not for simultaneous sample and hold (SS/H) hardware. For SS/H hardware, ChannelSkewMode can only be None, and ChannelSkew can only be 0. The values for ChannelSkewMode are given below.

### Table 5-4 ChannelSkewMode Property Values

<table>
<thead>
<tr>
<th>ChannelSkewModeValue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>No channel skew is defined. This is the only valid value for simultaneous sample and hold (SS/H) hardware.</td>
</tr>
<tr>
<td>Equisample</td>
<td>The channel skew is automatically calculated as ([(\text{sampling rate})(\text{number of channels})]^{-1}).</td>
</tr>
<tr>
<td>Manual</td>
<td>The channel skew must be set with the ChannelSkew property.</td>
</tr>
<tr>
<td>Minimum</td>
<td>The channel skew is given by the smallest value supported by the hardware.</td>
</tr>
</tbody>
</table>

If ChannelSkewMode is Minimum or Equisample, then ChannelSkew indicates the appropriate read-only value. If ChannelSkewMode is set to Manual, you must specify the channel skew with ChannelSkew.
Managing Acquired Data

In this section...

“Analog Input Data Management Properties” on page 5-8
“Previewing Data” on page 5-8
“Rules for Using peekdata” on page 5-9
“Extracting Data from the Engine” on page 5-11
“Returning Time Information” on page 5-16

Analog Input Data Management Properties

At the core of any analog input application lies the data you acquire from a sensor and input into your computer for subsequent analysis. The role of the analog input subsystem is to convert analog data to digitized data that can be read by the computer.

After data is extracted from the engine, you can analyze it, save it to disk, etc. In addition to these two functions, there are several properties associated with managing acquired data. These properties are as follows:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SamplesAcquired</td>
<td>Indicate the number of samples acquired per channel.</td>
</tr>
<tr>
<td>SamplesAvailable</td>
<td>Indicate the number of samples available per channel in the data acquisition engine.</td>
</tr>
<tr>
<td>SamplesPerTrigger</td>
<td>Specify the number of samples to acquire for each channel group member for each trigger that occurs.</td>
</tr>
</tbody>
</table>

Previewing Data

Before you extract and analyze acquired data, you might want to examine (preview) the data as it is being acquired. Previewing the data allows you to determine if the hardware is performing as expected and if your acquisition process is configured correctly. Once you are convinced that your system is in order, you might still want to monitor the data even as it is being analyzed or saved to disk.
Previewing data is managed with the `peekdata` function. For example, to preview the most recent 1000 samples acquired for the analog input object `ai`:

```
data = peekdata(ai,1000);
```

After `start` is issued, you can call `peekdata`. `peekdata` is a *nonblocking* function because it immediately returns control to MATLAB. Therefore, samples might be missed or repeated.

When a `peekdata` call is processed, the most recent samples requested are immediately returned, but the data is not extracted from the engine. In other words, `peekdata` provides a “snapshot” of the most recent requested samples. This situation is illustrated below.

If another `peekdata` call is issued, then once again, only the most recent requested samples are returned. This situation is illustrated below.

### Rules for Using `peekdata`

Using `peekdata` to preview data follows these rules:
You can call `peekdata` before a trigger executes. Therefore, `peekdata` is useful for previewing data before it is logged to the engine or a disk file.

In most cases, you will call `peekdata` while the device object is running. However, you can call `peekdata` once after the device object stops running.

If the specified number of preview samples is greater than the number of samples currently acquired, all available samples are returned with a warning message stating that the requested number of samples were not available.

**Example: Polling the Data Block**

Under certain circumstances, you might want to poll the data block. Polling the data block is useful when calling `peekdata` because this function does not block execution control. For example, you can issue `peekdata` calls based on the number of samples acquired by polling the `SamplesAcquired` property.

You can run this example by typing `daqdoc5_1` at the MATLAB command line.

1 **Create a device object** — Create the analog input object `AI` for a sound card. The available adaptors and hardware IDs are found with `daqhwinfo`.

   ```matlab
   AI = analoginput('winsound');
   %AI = analoginput('nidaq','Dev1');
   %AI = analoginput('mcc',1);
   ```

2 **Add channels** — Add one hardware channel to `AI`.

   ```matlab
   addchannel(AI,1);
   %addchannel(AI,0); % For NI and MCC
   ```

3 **Configure property values** — Define a 10 second acquisition, set up a plot, and store the plot handle and title handle in the variables `P` and `T`, respectively.

   ```matlab
   duration = 10; % Ten second acquisition
   ActualRate = get(AI,'SampleRate');
   set(AI,'SamplesPerTrigger',duration*ActualRate)
   figure
   set(gcf,'doublebuffer','on') %Reduce plot flicker
   P = plot(zeros(1000,1));
   ```
Managing Acquired Data

4 Acquire data — Start AI and update the display for each 1000 samples acquired by polling SamplesAcquired. The drawnow command forces MATLAB to update the plot. Because peekdata is used, all acquired data might not be displayed.

```
start(AI)
i = 1;
while AI.SamplesAcquired < AI.SamplesPerTrigger
    while AI.SamplesAcquired < 1000*i
        end
    data = peekdata(AI,1000);
    set(P,'ydata',data);
    set(T,'String',[sprintf('Peekdata calls: '),num2str(i)]);
    drawnow
    i = i + 1;
end
```

Make sure AI has stopped running before cleaning up the workspace.

```
wait(AI,2)
```

5 Clean up — When you no longer need AI, you should remove it from memory and from the MATLAB workspace.

```
delete(AI)
clear AI
```

As you run this example, you might not preview all 80,000 samples stored in the engine. This is because the engine might store data faster than it can be displayed, and peekdata does not guarantee that all requested samples are processed.

Extracting Data from the Engine

Many data acquisition applications require that data is acquired at a fixed (often high) rate, and that the data is processed in some way immediately after it is collected. For example, you might want to perform an FFT on the acquired data and then save it to disk. When processing data, you must
extract it from the engine. If acquired data is not extracted in a timely fashion, it can be overwritten.

Data is extracted from the engine with the `getdata` function. For example, to extract 1000 samples for the analog input object `ai`:

```matlab
data = getdata(ai,1000);
```

In addition to returning acquired data, `getdata` can return relative time, absolute time, and event information. As shown below, `data` is an m-by-n array containing acquired data where m is the number of samples and n is the number of channels.

```
[ d_{11} \ d_{12} \ ... \ d_{1n} \\
 d_{21} \ d_{22} \ ... \ d_{2n} \\
 d_{31} \ d_{32} \ ... \ d_{3n} \\
 \vdots \ \vdots \ \vdots \\
 d_{m1} \ d_{m2} \ ... \ d_{mn} ]
```

`getdata` is considered a *blocking* function because it returns control to MATLAB only when the requested data is available. Therefore, samples are not missed or repeated. When a trigger executes, acquired data fills the engine. When a `getdata` call is processed, the requested samples are returned when the data is available, and then extracted from the engine.

As shown below, if a fraction of the data stored in the engine is extracted, then `getdata` always extracts the oldest data.

```
\begin{center}
\begin{tikzpicture}
  \fill[gray!80] (0,0) rectangle (6,4);
  \node at (3,2) {Extract the requested data};
  \node at (3,1) {Data stored in engine};
  \draw (0,0) -- (6,0) -- (6,4) -- (0,4) -- (0,0);
  \node at (3,5) {Time};
  \node at (6,1) {\ldots};
\end{tikzpicture}
\end{center}
```
If another getdata call is issued, then once again, the oldest samples are extracted.

```
Time
```

![Diagram: Data stored in engine and data extracted from the engine]

**Rules for Using getdata**

Using getdata to extract data stored in the engine follows these rules:

- If the requested number of samples is greater than the samples to be acquired, then an error is returned.
- If the requested data is not returned in the expected amount of time, an error is returned. The expected time to return data is given by the time it takes the engine to fill one data block plus the time specified by the Timeout property.
- You can issue `^C` (Ctrl+C) while getdata is blocking. This will not stop the acquisition but will return control to MATLAB.
- The SamplesAcquired property keeps a running count of the total number of samples per channel that have been acquired.
- The SamplesAvailable property tells you how many samples you can extract from the engine per channel.
- MATLAB supports math operations only for the double data type. Therefore, if you extract data using the native data type of your hardware (typically int16), you must convert the data to doubles before performing math operations.

**Example: Previewing and Extracting Data**

Suppose you have a data acquisition application that is particularly time consuming. By previewing the data, you can ascertain whether the acquisition is proceeding as expected without acquiring all the data. If it is not, then you
can abort the session and diagnose the problem. This example illustrates how you might use peekdata and getdata together in such an application.

You can run this example by typing daqdoc5_2 at the MATLAB command line.

1 **Create a device object** — Create the analog input object AI for a sound card. The installed adaptors and hardware IDs are found with daqhwinfo.

   AI = analoginput('winsound');
   %AI = analoginput('nidaq','Dev1');
   %AI = analoginput('mcc',1);

2 **Add channels** — Add one hardware channel to AI.

   chan = addchannel(AI,1);
   %chan = addchannel(AI,0); % For NI and MCC

3 **Configure property values** — Define a 10-second acquisition, set up the plot, and store the plot handle in the variable P. The amount of data to display is given by preview.

   duration = 10; % Ten second acquisition
   set(AI,'SampleRate',8000)
   ActualRate = get(AI,'SampleRate');
   set(AI,'SamplesPerTrigger',duration*ActualRate)
   preview = duration*ActualRate/100;
   subplot(211)
   set(gcf,'doublebuffer','on')
   P = plot(zeros(preview,1)); grid on
   title('Preview Data')
   xlabel('Samples')
   ylabel('Signal Level (Volts)')

4 **Acquire data** — Start AI and update the display using peekdata every time an amount of data specified by preview is stored in the engine by polling SamplesAcquired. The drawnow command forces MATLAB to update the plot. After all data is acquired, it is extracted from the engine. Note that whenever peekdata is used, all acquired data might not be displayed.

   start(AI)
while AI.SamplesAcquired < preview
end
while AI.SamplesAcquired < duration*ActualRate
    data = peekdata(AI,preview);
    set(P,'ydata',data)
    drawnow
end

Extract all the acquired data from the engine, and plot the data.

wait(AI,duration+1)
data = getdata(AI);
subplot(212), plot(data), grid on
title('All Acquired Data')
xlabel('Samples')
ylabel('Signal level (volts)')

5 Clean up — When you no longer need AI, you should remove it from memory and from the MATLAB workspace.

    delete(AI)
    clear AI
The data is shown below.

**Returning Time Information**
You can return relative time and absolute time information with the `getdata` function. Relative time is associated with the extracted data. Absolute time is associated with the first trigger executed.

**Relative Time**
To return data and relative time information for the analog input object `ai`:

```matlab
[data,time] = getdata(ai);
```

time is an m-by-1 array of relative time values where m is the number of samples returned. time = 0 corresponds to the first sample logged by the data acquisition engine, and time is measured continuously until the acquisition is stopped.
The relationship between the samples acquired and the relative time for each sample is shown below for \( m \) samples and \( n \) channels.

\[
\begin{bmatrix}
d_{11} & d_{12} & \cdots & d_{1n} \\
d_{21} & d_{22} & \cdots & d_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
d_{m1} & d_{m2} & \cdots & d_{mn} \\
\end{bmatrix}
\begin{bmatrix}
t_1 \\
t_2 \\
\vdots \\
t_m \\
\end{bmatrix}
\]

**Absolute Time**

To return data, relative time information, and the absolute time of the first trigger for the analog input object \( ai \):

\[[\text{data, time, abstime}] = \text{getdata}(ai);\]

The absolute time is returned using the MATLAB `clock` format.

\[[\text{year month day hour minute seconds}]\]

The absolute time from the `getdata` call is

\[
\text{abstime} =
\begin{bmatrix}
1.9990 \\
0.0020 \\
0.0190 \\
0.0130 \\
0.0260 \\
0.0208 \\
\end{bmatrix}
\]

To convert the `clock` vector to a more convenient form:

\[
t = \text{fix(abstime)};
\text{fprintf}('%d:%d:%d',t(4),t(5),t(6))
\]

\[
\text{ans} =
13:26:20
\]

The absolute time of the first trigger is also recorded by the `InitialTriggerTime` property.
Note that absolute times are recorded by the EventLog property for each trigger executed. You can always find the absolute time associated with a data sample by adding its relative time to the absolute time of the associated trigger. Refer to “Recording and Retrieving Event Information” on page 5-48 for more information about returning absolute time information with the EventLog property.
Configuring Analog Input Triggers

<table>
<thead>
<tr>
<th>In this section...</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Analog Input Trigger Properties” on page 5-19</td>
</tr>
<tr>
<td>“Defining a Trigger: Trigger Types and Conditions” on page 5-20</td>
</tr>
<tr>
<td>“Executing the Trigger” on page 5-25</td>
</tr>
<tr>
<td>“Trigger Delays” on page 5-26</td>
</tr>
<tr>
<td>“Repeating Triggers” on page 5-30</td>
</tr>
<tr>
<td>“How Many Triggers Occurred?” on page 5-35</td>
</tr>
<tr>
<td>“When Did the Trigger Occur?” on page 5-36</td>
</tr>
<tr>
<td>“Device-Specific Hardware Triggers” on page 5-37</td>
</tr>
</tbody>
</table>

Analog Input Trigger Properties

An analog input trigger is defined as an event that initiates data logging. You can log data to the engine (memory) and to a disk file. As shown in the figure below, when a trigger occurs, the Logging property is automatically set On and data is stored in the specified target.

![Logging Diagram]

When defining a trigger, you must specify the trigger type. Additionally, you might need to specify one or more of these parameters:

- A trigger condition and trigger condition value
- The number of times to repeat the trigger
- A trigger delay
- A callback function to execute when the trigger event occurs
Properties associated with analog input triggers are as follows:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitialTriggerTime</td>
<td>Indicate the absolute time of the first trigger.</td>
</tr>
<tr>
<td>ManualTriggerHwOn</td>
<td>Specify that the hardware device starts when a manual trigger is issued.</td>
</tr>
<tr>
<td>TriggerFcn</td>
<td>Specify the M-file callback function to execute when a trigger occurs.</td>
</tr>
<tr>
<td>TriggerChannel</td>
<td>Specify the channel serving as the trigger source.</td>
</tr>
<tr>
<td>TriggerCondition</td>
<td>Specify the condition that must be satisfied before a trigger executes.</td>
</tr>
<tr>
<td>TriggerConditionValue</td>
<td>Specify one or more voltage values that must be satisfied before a trigger executes.</td>
</tr>
<tr>
<td>TriggerDelay</td>
<td>Specify the delay value for data logging.</td>
</tr>
<tr>
<td>TriggerDelayUnits</td>
<td>Specify the units in which trigger delay data is measured.</td>
</tr>
<tr>
<td>TriggerRepeat</td>
<td>Specify the number of additional times the trigger executes.</td>
</tr>
<tr>
<td>TriggersExecuted</td>
<td>Indicate the number of triggers that execute.</td>
</tr>
<tr>
<td>TriggerType</td>
<td>Specify the type of trigger to execute.</td>
</tr>
</tbody>
</table>

Except for TriggerFcn, these trigger-related properties are discussed in the following sections. TriggerFcn is discussed in “Events and Callbacks” on page 5-45.

**Defining a Trigger: Trigger Types and Conditions**

This section contains the following topics:

- “Immediate Trigger” on page 5-22
- “Manual Trigger” on page 5-22
Defining a trigger for an analog input object involves specifying the trigger type with the \texttt{TriggerType} property. You can think of the trigger type as the source of the trigger. For some trigger types, you might need to specify a trigger condition and a trigger condition value. Trigger conditions are specified with the \texttt{TriggerCondition} property, while trigger condition values are specified with the \texttt{TriggerConditionValue} property.

The analog input \texttt{TriggerType} and \texttt{TriggerCondition} values are given below.

\begin{table}
\caption{Analog Input \texttt{TriggerType} and \texttt{TriggerCondition} Values}
\centering
\begin{tabular}{|l|l|l|}
\hline
\textbf{TriggerType Value} & \textbf{TriggerCondition Value} & \textbf{Description} \\
\hline
{Immediate} & None & The trigger occurs just after you issue the \texttt{start} function. \\
\hline
Manual & None & The trigger occurs just after you manually issue the \texttt{trigger} function. \\
\hline
Software & {Rising} & The trigger occurs when the signal has a positive slope when passing through the specified value. \\
\hline
Falling & & The trigger occurs when the signal has a negative slope when passing through the specified value. \\
\hline
Leaving & & The trigger occurs when the signal leaves the specified range of values. \\
\hline
Entering & & The trigger occurs when the signal enters the specified range of values. \\
\hline
\end{tabular}
\end{table}

For some devices, additional trigger types and trigger conditions are available. Refer to the \texttt{TriggerType} and \texttt{TriggerCondition} reference pages in Chapter 14, “Base Properties — Alphabetical List” for these device-specific values.
Trigger types are grouped into two main categories:

- Device-independent triggers
- Device-specific hardware triggers

The trigger types shown above are device-independent triggers because they are available for all supported hardware. For these trigger types, the callback that initiates the trigger event involves satisfying a trigger condition in the engine (software trigger type), or issuing a toolbox function (`start` or `trigger`). Conversely, device-specific hardware triggers depend on the specific hardware device you are using. For these trigger types, the callback that initiates the trigger event involves an external analog or digital signal.

Device-specific hardware triggers for National Instruments, Measurement Computing, and VXI Technology devices are discussed in “Device-Specific Hardware Triggers” on page 5-37. Device-independent triggers are discussed below.

**Immediate Trigger**
If `TriggerType` is `Immediate` (the default value), the trigger occurs immediately after the `start` function is issued. You can configure an analog input object for continuous acquisition by using an immediate trigger and setting `SamplesPerTrigger` or `TriggerRepeat` to `inf`. Trigger repeats are discussed in “Repeating Triggers” on page 5-30.

**Manual Trigger**
If `TriggerType` is `Manual`, the trigger occurs just after you issue the `trigger` function. A manual trigger might provide you with more control over the data that is logged. For example, if the acquired data is noisy, you can preview the data using `peekdata`, and then manually execute the trigger after you observe that the signal is well-behaved.

**Software Trigger**
If `TriggerType` is `Software`, the trigger occurs when a signal satisfying the specified condition is detected on the hardware channel specified by the `TriggerChannel` property. The trigger condition is specified as either a voltage
value and slope, or a range of voltage values using the TriggerCondition and TriggerConditionValue properties.

Some acquisition speeds on some devices may not be available when the TriggerType is Software, due to hardware limitations. When you set TriggerType to Software, the device is put into a continuous acquisition mode, and acquisition begins when you call start. The data collected is analyzed as it comes in to detect the trigger condition you have specified. If the data does not contain your trigger condition, it is discarded. When the trigger condition is met, the engine begins storing data. This data can be retrieved using getdata. With some devices, the maximum speed of the device changes when it is running in continuous acquisition mode, making some speeds unavailable when setting TriggerType to Software.

**Example: Voice Activation Using a Software Trigger**

This example demonstrates how to configure an acquisition with a sound card based on voice activation. The sample rate is set to 44.1 kHz and data is logged when an acquired sample has a value greater than or equal to 0.2 volt and a rising slope. A portion of the data is then extracted from the engine and plotted.

You can run this example by typing daqdoc5_3 at the MATLAB command line.

1 **Create a device object** — Create the analog input object AIVoice for a sound card. The installed adaptors and hardware IDs are found with daqhwinfo.

   ```matlab
   AIVoice = analoginput('winsound');
   %AIVoice = analoginput('nidaq','Dev1');
   %AIVoice = analoginput('mcc',1);
   ``

2 **Add channels** — Add one hardware channel to AIVoice.

   ```matlab
   chan = addchannel(AIVoice,1);
   %chan = addchannel(AIVoice,0); % For NI and MCC
   ``

3 **Configure property values** — Define a 2-second acquisition and configure a software trigger. The source of the trigger is chan, and the trigger executes when a rising voltage level has a value of at least 0.2 volt.

   ```matlab
   ```
duration = 2; % two second acquisition
set(AIVoice,'SampleRate',44100)
ActualRate = get(AIVoice,'SampleRate');
set(AIVoice,'SamplesPerTrigger',ActualRate*duration)
set(AIVoice,'TriggerChannel',chan)
set(AIVoice,'TriggerType','Software')
set(AIVoice,'TriggerCondition','Rising')
set(AIVoice,'TriggerConditionValue',0.2)

4 Acquire data — Start AIVoice, acquire the specified number of samples, and extract the first 1000 samples from the engine as sample-time pairs. Display the number of samples remaining in the engine.

    start(AIVoice)
    wait(AIVoice, duration+1)
    [data,time] = getdata(AIVoice,1000);
    remsamp = num2str(AIVoice.SamplesAvailable);
    disp(['Number of samples remaining in engine: ', remsamp])

Plot all extracted data.

    plot(time,data)
    drawnow
    xlabel('Time (sec.)')
    ylabel('Signal Level (Volts)')
    grid on

5 Clean up — When you no longer need AIVoice, you should remove it from memory and from the MATLAB workspace.

    delete(AIVoice)
    clear AIVoice
Note that when using software triggers, you must specify the TriggerType value before the TriggerCondition value. The output from this example is shown below.

The first logged sample has a signal level value of at least 0.2 volt, and this value corresponds to time = 0. Note that after you issue the getdata function, 87,200 samples remain in the engine.

\[
\text{AIVoice.SamplesAvailable} \\
\text{ans} = \\
87200
\]

**Executing the Trigger**

For an analog input trigger to occur, you must follow these steps:

1. Configure the appropriate trigger properties.
2. Issue the start function.
3. Issue the trigger function if TriggerType value is Manual.
Once the trigger occurs, data logging is initiated. The device object and hardware device stop executing when the requested samples are acquired, a run-time error occurs, or you issue the `stop` function.

**Note** After a trigger occurs, the number of samples specified by `SamplesPerTrigger` is acquired for each channel group member before the next trigger can occur.

**Trigger Delays**

Trigger delays allow you to control exactly when data is logged after a trigger occurs. You can log data either before the trigger or after the trigger. Logging data before the trigger occurs is called *pretriggering*, while logging data after a trigger occurs is called *posttriggering*.

You configure trigger delays with the `TriggerDelay` property. Pretriggers are specified by a negative `TriggerDelay` value, while posttriggers are specified by a positive `TriggerDelay` value. You can delay data logging in time or in samples using the `TriggerDelayUnits` property. When `TriggerDelayUnits` is set to `Samples`, data logging is delayed by the specified number of samples. When the `TriggerDelayUnits` property is set to `Seconds`, data logging is delayed by the specified number of seconds.
Capturing Pretrigger Data
In some circumstances, you might want to capture data before the trigger occurs. Such data is called pretrigger data. When capturing pretrigger data, the SamplesPerTrigger property value includes the data captured before and after the trigger occurs. Capturing pretrigger data is illustrated below.

You can capture pretrigger data for manual triggers and software triggers. If TriggerType is Manual, and the trigger function is issued before the trigger delay passes, then a warning is returned and the trigger is ignored (the trigger event does not occur).

You cannot capture pretrigger data for immediate triggers or device-specific hardware triggers.

**Note** Pretrigger data has negative relative time values associated with it. This is because time = 0 corresponds to the time the trigger event occurs and data logging is initiated.
Capturing Posttrigger Data

In some circumstances, you might want to capture data after the trigger occurs. Such data is called posttrigger data. When capturing posttrigger data, the `SamplesPerTrigger` property value and the number of posttrigger samples are equal. Capturing posttrigger data is illustrated below.

You can capture posttrigger data using any supported trigger type.

Example: Voice Activation and Pretriggers

This example modifies `daqdoc5_3` such that 500 pretrigger samples are acquired. You can run this example by typing `daqdoc5_4` at the MATLAB command line.

1 Create a device object — Create the analog input object `AIVoice` for a sound card. The installed adaptors and hardware IDs are found with `daqhwinfo`.

   ```matlab
   AIVoice = analoginput('winsound');
   %AIVoice = analoginput('nidaq','Dev1');
   %AIVoice = analoginput('mcc',1);
   ``

2 Add channels — Add one hardware channel to `AIVoice`.

   ```matlab
   chan = addchannel(AIVoice,1);
   %chan = addchannel(AIVoice,0); % For NI and MCC
   ``

3 Configure property values — Define a 2-second acquisition, and configure a software trigger. The source of the trigger is `chan`, and the trigger executes when a rising voltage level has a value of at least 0.2 volt. Additionally, 500 pretrigger samples are collected.
duration = 2; % two second acquisition
set(AIVoice,'SampleRate',44100)
ActualRate = get(AIVoice,'SampleRate');
set(AIVoice,'SamplesPerTrigger',ActualRate*duration)
set(AIVoice,'TriggerChannel',chan)
set(AIVoice,'TriggerType','Software')
set(AIVoice,'TriggerCondition','Rising')
set(AIVoice,'TriggerConditionValue',0.2)
set(AIVoice,'TriggerDelayUnits','Samples')
set(AIVoice,'TriggerDelay',-500)

4 Acquire data — Start AIVoice, acquire the specified number of samples, and extract the first 1000 samples from the engine as sample-time pairs.

    start(AIVoice)
    wait(AIVoice,duration+1)
    [data,time] = getdata(AIVoice,1000);

Plot all the extracted data.

    plot(time,data)
    xlabel('Time (sec.)')
    ylabel('Signal Level (Volts)')
    grid on

5 Clean up When you no longer need AIVoice, you should remove it from memory and from the MATLAB workspace.

    delete(AIVoice)
    clear AIVoice
The output from this example is shown below. Note that the pretrigger data constitutes half of the 1000 samples extracted from the engine. Additionally, pretrigger data has negative time associated with it because time = 0 corresponds to the time the trigger event occurs and data logging is initiated.

Repeating Triggers
You can configure triggers to occur once (one-shot acquisition) or multiple times. You control trigger repeats with the TriggerRepeat property. If TriggerRepeat is set to its default value of 0, then the trigger occurs once. If TriggerRepeat is set to a positive integer value, then the trigger is repeated the specified number of times. If TriggerRepeat is set to inf, then the trigger repeats continuously and you can stop the device object only by issuing the stop function.

Example: Voice Activation and Repeating Triggers
This example modifies daqdoc5_3 such that two triggers are issued. The specified amount of data is acquired for each trigger and stored in separate variables. The Timeout value is set to five seconds. Therefore, if getdata
does not return the specified number of samples in the time given by the Timeout property plus the time required to acquire the data, the acquisition will be aborted.

You can run this example by typing daqdoc5_5 at the MATLAB command line.

1 Create a device object — Create the analog input object AIVoice for a sound card. The installed adaptors and hardware IDs are found with daqhwinfo.

   AIVoice = analoginput('winsound');
   %AIVoice = analoginput('nidaq','Dev1');
   %AIVoice = analoginput('mcc',1);

2 Add channels — Add one hardware channel to AIVoice.

   chan = addchannel(AIVoice,1);
   %chan = addchannel(AIVoice,0); % For NI and MCC

3 Configure property values — Define a 1-second total acquisition time and configure a software trigger. The source of the trigger is chan, and the trigger executes when a rising voltage level has a value of at least 0.2 volt. Additionally, the trigger is repeated once when the trigger condition is met.

   duration = 0.5; % One-half second acquisition for each trigger
   set(AIVoice,'SampleRate',44100)
   ActualRate = get(AIVoice,'SampleRate');
   set(AIVoice,'Timeout',5)
   set(AIVoice,'SamplesPerTrigger',ActualRate*duration)
   set(AIVoice,'TriggerChannel',chan)
   set(AIVoice,'TriggerType','Software')
   set(AIVoice,'TriggerCondition','Rising')
   set(AIVoice,'TriggerConditionValue',0.2)
   set(AIVoice,'TriggerRepeat',1)

4 Acquire data — Start AIVoice, acquire the specified number of samples, extract all the data from the first trigger as sample-time pairs, and extract all the data from the second trigger as sample-time pairs. Note that you can extract the data acquired from both triggers with the command getdata(AIVoice,44100).


```matlab
start(AIVoice)
wait(AIVoice,duration+1)
[d1,t1] = getdata(AIVoice);
[d2,t2] = getdata(AIVoice);

Plot the data for both triggers.

subplot(211), plot(t1,d1), grid on, hold on
axis([t1(1)-0.05 t1(end)+0.05 -0.8 0.8])
xlabel('Time (sec.)'), ylabel('Signal level (Volts)'),
title('Voice Activation First Trigger')
subplot(212), plot(t2,d2), grid on
axis([t2(1)-0.05 t2(end)+0.05 -0.8 0.8])
xlabel('Time (sec.)'), ylabel('Signal level (Volts)'),
title('Voice Activation Second Trigger')
```

**5 Clean up** — When you no longer need AIVoice, you should remove it from memory and from the MATLAB workspace.

```matlab
delete(AIVoice)
clear AIVoice
```
The data acquired for both triggers is shown below.

As described in “Extracting Data from the Engine” on page 5-11, if you do not specify the amount of data to extract from the engine with \texttt{getdata}, then the amount of data returned is given by the \texttt{SamplesPerTrigger} property. You can return data from multiple triggers with one call to \texttt{getdata} by specifying the appropriate number of samples. When you return data that spans multiple triggers, a NaN is inserted in the data stream between trigger events. Therefore, an extra “sample” (the NaN) is stored in the engine and returned by \texttt{getdata}. Identifying these NaNs allows you to locate where and when each trigger was issued in the data stream.
The figure below illustrates the data stored by the engine during a multiple-trigger acquisition. The data acquired for each trigger is given by the `SamplesPerTrigger` property value. The relative trigger times are shown on the `Time` axis where the first trigger time corresponds to $t_1$ (0 seconds by definition), the second trigger time corresponds to $t_2$, and so on.

The following code modifies `daqdoc5_5` so that multiple-trigger data is extracted from the engine with one call to `getdata`.

```plaintext
returndata = ActualRate*duration*(AIVoice.TriggerRepeat + 1);
start(AIVoice)
wait(AIVoice,duration+1)
[d,t] = getdata(AIVoice,returndata);
```

Plot the data.

```plaintext
plot(t,d)
xlabel('Time (sec.)')
ylabel('Signal Level (Volts)')
title('Voice Activation for Both Triggers')
grid on
```
The multiple-trigger data is shown below.

You can find the relative trigger times by searching for NaNs in the returned data. You can find the index location of the NaN in d or t using the isnan function.

\[
\text{index} = \text{find(isnan(d))}
\]
\[
\text{index} = 22051
\]

With this information, you can find the relative time for the second trigger.

\[
\text{t2time} = \text{t(index+1)}
\]
\[
\text{t2time} = 0.5980
\]

**How Many Triggers Occurred?**

You can find out how many triggers occurred with the TriggersExecuted property value. The trigger number for each trigger executed is also recorded by the EventLog property. A convenient way to access event log information is with the showdaqevents function.
For example, suppose you create the analog input object ai for a sound card and add one channel to it. ai is configured to acquire 40,000 samples with five triggers using the default sampling rate of 8000 Hz.

```matlab
ai = analoginput('winsound');
ch = addchannel(ai,1);
set(ai,'TriggerRepeat',4);
start(ai)
```

TriggersExecuted returns the number of triggers executed.

```matlab
ai.TriggersExecuted
ans =
   5
```

dshowdaqevents returns information for all the events that occurred while ai was executing.

```matlab
dshowdaqevents(ai)
   1 Start       ( 10:22:04,  0 )
   2 Trigger#1  ( 10:22:04,  0 )       Channel: N/A
   3 Trigger#2  ( 10:22:05,  8000 )     Channel: N/A
   4 Trigger#3  ( 10:22:06, 16000 )     Channel: N/A
   5 Trigger#4  ( 10:22:07, 24000 )     Channel: N/A
   6 Trigger#5  ( 10:22:08, 32000 )     Channel: N/A
   7 Stop       ( 10:22:09, 40000 )
```

For more information about recording and retrieving events, refer to “Recording and Retrieving Event Information” on page 5-48.

**When Did the Trigger Occur?**

You can find the absolute time of the first trigger event with the InitialTriggerTime property value. The absolute time is returned using the MATLAB clock format.

```
[year month day hour minute seconds]
```
For example, the absolute time of the first trigger event for the preceding example is

```plaintext
abstime = ai.InitialTriggerTime
abstime =
1.0e+003 *
   1.9990  0.0040  0.0170  0.0100  0.0220  0.0041
```

To convert the clock vector to a more convenient form, you can use the `sprintf` function.

```plaintext
t = fix(abstime);
sprintf('%d:%d:%d', t(4),t(5),t(6))
ans =
  10:22:4
```

You can also use the `showdaqevents` function to return the absolute time of each trigger event. For more information about trigger events, refer to “Recording and Retrieving Event Information” on page 5-48.

**Device-Specific Hardware Triggers**

Many data acquisition devices possess the ability to accept a hardware trigger. Hardware triggers are processed directly by the hardware and can be either a digital signal or an analog signal. Hardware triggers are often used when speed is required because a hardware device can process an input signal much faster than software.

The device-specific hardware triggers are presented to you as additional property values. Hardware triggers for VXI Technology, Measurement Computing, and National Instruments devices are discussed below and in Chapter 14, “Base Properties — Alphabetical List”.

Note that the available hardware trigger support depends on the board you are using. Refer to your hardware documentation for detailed information about its triggering capabilities.

**VXI Technology**

When using VXI Technology hardware, there are additional trigger types and trigger conditions available to you. These device-specific property values
Doing More with Analog Input

fall into two categories: hardware digital triggering and hardware analog triggering.

The device-specific trigger types and trigger conditions are described below and in Chapter 14, “Base Properties — Alphabetical List”.

Analog Input TriggerType and TriggerCondition Property Values for VXI Hardware

<table>
<thead>
<tr>
<th>TriggerType Value</th>
<th>TriggerCondition Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HwDigital</td>
<td>{PositiveEdge}</td>
<td>The trigger occurs when the positive (rising) edge of a digital signal is detected.</td>
</tr>
<tr>
<td></td>
<td>NegativeEdge</td>
<td>The trigger occurs when the negative (falling) edge of a digital signal is detected.</td>
</tr>
<tr>
<td>HwAnalog</td>
<td>{Rising}</td>
<td>The trigger occurs when the analog signal has a positive slope when passing through the specified range of values.</td>
</tr>
<tr>
<td></td>
<td>Falling</td>
<td>The trigger occurs when the analog signal has a negative slope when passing through the specified range of values.</td>
</tr>
<tr>
<td></td>
<td>Leaving</td>
<td>The trigger occurs when the analog signal leaves the specified range of values.</td>
</tr>
<tr>
<td></td>
<td>Entering</td>
<td>The trigger occurs when the analog signal enters the specified range of values.</td>
</tr>
</tbody>
</table>

Note that when TriggerType is HwAnalog, the trigger condition values are all specified as two-element vectors. Setting two trigger levels prevents the module from triggering repeatedly because of a noisy signal.

**Hardware Digital Triggering.** If TriggerType is HwDigital, the trigger is given by a digital (TTL) signal. For example, to trigger your acquisition when the negative edge of a digital signal is detected

```matlab
ai = analoginput('hpe1432',8);
addchannel(ai,1:16);
set(ai,'TriggerType','HwDigital');
set(ai,'TriggerCondition','NegativeEdge');
```
Hardware Analog Triggering. If `TriggerType` is `HwAnalog`, the trigger is given by an analog signal. For example, to trigger your acquisition when the trigger signal is between -4 volts and 4 volts:

```matlab
ai = analoginput('hpe1432',8);
addchannel(ai,1:16);
set(ai,'TriggerType','HwAnalog');
set(ai,'TriggerCondition','Entering');
set(ai,'TriggerConditionValue',[-4.0 4.0]);
set(ai,'TriggerChannel',ai.Channel(1:4));
```

**Measurement Computing**

When using Measurement Computing hardware, there are additional trigger types and trigger conditions available to you. These device-specific property values fall into two categories: hardware digital triggering and hardware analog triggering.

The device-specific trigger types and trigger conditions are described below and in Chapter 14, “Base Properties — Alphabetical List”.

### Analog Input TriggerType and TriggerCondition Values for MCC Hardware

<table>
<thead>
<tr>
<th>TriggerType Value</th>
<th>TriggerCondition Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HwDigital</td>
<td>GateHigh</td>
<td>The trigger occurs as long as the digital signal is high.</td>
</tr>
<tr>
<td></td>
<td>GateLow</td>
<td>The trigger occurs as long as the digital signal is low.</td>
</tr>
<tr>
<td></td>
<td>TrigHigh</td>
<td>The trigger occurs when the digital signal is high.</td>
</tr>
<tr>
<td></td>
<td>TrigLow</td>
<td>The trigger occurs when the digital signal is low.</td>
</tr>
<tr>
<td></td>
<td>TrigPosEdge</td>
<td>The trigger occurs when the positive (rising) edge of the digital signal is detected.</td>
</tr>
<tr>
<td></td>
<td>{TrigNegEdge}</td>
<td>The trigger occurs when the negative (falling) edge of the digital signal is detected.</td>
</tr>
</tbody>
</table>
### Analog Input TriggerType and TriggerCondition Values for MCC Hardware (Continued)

<table>
<thead>
<tr>
<th>TriggerType Value</th>
<th>TriggerCondition Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HwAnalog</td>
<td>{TrigAbove}</td>
<td>The trigger occurs when the analog signal makes a transition from below the specified value to above.</td>
</tr>
<tr>
<td></td>
<td>TrigBelow</td>
<td>The trigger occurs when the analog signal makes a transition from above the specified value to below.</td>
</tr>
<tr>
<td></td>
<td>GateNegHys</td>
<td>The trigger occurs when the analog signal is more than the specified high value. The acquisition stops if the analog signal is less than the specified low value.</td>
</tr>
<tr>
<td></td>
<td>GatePosHys</td>
<td>The trigger occurs when the analog signal is less than the specified low value. The acquisition stops if the analog signal is more than the specified high value.</td>
</tr>
<tr>
<td></td>
<td>GateAbove</td>
<td>The trigger occurs as long as the analog signal is more than the specified value.</td>
</tr>
<tr>
<td></td>
<td>GateBelow</td>
<td>The trigger occurs as long as the analog signal is less than the specified value.</td>
</tr>
<tr>
<td></td>
<td>GateInWindow</td>
<td>The trigger occurs as long as the analog signal is within the specified range of values.</td>
</tr>
<tr>
<td></td>
<td>GateOutWindow</td>
<td>The trigger occurs as long as the analog signal is outside the specified range of values.</td>
</tr>
</tbody>
</table>

**Hardware Digital Triggering.** If TriggerType is HwDigital, the trigger is given by a digital (TTL) signal. For example, to trigger your acquisition when the positive edge of a digital signal is detected:

```matlab
ai = analoginput('mcc',1);
addchannel(ai,0:7);
set(ai,'TriggerType','HwDigital')
set(ai,'TriggerCondition','TrigPosEdge')
```
The diagram below illustrates how you connect a digital trigger signal to a PCI-DAS1602/16 board. A/D External Trigger corresponds to pin 45.

Hardware Analog Triggering. If TriggerType is HwAnalog, the trigger is given by an analog signal. For example, to trigger your acquisition when the trigger signal is between -4 volts and 4 volts:

```matlab
ai = analoginput('mcc',1);
addchannel(ai,0:7);
set(ai,'TriggerType','HwAnalog');
set(ai,'TriggerCondition','GateInWindow');
set(ai,'TriggerConditionValue',[-4.0 4.0]);
```

The diagram below illustrates how you connect an analog trigger signal to a PCI-DAS1602/16 board. AI Ch 0-7 corresponds to pins 2-17, while Analog Trigger In corresponds to pin 43.

National Instruments
When using National Instruments (NI) hardware, there are additional trigger types and trigger conditions available to you. These device-specific property
values fall into two categories: hardware digital triggering and hardware analog triggering.

The device-specific trigger types and trigger conditions are described below and in Chapter 14, “Base Properties — Alphabetical List”.

### Analog Input TriggerType and TriggerCondition Property Values for NI Hardware

<table>
<thead>
<tr>
<th>TriggerType Value</th>
<th>TriggerCondition Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HwDigital</td>
<td>{NegativeEdge}</td>
<td>The trigger occurs when the negative (falling) edge of a digital signal is detected.</td>
</tr>
<tr>
<td></td>
<td>PositiveEdge</td>
<td>The trigger occurs when the positive (rising) edge of a digital signal is detected.</td>
</tr>
<tr>
<td>HwAnalogChannelor</td>
<td>{AboveHighLevel}</td>
<td>The trigger occurs when the analog signal is above the specified value.</td>
</tr>
<tr>
<td>HwAnalogPin</td>
<td>BelowLowLevel</td>
<td>The trigger occurs when the analog signal is below the specified value.</td>
</tr>
<tr>
<td></td>
<td>HighHysteresis</td>
<td>The trigger occurs when the analog signal is greater than the specified high value with hysteresis given by the specified low value.</td>
</tr>
<tr>
<td></td>
<td>InsideRegion</td>
<td>The trigger occurs when the analog signal is inside the specified region.</td>
</tr>
<tr>
<td></td>
<td>LowHysteresis</td>
<td>The trigger occurs when the analog signal is less than the specified low value with hysteresis given by the specified high value.</td>
</tr>
</tbody>
</table>

**Hardware Digital Triggering.** If TriggerType is HwDigital, the trigger occurs when the falling edge of a digital (TTL) signal is detected. The following example illustrates how to configure a hardware digital trigger.

```matlab
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:7);
set(ai,'TriggerType','HwDigital')
```
The diagram below illustrates how you connect a digital trigger signal to an MIO-16E Series board. PFI0/TRIG1 corresponds to pin 11.

**Hardware Analog Triggering.** If `TriggerType` is `HwAnalogPin`, the trigger is given by a low-range analog signal (typically between -10 and 10 volts) connected to the appropriate trigger pin. For example, to trigger your acquisition when the trigger signal is between -4 volts and 4 volts:

```matlab
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:7);
set(ai,'TriggerType','HwAnalogPin')
set(ai,'TriggerCondition','InsideRegion')
set(ai,'TriggerConditionValue',[-4.0 4.0])
```

If `TriggerType` is `HwAnalogChannel`, the trigger is given by an analog signal and the trigger channel is the first channel in the channel group (MATLAB index of one). The valid range of the analog trigger signal is given by the full-scale range of the trigger channel. The following example illustrates how to configure such a trigger where the trigger channel is assigned the descriptive name `TrigChan` and the default `TriggerCondition` property value is used.

```matlab
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:7);
set(ai.Channel(1),'ChannelName','TrigChan')
set(ai,'TriggerChannel',ai.Channel(1))
set(ai,'TriggerType','HwAnalogChannel')
set(ai,'TriggerConditionValue',0.2)
```
The diagram below illustrates how you can connect an analog trigger signal to an MIO-16E Series board.
Events and Callbacks

In this section...

“Understanding Events and Callbacks” on page 5-45
“Event Types” on page 5-45
“Recording and Retrieving Event Information” on page 5-48
“Creating and Executing Callback Functions” on page 5-52
“Examples: Using Callback Properties and Functions” on page 5-54

Understanding Events and Callbacks

You can enhance the power and flexibility of your analog input application by utilizing events. An event occurs at a particular time after a condition is met and might result in one or more callbacks.

While the analog input object is running, you can use events to display a message, display data, analyze data, and so on. Callbacks are controlled through callback properties and callback functions. All event types have an associated callback property. Callback functions are M-file functions that you construct to suit your specific data acquisition needs.

You execute a callback when a particular event occurs by specifying the name of the M-file callback function as the value for the associated callback property. Note that daqcallback is the default value for some callback properties.

Event Types

The analog input event types and associated callback properties are described below.

Analog Input Callback Properties

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data missed</td>
<td>DataMissedFcn</td>
</tr>
<tr>
<td>Input overrange</td>
<td>InputOverRangeFcn</td>
</tr>
</tbody>
</table>
### Analog Input Callback Properties (Continued)

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run-time error</td>
<td>RuntimeErrorFcn</td>
</tr>
<tr>
<td>Samples acquired</td>
<td>SamplesAcquiredFcn</td>
</tr>
<tr>
<td></td>
<td>SamplesAcquiredFcnCount</td>
</tr>
<tr>
<td>Start</td>
<td>StartFcn</td>
</tr>
<tr>
<td>Stop</td>
<td>StopFcn</td>
</tr>
<tr>
<td>Timer</td>
<td>TimerFcn</td>
</tr>
<tr>
<td></td>
<td>TimerPeriod</td>
</tr>
<tr>
<td>Trigger</td>
<td>TriggerFcn</td>
</tr>
</tbody>
</table>

#### Data Missed Event

A data missed event is generated immediately after acquired data is missed. In most cases, data is missed because

- The engine cannot keep up with the rate of acquisition.
- The driver wrote new data into the hardware’s FIFO buffer before the previously acquired data was read. You can usually avoid this problem by increasing the size of the memory block with the BufferingConfig property.

This event executes the callback function specified for the DataMissedFcn property. The default value for DataMissedFcn is daqcallback, which displays the event type and the device object name. When a data missed event occurs, the analog input object is automatically stopped.

#### Input Overrange Event

An input overrange event is generated immediately after an overrange condition is detected for any channel group member. An overrange condition occurs when an input signal exceeds the range specified by the InputRange property.
This event executes the callback function specified for the `InputOverRangeFcn` property. Overrange detection is enabled only when a callback function is specified for `InputOverRangeFcn`, and the analog input object is running.

**Run-time Error Event**
A run-time error event is generated immediately after a run-time error occurs. Additionally, a toolbox error message is automatically displayed to the MATLAB workspace. If an error occurs that is not explicitly handled by the toolbox, then the hardware-specific error message is displayed.

This event executes the callback function specified for `RuntimeErrorFcn`. The default value for `RuntimeErrorFcn` is `daqcallback`, which displays the event type, the time the event occurred, the device object name, and the error message.

Run-time errors include hardware errors and timeouts. Run-time errors do not include configuration errors such as setting an invalid property value.

**Samples Acquired Event**
A samples acquired event is generated immediately after a predetermined number of samples is acquired.

This event executes the callback function specified for the `SamplesAcquiredFcn` property every time the number of samples specified by `SamplesAcquiredFcnCount` is acquired for each channel group member.

You should use `SamplesAcquiredFcn` if you must access each sample that is acquired. However, if you are performing a CPU-intensive task with the data, then system performance might be adversely affected. If you do not have this requirement, you might want to use the `TimerFcn` property.

**Start Event**
A start event is generated immediately after the `start` function is issued. This event executes the callback function specified for `StartFcn`. When the `StartFcn` M-file has finished executing, `Running` is automatically set to `On` and the device object and hardware device begin executing. The device object is not started if an error occurs while executing the callback function.
**Stop Event**
A stop event is generated immediately after the device object and hardware device stop running. This occurs when

- The stop function is issued.
- The requested number of samples is acquired.
- A run-time error occurs.

A stop event executes the callback function specified for `StopFcn`. Under most circumstances, the callback function is not guaranteed to complete execution until sometime after the device object and hardware device stop running, and the `Running` property is set to `Off`.

**Timer Event**
A timer event is generated whenever the time specified by the `TimerPeriod` property passes. This event executes the callback function specified for `TimerFcn`. Time is measured relative to when the device object starts running.

Some timer events might not be processed if your system is significantly slowed or if the `TimerPeriod` value is too small. For example, a common application for timer events is to display data. However, because displaying data is a CPU-intensive task, some of these events might be dropped. To guarantee that events are not dropped, use the `SamplesAcquiredFcn` property.

**Trigger Event**
A trigger event is generated immediately after a trigger occurs. This event executes the callback function specified for the `TriggerFcn` property. Under most circumstances, the callback function is not guaranteed to complete execution until sometime after `Logging` is set to `On`.

**Recording and Retrieving Event Information**
While the analog input object is running, certain information is automatically recorded in the `EventLog` property for some of the event types listed in the preceding section. `EventLog` is a structure that contains two fields: `Type` and `Data`. The `Type` field contains the event type. The `Data` field contains event-specific information. Events are recorded in the order in which they
occur. The first EventLog entry reflects the first event recorded, the second EventLog entry reflects the second event recorded, and so on.

The event types recorded in EventLog for analog input objects, as well as the values for the Type and Data fields, are given below.

**Table 5-6  Analog Input Event Information Stored in EventLog**

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Type Field Value</th>
<th>Data Field Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data missed</td>
<td>'DataMissed'</td>
<td>AbsTime</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RelSample</td>
</tr>
<tr>
<td>Input overrange</td>
<td>'OverRange'</td>
<td>AbsTime</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RelSample</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Channel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OverRange</td>
</tr>
<tr>
<td>Run-time error</td>
<td>'Error'</td>
<td>AbsTime</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RelSample</td>
</tr>
<tr>
<td></td>
<td></td>
<td>String</td>
</tr>
<tr>
<td>Start</td>
<td>'Start'</td>
<td>AbsTime</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RelSample</td>
</tr>
<tr>
<td>Stop</td>
<td>'Stop'</td>
<td>AbsTime</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RelSample</td>
</tr>
<tr>
<td>Trigger</td>
<td>'Trigger'</td>
<td>AbsTime</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RelSample</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Channel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Trigger</td>
</tr>
</tbody>
</table>

Samples acquired events and timer events are not stored in EventLog.
Note Unless a run-time error occurs, EventLog records a start event, trigger event, and stop event for each data acquisition session.

The Data field values are described below.

The AbsTime Field
AbsTime is used by the run-time error, start, stop, and trigger events to indicate the absolute time the event occurred. The absolute time is returned using the MATLAB clock format.

day-month-year hour:minute:second

The Channel Field
Channel is used by the input overrange event and the trigger event. For the input overrange event, Channel indicates the index number of the input channel that experienced an overrange signal. For the trigger event, Channel indicates the index number for each input channel serving as a trigger source.

The OverRange Field
OverRange is used by the input overrange event, and can be On or Off. If OverRange is On, then the input channel experienced an overrange signal. If OverRange is Off, then the input channel no longer experienced an overrange signal.

The RelSample Field
RelSample is used by all events stored in EventLog to indicate the sample number that was acquired when the event occurred. RelSample is 0 for the start event and for the first trigger event regardless of the trigger type. RelSample is NaN for any event that occurs before the first trigger executes.

The String Field
String is used by the run-time error event to store the descriptive message that is generated when a run-time error occurs. This message is also displayed at the MATLAB command line.
The Trigger Field

Trigger is used by the trigger event to indicate the trigger number. For example, if three trigger events occur, then Trigger is 3 for the third trigger event. The total number of triggers executed is given by the TriggersExecuted property.

Example: Retrieving Event Information

Suppose you want to examine the events logged for the example given by “Example: Voice Activation Using a Software Trigger” on page 5-23. You can do this by accessing the EventLog property.

```matlab
events = AIVoice.EventLog
events =
3x1 struct array with fields:
  Type
  Data
```

By examining the contents of the Type field, you can list the events that occurred while AIVoice was running.

```matlab
{events.Type}
an =
    'Start'    'Trigger'    'Stop'
```

To display information about the trigger event, you must access the Data field, which stores the absolute time the trigger occurred, the number of samples acquired when the trigger occurred, the index of the trigger channel, and the trigger number.

```matlab
trigdata = events(2).Data
trigdata =
    AbsTime: [1999 4 15 18 12 5.8615]
    RelSample: 0
    Channel: 1
    Trigger: 1
```
You can display a summary of the event log with the `showdaqevents` function. For example, to display a summary of the second event contained by the structure `events`:

```matlab
showdaqevents(events,2)
2 Trigger#1 ( 18:12:05, 0 ) Channel: 1
```

Alternatively, you can display event summary information via the Workspace browser by right-clicking the device object and selecting **Explore > Show DAQ Events** from the context menu.

### Creating and Executing Callback Functions

When using callback functions, you should be aware of these execution rules:

- Callback functions execute in the order in which they are issued.
- All callback functions except those associated with timer events are guaranteed to execute.
- Callback function execution might be delayed if the callback involves a CPU-intensive task such as updating a figure.

You specify the callback function to be executed when a specific event type occurs by including the name of the M-file as the value for the associated callback property. You can specify the callback function as a function handle or as a string cell array element. Function handles are described in the MATLAB `function_handle` reference pages. Note that if you are executing a local callback function from within an M-file, then you must specify the callback as a function handle.

For example, to execute the callback function `mycallback` for the analog input object `ai` every time 1000 samples are acquired:

```matlab
ai.SamplesAcquiredFcnCount = 1000;
ai.SamplesAcquiredFcn = @mycallback;
```

Alternatively, you can specify the callback function as a cell array.

```matlab
ai.SamplesAcquiredFcn = {'mycallback'};
```
M-file callback functions require at least two input arguments. The first argument is the device object. The second argument is a variable that captures the event information given in Table 5-6, Analog Input Event Information Stored in EventLog. This event information pertains only to the event that caused the callback function to execute. The function header for mycallback is shown below.

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

You pass additional parameters to the callback function by including both the callback function and the parameters as elements of a cell array. For example, to pass the MATLAB variable `time` to mycallback:

```matlab
time = datestr(now,0);
ai.SamplesAcquiredFcnCount = 1000;
ai.SamplesAcquiredFcn = {@mycallback,time};
```

Alternatively, you can specify mycallback as a string in the cell array.

```matlab
ai.SamplesAcquiredFcn = {'mycallback',time};
```

The corresponding function header is

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

If you pass additional parameters to the callback function, then they must be included in the function header after the two required arguments.

---

**Note** You can also specify the callback function as a string. In this case, the callback is evaluated in the MATLAB workspace and no requirements are made on the input arguments of the callback function.

---

**Specifying a Toolbox Function as a Callback**

In addition to specifying your own callback function, you can specify the start, stop, or trigger toolbox functions as callbacks. For example, to configure ai to stop running when an overrange condition occurs:

```matlab
ai.InputOverRangeFcn = @stop;
```
Examples: Using Callback Properties and Functions

This section provides examples that show you how to create callback functions and configure callback properties.

Displaying Event Information with a Callback Function

This example illustrates how callback functions allow you to easily display event information. The example uses daqcallback to display information for trigger, run-time error, and stop events. The default SampleRate and SamplesPerTrigger values are used, which results in a 1-second acquisition for each trigger executed.

You can run this example by typing daqdoc5_6 at the MATLAB command line.

1 Create a device object — Create the analog input object AI for a sound card. The installed adaptors and hardware IDs are found with daqhwinfo.

   AI = analoginput('winsound');
   %AI = analoginput('nidaq','Dev1');
   %AI = analoginput('mcc',1);

2 Add channels — Add one hardware channel to AI.

   chan = addchannel(AI,1);
   %chan = addchannel(AI,0); % For NI and MCC

3 Configure property values — Repeat the trigger three times, find the time for the acquisition to complete, and define daqcallback as the M-file to execute when a trigger, run-time error, or stop event occurs.

   set(AI,'TriggerRepeat',3)
   time = (AI.SamplesPerTrig/AI.SampleRate)*(AI.TriggerRepeat+1);
   set(AI,'TriggerFcn',@daqcallback)
   set(AI,'RuntimeErrorFcn',@daqcallback)
   set(AI,'StopFcn',@daqcallback)

4 Acquire data — Start AI and wait for it to stop running. The wait function blocks the MATLAB command line, and waits for AI to stop running.

   start(AI)
   wait(AI,time)
5 **Clean up** — When you no longer need AI, you should remove it from memory and from the MATLAB workspace.

```matlab
delete(AI)
clear AI
```

**Passing Additional Parameters to a Callback Function**

This example illustrates how additional arguments are passed to the callback function. Timer events are generated every 0.5 second to display data using the local callback function `daqdoc5_7plot` (not shown below).

You can run this example by typing `daqdoc5_7` at the MATLAB command line.

1 **Create a device object** — Create the analog input object AI for a sound card. The installed adaptors and hardware IDs are found with `daqhwinfo`.

```matlab
AI = analoginput('winsound');
%AI = analoginput('nidaq','Dev1');
%AI = analoginput('mcc',1);
```

2 **Add channels** — Add one hardware channel to AI.

```matlab
chan = addchannel(AI,1);
%chan = addchannel(AI,0); % For NI and MCC
```

3 **Configure property values** — Define a 10-second acquisition and execute the M-file `daqdoc5_7plot` every 0.5 seconds. Note that the variables bsize, P, and T are passed to the callback function.

```matlab
duration = 10; % Ten second duration
set(AI,'SampleRate',22050)
ActualRate = get(AI,'SampleRate');
set(AI,'SamplesPerTrigger',duration*ActualRate)
set(AI,'TimerPeriod',0.5)
bsize = (AI.SampleRate)*(AI.TimerPeriod);
figure
P = plot(zeros(bsize,1));
T = title(['Number of callback function calls: ', num2str(0)]);
xlabel('Samples'), ylabel('Signal (Volts)')
grid on
set(gcf,'doublebuffer','on')
set(AI,'TimerFcn',{@daqdoc5_7plot,bsize,P,T})

4 Acquire data — Start AI. The drawnow command in daqdoc5_7plot forces MATLAB to update the display. The wait function blocks the MATLAB command line, and waits for AI to stop running.

    start(AI)
    wait(AI,duration)

5 Clean up — When you no longer need AI, you should remove it from memory and from the MATLAB workspace.

    delete(AI)
    clear AI
### Linearly Scaling the Data: Engineering Units

#### Analog Input Engineering Units Properties

Data Acquisition Toolbox provides you with a way to linearly scale analog input signals from your sensor. You can associate this scaling with specific engineering units, such as volts or Newtons, that you might want to apply to your data. When specifying engineering units, there are three important considerations:

- The expected data range produced by your sensor. This range depends on the physical phenomena you are measuring and the maximum output range of the sensor.
- The range of your analog input hardware. For many devices, this range is specified by the gain and polarity. You can return valid input ranges with the `daqhwinfo` function.
- The engineering units associated with your acquisition. By default, most analog input hardware converts data to voltage values. However, after the data is digitized, you might want to define a linear scaling that represents specific engineering units when data is returned to MATLAB.

The properties associated with engineering units and linearly scaling acquired data are as follows:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SensorRange</td>
<td>Specify the range of data you expect from your sensor.</td>
</tr>
<tr>
<td>InputRange</td>
<td>Specify the range of the analog input subsystem.</td>
</tr>
</tbody>
</table>
(Continued)

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Units</td>
<td>Specify the engineering units label.</td>
</tr>
<tr>
<td>UnitsRange</td>
<td>Specify the range of data as engineering units.</td>
</tr>
</tbody>
</table>

**Note** If supported by the hardware, you can set the engineering units properties on a per-channel basis. Therefore, you can configure different engineering unit conversions for each hardware channel.

Linearly scaled acquired data is given by the formula

\[
\text{scaled value} = \frac{(A/D \text{ value})(\text{units range})}{(\text{sensor range})}
\]

**Note** The above formula assumes you are using symmetric units range and sensor range values, and represents the simplest scenario. If your units range or sensor range values are asymmetric, the formula includes the appropriate offset.

The A/D value is constrained by the InputRange property, which reflects the gain and polarity of your hardware channels, and is usually returned as a voltage value. You should choose an input range that utilizes the maximum dynamic range of your A/D subsystem. The best input range is the one that most closely encompasses the expected sensor range. If the sensor signal is larger than the input range, then the hardware will usually clip (saturate) the signal.

The units range is given by the UnitsRange property, while the sensor range is given by the SensorRange property. SensorRange is specified as a voltage value, while UnitsRange is specified as an engineering unit such as Newtons or g’s (1 g = 9.80 m/s²). These property values control the scaling of data when it is extracted from the engine with the getdata function. You can find the appropriate units range and sensor range from your sensor’s specification sheet.
For example, suppose SensorRange is \([-1, 1]\) and UnitsRange is \([-10, 10]\). If an A/D value is 2.5, then the scaled value is \((2.5)(20/2)\) or 25, in the appropriate units.

**Example: Performing a Linear Conversion**

This example illustrates how to configure the engineering units properties for an analog input object connected to a National Instruments PCI-6024E board.

An accelerometer is connected to a device which is undergoing a vibration test. Your job is to measure the acceleration and the frequency components of the device while it is vibrating. The accelerometer has a range of \(\pm 50\) g, a voltage sensitivity of 99.7 mV/g, and a resolution of 0.00016 g.

The accelerometer signal is input to a Tektronix TDS 210 digital oscilloscope and to channel 0 of the data acquisition board. By observing the signal on the scope, the maximum expected range of data from the sensor is \(\pm 200\) mV, which corresponds to approximately \(\pm 2\) g. Given this constraint, you should configure the board’s input range to \(\pm 500\) mV, which is the closest input range that encompasses the expected data range.

You can run this example by typing daqdoc5_8 at the MATLAB command line.

1. **Create a device object** — Create the analog input object AI for a National Instruments board. The installed adaptors and hardware IDs are found with daqhwinfo.

   ```matlab
   AI = analoginput('nidaq','Dev1');
   ```

2. **Add channels** — Add one hardware channel to AI.

   ```matlab
   chan = addchannel(AI,0);
   ```

3. **Configure property values** — Configure the sampling rate to 200 kHz and define a two-second acquisition.

   ```matlab
   duration = 2;
   ActualRate = setverify(AI,'SampleRate',200000);
   set(AI,'SamplesPerTrigger',duration*ActualRate)
   ```
Configure the engineering units properties. This example assumes you are using a National Instruments PCI-6024E board or an equivalent hardware device. SensorRange is set to the maximum accelerometer range in volts, and UnitsRange is set to the corresponding range in g's. InputRange is set to the value that most closely encompasses the expected data range of ±200 mV.

```matlab
set(chan,'SensorRange',[-5 5])
set(chan,'InputRange',[-0.5 0.5])
set(chan,'UnitsRange',[-50 50])
set(chan,'Units','g (1 g = 9.80 m/s/s)')
```

**4 Acquire data** — Start the acquisition and wait before acquiring data.

```matlab
start(AI)
wait(AI,duration+1)
```

Extract and plot all the acquired data.

```matlab
data = getdata(AI);
subplot(2,1,1),plot(data)
```

Calculate and display the frequency information.

```matlab
Fs = ActualRate;
blocksize = duration*ActualRate;
[f,mag]= daqdocfft(data,Fs,blocksize);
subplot(2,1,2),plot(f,mag)
```

**5 Clean up** — When you no longer need AI, you should remove it from memory and from the MATLAB workspace.

```matlab
delete(AI)
clear AI
```

### Linear Conversion with Asymmetric Data

The properties related to engineering units provide a way for Data Acquisition Toolbox to convert raw measurement data into its original values and units.

SensorRange is the output voltage range that your sensor is capable of producing.
UnitsRange is the range of real-world values (physical phenomena) that your sensor is measuring.

In many cases, it is appropriate to set InputRange, SensorRange, and UnitsRange to the same values. However, if there are significant differences in these ranges or the data is not symmetric, then using different values for these properties might be appropriate, as illustrated in the following scenario.

Suppose you have a speed sensor that generates 5 volts to 7 volts according to the detected speed, so you set SensorRange to [5 7]. When the sensor detects a speed of 0 m/s it generates a 5-volt signal; when it senses 20 m/s, it generates a 7-volt signal; so you set UnitsRange to [0 20].

For example, when the sensor transmits 6 volts, Data Acquisition Toolbox converts this value according to the formula

\[
\text{scaled value} = (\text{Sensor output} - \text{Offset}) \times \frac{\text{UnitsRange}}{\text{SensorRange}}
\]

\[
\text{scaled value} = (6 \text{ V} - 5 \text{ V}) \times \frac{20 - 0}{7 - 5}
\]

\[
\text{scaled value} = (1) \times \frac{20}{2}
\]

\[
\text{scaled value} = 10 \text{ m/s}
\]

For a sensor output value of 6.5 V, scaled value = (6.5 - 5) x (20)/(2) = 15 m/s; and so on, as shown in the following graph.
Linear Data Conversion

Scaled Value (m/s)

Sensor Output (V)
Analog output subsystems convert digital data stored on your computer to a real-world analog signal. Typical plug-in acquisition boards offer two output channels with 12 bits of resolution, with special hardware available to support multiple channel analog output operations. Data Acquisition Toolbox provides access to analog output subsystems through an analog output object.

The purpose of this chapter is to show you how to perform data acquisition tasks using your analog output hardware. The sections are as follows.

- **Getting Started with Analog Output** (p. 6-2)
  - Perform basic tasks with your analog output hardware

- **Managing Output Data** (p. 6-16)
  - Queue data in memory for eventual output to the hardware

- **Configuring Analog Output Triggers** (p. 6-20)
  - Initiate the output of queued data to the hardware

- **Events and Callbacks** (p. 6-26)
  - Enhance your analog output session using events and callbacks

- **Linearly Scaling the Data** (p. 6-35)
  - Configure engineering units properties so that output data is linearly scaled

- **Starting Multiple Device Objects** (p. 6-38)
  - Simultaneously use your hardware’s analog output and analog input subsystems
Getting Started with Analog Output

In this section...

“Creating an Analog Output Object” on page 6-2
“Adding Channels to an Analog Output Object” on page 6-3
“Configuring Analog Output Properties” on page 6-5
“Outputting Data” on page 6-7
“Analog Output Examples” on page 6-9
“Evaluating the Analog Output Object Status” on page 6-12

Creating an Analog Output Object

You must create an Analog Output object with which you can use the Data Acquisition Toolbox to perform basic tasks with your analog output (AO) hardware. This section describes the important properties and functions required for an analog output data acquisition session, and also provides several device-specific examples and ways to evaluate the status of the analog output object.

You create an analog output object with the `analogoutput` function. `analogoutput` accepts the adaptor name and the hardware device ID as input arguments. For a list of supported adaptors, refer to . The device ID refers to the number associated with your board when it is installed. (When using NI-DAQmx, this is usually a string such as ’Dev1’.) Some vendors refer to the device ID as the device number or the board number. The device ID is optional for sound cards with an ID of 0. Use the `daqhwinfo` function to determine the available adaptors and device IDs.

Each analog output object is associated with one board and one analog output subsystem. For example, to create an analog output object associated with a National Instruments board with device ID 1:

```matlab
ao = analogoutput('nidaq','Dev1');
```

The analog output object `ao` now exists in the MATLAB workspace. You can display the class of `ao` with the `whos` command.
Once the analog output object is created, the properties listed below are automatically assigned values. These general purpose properties provide descriptive information about the object based on its class type and adaptor.

### Table 6-1 Descriptive Analog Output Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Specify a descriptive name for the device object.</td>
</tr>
<tr>
<td>Type</td>
<td>Indicate the device object type.</td>
</tr>
</tbody>
</table>

You can display the values of these properties for `ao` with the `get` function.

```matlab
get(ao,{'Name','Type'})
ans =
    'nidaq1-AO'    'Analog Output'
```

## Adding Channels to an Analog Output Object

After creating the analog output object, you must add hardware channels to it. As shown by the figure in “Hardware Channels or Lines” on page 3-9, you can think of a device object as a container for channels. The collection of channels contained by the device object is referred to as a *channel group*. As described in “Mapping Hardware Channel IDs to MATLAB Indices” on page 3-10, a channel group consists of a mapping between hardware channel IDs and MATLAB indices.

When adding channels to an analog output object, you must follow these rules:

- The channels must reside on the same hardware device. You cannot add channels from different devices, or from different subsystems on the same device.
- The channels must be sampled at the same rate.
You add channels to an analog output object with the `addchannel` function. `addchannel` requires the device object and at least one hardware channel ID as input arguments. You can optionally specify MATLAB indices, descriptive channel names, and an output argument. For example, to add two hardware channels to the device object `ao` created in the preceding section:

```matlab
chans = addchannel(ao,0:1);
```

The output argument `chans` is a *channel object* that reflects the channel array contained by `ao`. You can display the class of `chans` with the `whos` command.

```matlab
whos chans
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>chans</td>
<td>2x1</td>
<td>512</td>
<td>aochannel object</td>
</tr>
</tbody>
</table>

Grand total is 7 elements using 512 bytes

You can use `chans` to easily access channels. For example, you can easily configure or return property values for one or more channels. As described in “Referencing Individual Hardware Channels” on page 4-7, you can also access channels with the `Channel` property.

Once you add channels to an analog output object, the properties listed below are automatically assigned values. These properties provide descriptive information about the channels based on their class type and ID.

### Table 6-2 Descriptive Analog Output Channel Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HwChannel</td>
<td>Specify the hardware channel ID.</td>
</tr>
<tr>
<td>Index</td>
<td>Indicate the MATLAB index of a hardware channel.</td>
</tr>
<tr>
<td>Parent</td>
<td>Indicate the parent (device object) of a channel.</td>
</tr>
<tr>
<td>Type</td>
<td>Indicate a channel.</td>
</tr>
</tbody>
</table>

You can display the values of these properties for `chans` with the `get` function.

```matlab
get(chans,{'HwChannel','Index','Parent','Type'})
ans =
```
To reference individual channels, you must specify either MATLAB indices or descriptive channel names. Refer to “Referencing Individual Hardware Channels” on page 4-7 for more information.

**Configuring Analog Output Properties**

After hardware channels are added to the analog output object, you should configure property values. As described in “Configuring and Returning Properties” on page 3-13, Data Acquisition Toolbox supports two basic types of properties for analog output objects: common properties and channel properties. Common properties apply to all channels contained by the device object while channel properties apply to individual channels.

The properties you configure depend on your particular analog output application. For many common applications, there is a small group of properties related to the basic setup that you will typically use. These basic setup properties control the sampling rate and define the trigger type. Analog output properties related to the basic setup are given below.

**Table 6-3 Analog Output Basic Setup Properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SampleRate</td>
<td>Specify the per-channel rate at which digital data is converted to analog data.</td>
</tr>
<tr>
<td>TriggerType</td>
<td>Specify the type of trigger to execute.</td>
</tr>
</tbody>
</table>

**Setting the Sampling Rate**

You control the rate at which an analog output subsystem converts digital data to analog data is controlled with the SampleRate property. SampleRate must be specified as samples per second. For example, to set the sampling rate for each channel of your National Instruments board to 100,000 samples per second (100 kHz):

```matlab
ao = analogoutput('nidaq','Dev1');
```
addchannel(ao,0:1);
set(ao,'SampleRate',100000)

Data acquisition boards typically have predefined sampling rates that you can set. If you specify a sampling rate that does not match one of these predefined values, there are two possibilities:

- If the rate is within the range of valid values, then the engine automatically selects a valid sampling rate. The rules governing this selection process are described in the SampleRate reference pages.
- If the rate is outside the range of valid values, then an error is returned.

**Note** For some sound cards, you can set the sampling rate to any value between the minimum and maximum values defined by the hardware. You can enable this feature with the StandardSampleRates property. Refer to Chapter 16, “Device-Specific Properties — Alphabetical List” for more information.

Most analog output subsystems allow simultaneous sampling of channels. Therefore, the maximum sampling rate for each channel is given by the maximum board rate.

After setting a value for SampleRate, you should find out the actual rate set by the engine.

```
ActualRate = get(ao,'SampleRate');
```

Alternatively, you can use the setverify function, which sets a property value and returns the actual value set.

```
ActualRate = setverify(ao,'SampleRate',100000);
```

You can find the range of valid sampling rates for your hardware with the propinfo function.

```
ValidRates = propinfo(ao,'SampleRate');
ValidRates.ConstraintValue
ans =
     1.0e+005 *
     0.0000   2.0000
```
Defining a Trigger

For analog output objects, a trigger is defined as an event that initiates the output of data from the engine to the analog output hardware.

Defining a trigger for an analog output object involves specifying the trigger type. Trigger types are specified with the `TriggerType` property. The valid `TriggerType` values that are supported for all hardware are given below.

**Table 6-4 Analog Output TriggerType Property Values**

<table>
<thead>
<tr>
<th>TriggerType Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Immediate}</td>
<td>The trigger occurs just after you issue the start function.</td>
</tr>
<tr>
<td>Manual</td>
<td>The trigger occurs just after you manually issue the trigger function.</td>
</tr>
</tbody>
</table>

Most devices have hardware-specific trigger types, which are available to you through the `TriggerType` property. For example, to see all the trigger types (including hardware-specific trigger types) for the analog output object `ao` created in the preceding section:

```matlab
set(ao,'TriggerType')
[] Manual | {Immediate} | HwDigital |
```

This information tells you that the National Instruments board also supports a hardware digital trigger. For a description of device-specific trigger types, refer to “Device-Specific Hardware Triggers” on page 6-24, or the `TriggerType` reference pages.

Outputting Data

After you configure the analog output object, you can output data. Outputting data involves these three steps:

1. Queuing data
2. Starting the analog output object
3 Stopping the analog output object

Queuing Data in the Engine
Before you can start the device object, data must be queued in the engine. Data is queued in the engine with the `putdata` function. For example, to queue one second of data for each channel contained by the analog output object `ao`:

```matlab
ao = analogoutput('winsound');
addchannel(ao,1:2);
data = sin(linspace(0,2*pi,8000))';
putdata(ao,[data data])
```

`putdata` is a blocking function, and will not return execution control to MATLAB until the specified data is queued. `putdata` is described in detail in “Managing Output Data” on page 6-16 and in Chapter 12, “Functions — Alphabetical List”.

Starting the Analog Output Object
You start an analog output object with the `start` function. For example, to start the analog output object `ao`:

```matlab
start(ao)
```

After `start` is issued, the Running property is automatically set to On, and both the device object and hardware device execute according to the configured and default property values. While the device object is running, you can continue to queue data.

However, running does not necessarily mean that data is being output from the engine to the analog output hardware. For that to occur, a trigger must execute. When the trigger executes, the Sending property is automatically set to On. Analog output triggers are described on “Defining a Trigger” on page 6-7 and “Configuring Analog Output Triggers” on page 6-20.

Stopping the Analog Output Object
An analog output object can stop under one of these conditions:

- You issue the stop function.
• The queued data is output.
• A run-time hardware error occurs.
• A timeout occurs.

When the device object stops, the Running and Sending properties are automatically set to Off. At this point, you can reconfigure the device object or immediately queue more data, and issue another start command using the current configuration.

**Analog Output Examples**

This section illustrates how to perform basic data acquisition tasks using analog output subsystems and Data Acquisition Toolbox. For most data acquisition applications using analog output subsystems, you must follow these basic steps:

1. Install and connect the components of your data acquisition hardware. At a minimum, this involves connecting an actuator to a plug-in or external data acquisition device.

2. Configure your data acquisition session. This involves creating a device object, adding channels, setting property values, and using specific functions to output data.

Simple data acquisition applications using a sound card and a National Instruments board are given below.

**Outputting Data with a Sound Card**

In this example, sine wave data is generated in MATLAB, output to the D/A converter on the sound card, and sent to a speaker. The setup is shown below.
You can run this example by typing daqdoc6_1 at the MATLAB command line.

1 **Create a device object** — Create the analog output object A0 for a sound card. The installed adaptors and hardware IDs are found with daqhwinfo.

   AO = analogoutput('winsound');

2 **Add channels** — Add one channel to A0.

   chan = addchannel(AO,1);

3 **Configure property values** — Define an output time of four seconds, assign values to the basic setup properties, generate data to be queued, and queue the data with one call to putdata.

   duration = 4;
   set(AO,'SampleRate',8000)
   set(AO,'TriggerType','Manual')
   ActualRate = get(AO,'SampleRate');
   len = ActualRate*duration;
   data = sin(linspace(0,2*pi*500,len))';
   putdata(AO,data)

4 **Output data** — Start A0, issue a manual trigger, and wait for the device object to stop running.

   start(AO)
   trigger(AO)
   wait(AO,5)

5 **Clean up** — When you no longer need A0, you should remove it from memory and from the MATLAB workspace.

   delete(A0)
   clear AO
Outputting Data with a National Instruments Board

In this example, sine wave data is generated in MATLAB, output to the D/A converter on a National Instruments board, and displayed with an oscilloscope. The setup is shown below.

You can run this example by typing daqdoc6_2 at the MATLAB command line.

1 **Create a device object** — Create the analog output object AO for a National Instruments board. The installed adaptors and hardware IDs are found with daqhwinfo.

   
   AO = analogoutput('nidaq','Dev1');

2 **Add channels** — Add one channel to AO.

   
   chan = addchannel(AO,0);

3 **Configure property values** — Define an output time of four seconds, assign values to the basic setup properties, generate data to be queued, and queue the data with one call to putdata.

   
   duration = 4; 
   set(AO,'SampleRate',10000) 
   set(AO,'TriggerType','Manual') 
   ActualRate = get(AO,'SampleRate'); 
   len = ActualRate*duration; 
   data = sin(linspace(0,2*pi*500,len))'; 
   putdata(AO,data)
4 **Output data** — Start AO, issue a manual trigger, and wait for the device object to stop running.

\[
\text{start}(\text{AO}) \\
\text{trigger}(\text{AO}) \\
\text{wait}(\text{AO}, 5)
\]

5 **Clean up** — When you no longer need AO, you should remove it from memory and from the MATLAB workspace.

\[
\text{delete}(\text{AO}) \\
\text{clear AO}
\]

**Evaluating the Analog Output Object Status**

You can evaluate the status of an analog output (AO) object by

- Returning the values of certain properties
- Invoking the display summary

**Status Properties**

The properties associated with the status of your analog output object allow you to evaluate

- If the device object is running
- If data is being output from the engine
- How much data is queued in the engine
- How much data has been output from the engine

These properties are given below.

**Table 6-5  Analog Output Status Properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running</td>
<td>Indicate if the device object is running.</td>
</tr>
<tr>
<td>SamplesAvailable</td>
<td>Indicate the number of samples available per channel in the engine.</td>
</tr>
</tbody>
</table>
Table 6-5 Analog Output Status Properties (Continued)

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SamplesOutput</td>
<td>Indicate the number of samples output per channel from the engine.</td>
</tr>
<tr>
<td>Sending</td>
<td>Indicate if data is being sent (output) to the hardware device.</td>
</tr>
</tbody>
</table>

When data is queued in the engine, SamplesAvailable is updated to reflect the total number of samples per channel that was queued. When start is issued, Running is automatically set to On.

When the trigger executes, Sending is automatically set to On and SamplesOutput keeps a running count of the total number of samples per channel output from the engine to the hardware. Additionally, SamplesAvailable tells you how many samples per channel are still queued in the engine and ready to be output to the hardware.

When all the queued data is output from the engine, both Running and Sending are automatically set to Off, SamplesAvailable is 0, and SamplesOutput reflects the total number of samples per channel that was output.

The Display Summary
You can invoke the display summary by typing an AO object or a channel object at the MATLAB command line, or by excluding the semicolon when

- Creating an AO object
- Adding channels
- Configuring property values using the dot notation

You can also display summary information via the Workspace browser by right-clicking a toolbox object and selecting Explore > Display Summary from the context menu.

The information displayed reflects many of the basic setup properties described in “Configuring Analog Output Properties” on page 6-5, and is
designed so you can quickly evaluate the status of your data acquisition session. The display is divided into two main sections: general summary information and channel summary information.

**General Summary Information**

The general display summary includes the device object type and the hardware device name, followed by this information:

- Output parameters — The sampling rate
- Trigger parameters — The trigger type
- The engine status
  - Whether the engine is sending data, waiting to start, or waiting to trigger
  - The total time required to output the queued data
  - The number of samples queued by putdata
  - The number of samples sent to the hardware device

**Channel Summary Information**

The channel display summary includes property values associated with

- The hardware channel mapping
- The channel name
- The engineering units
The display summary shown below is for the example given in “Outputting Data with a Sound Card” on page 6-9 prior to issuing the start function.

Display Summary of Analog Output (AO) Object Using 'AudioPCI Playback'.

Output Parameters: 8000 samples per second on each channel.

Trigger Parameters: 1 'Immediate' trigger on START.

Engine status: Waiting for START.
0 total sec. of data currently queued for START
0 samples currently queued by PUTDATA.
0 samples sent to output device since START.

AO object contains channel(s):

<table>
<thead>
<tr>
<th>Index</th>
<th>ChannelName</th>
<th>HwChannel</th>
<th>OutputRange</th>
<th>UnitsRange</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>'Mono'</td>
<td>1</td>
<td>[-1 1]</td>
<td>[-1 1]</td>
<td>'Volts'</td>
</tr>
</tbody>
</table>

You can use the Channel property to display only the channel summary information.

AO.Channel
Managing Output Data

<table>
<thead>
<tr>
<th>In this section...</th>
</tr>
</thead>
<tbody>
<tr>
<td>“The Analog Output Subsystem” on page 6-16</td>
</tr>
<tr>
<td>“Queuing Data with putdata” on page 6-16</td>
</tr>
<tr>
<td>“Example: Queuing Data with putdata” on page 6-18</td>
</tr>
</tbody>
</table>

The Analog Output Subsystem

At the core of any analog output application lies the data you want to send from a computer to an output device such as an actuator. The role of the analog output subsystem is to convert digitized data to analog data for subsequent output.

Before you can output data to the analog output subsystem, it must be queued in the engine. Queuing data is managed with the putdata function. In addition to this function, there are several properties associated with managing output data. These properties are given below.

Table 6-6 Analog Output Data Management Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxSamplesQueued</td>
<td>Indicate the maximum number of samples that can be queued in the engine.</td>
</tr>
<tr>
<td>RepeatOutput</td>
<td>Specify the number of additional times queued data is output.</td>
</tr>
<tr>
<td>Timeout</td>
<td>Specify an additional waiting time to queue data.</td>
</tr>
</tbody>
</table>

Queuing Data with putdata

Before data can be sent to the analog output hardware, you must queue it in the engine. Queuing data is managed with the putdata function. One column of data is required for each channel contained by the analog output object. For example, to queue one second of data for each channel contained by the analog output object ao:
ao = analogoutput('winsound');
addchannel(ao,1:2);
data = sin(linspace(0,2*pi*500,8000))';
putdata(ao,[data data])

A data source consisting of m samples and n channels is illustrated below.

\[
\begin{bmatrix}
c_{11} & d_{12} & d_{13} \\
c_{21} & d_{22} & d_{23} \\
c_{31} & d_{32} & \ldots & d_{3n} \\
\vdots & \vdots & \ddots & \vdots \\
c_{m1} & d_{m2} & d_{mn}
\end{bmatrix}
\]

Data source. Each column represents a separate output channel.

**Rules for Using putdata**

Using putdata to queue data in the engine follows these rules:

- You must queue data in the engine before starting the analog output object.
- If the value of the RepeatOutput property is greater than 0, then all queued data is automatically requeued until the RepeatOutput value is reached. You must configure RepeatOutput before start is issued.
- While the analog output object is running, you can continue to queue data unless RepeatOutput is greater than 0.
- You can queue data in the engine until the value specified by the MaxSamplesQueued property is reached, or the limitations of your hardware or computer are reached.

**Rules for Queuing Data**

Data to be queued in the engine follows these rules:

- Data is output as soon as a trigger occurs.
- An error is returned if a NaN is included in the data stream.
- You can use the native data type of the hardware.
- If the data is not within the range of the `UnitsRange` property, then it is clipped to the maximum or minimum value specified by `UnitsRange`. Refer to “Linearly Scaling the Data” on page 6-35 for more information about clipping.

**Example: Queuing Data with `putdata`**

This example illustrates how you can use `putdata` to queue 8000 samples, and then output the data a total of five times using the `RepeatOutput` property.

You can run this example by typing `daqdoc6_3` at the MATLAB command line.

1 **Create a device object** — Create the analog output object `AO` for a sound card. The installed adaptors and hardware IDs are found with `daqhwinfo`.

   ```matlab
   AO = analogoutput('winsound');
   %AO = analogoutput('nidaq','Dev1');
   %AO = analogoutput('mcc',1);
   ```

2 **Add channels** — Add one channel to `AO`.

   ```matlab
   chans = addchannel(AO,1);
   %chans = addchannel(AO,0); % For NI and MCC
   ```

3 **Configure property values** — Define an output time of one second, assign values to the basic setup properties, generate data to be queued, and issue two `putdata` calls. Because the queued data is repeated four times and two `putdata` calls are issued, a total of 10 seconds of data is output.

   ```matlab
   duration = 1;
   set(AO,'SampleRate',8000)
   ActualRate = get(AO,'SampleRate');
   len = ActualRate*duration;
   set(AO,'RepeatOutput',4)
   data = sin(linspace(0,2*pi*500,len))';
   putdata(AO,data)
   putdata(AO,data)
   ```

4 **Output data** — Start `AO` and wait for the device object to stop running.

   ```matlab
   start(AO)
   wait(AO,11)
   ```
5 **Clean up** — When you no longer need A0, you should remove it from memory and from the MATLAB workspace.

```matlab
delete(A0)
clear A0
```
Configuring Analog Output Triggers

In this section...

“Analog Output Trigger Properties” on page 6-20
“Defining a Trigger: Trigger Types” on page 6-21
“Executing the Trigger” on page 6-22
“How Many Triggers Occurred?” on page 6-22
“When Did the Trigger Occur?” on page 6-23
“Device-Specific Hardware Triggers” on page 6-24

Analog Output Trigger Properties

An analog output trigger is defined as an event that initiates the output of data. As shown in the figure below, when a trigger occurs, the Sending property is automatically set to On and queued data is output from the engine to the hardware subsystem.

Properties associated with analog output triggers are as follows:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitialTriggerTime</td>
<td>Indicate the absolute time of the first trigger.</td>
</tr>
<tr>
<td>TriggerFcn</td>
<td>Specify the M-file callback function to execute when a trigger occurs.</td>
</tr>
</tbody>
</table>
(Continued)

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TriggersExecuted</td>
<td>Indicate the number of triggers that execute.</td>
</tr>
<tr>
<td>TriggerType</td>
<td>Specify the type of trigger to execute.</td>
</tr>
</tbody>
</table>

Except for TriggerFcn, these trigger-related properties are discussed in the following sections. TriggerFcn is discussed in “Events and Callbacks” on page 6-26.

**Defining a Trigger: Trigger Types**

Defining a trigger for an analog output object involves specifying the trigger type with the TriggerType property. You can think of the trigger type as the source of the trigger. The analog output TriggerType values are given below.

**Table 6-7 Analog Output TriggerType Property Values**

<table>
<thead>
<tr>
<th>TriggerType Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Immediate}</td>
<td>The trigger occurs just after you issue the start function.</td>
</tr>
<tr>
<td>Manual</td>
<td>The trigger occurs just after you manually issue the trigger function.</td>
</tr>
</tbody>
</table>

Trigger types can be grouped into two main categories:

- Device-independent triggers
- Device-specific hardware triggers

The trigger types shown above are device-independent triggers because they are available for all supported hardware. For these trigger types, the callback that initiates the trigger event involves issuing a toolbox function (start or trigger). Conversely, device-specific hardware triggers depend on the specific hardware device you are using. For these trigger types, the callback that initiates the trigger event involves an external digital signal.
Device-specific hardware triggers for National Instruments and Agilent Technologies devices are discussed in “Device-Specific Hardware Triggers” on page 6-24. Device-independent triggers are discussed below.

**Immediate Trigger**
If TriggerType is Immediate (the default value), the trigger occurs immediately after the start function is issued. You can configure an analog output object for continuous output, by using an immediate trigger and setting RepeatOutput to inf.

**Manual Trigger**
If TriggerType is Manual, the trigger occurs immediately after the trigger function is issued.

**Executing the Trigger**
For an analog output trigger to occur, you must follow these steps:

1. Queue data in the engine.
2. Configure the appropriate trigger properties.
3. Issue the start function.
4. Issue the trigger function if TriggerType is Manual.

Once the trigger occurs, queued data is output to the hardware, and the device object stops executing when all the queued data is output.

**Note** Only one trigger event can occur for analog output objects.

**How Many Triggers Occurred?**
For analog output objects, only one trigger can occur. You can determine if the trigger event occurred by returning the value of the TriggersExecuted property. If TriggersExecuted is 0, then the trigger event did not occur. If TriggersExecuted is 1, then the trigger event occurred. Event information is
also recorded by the EventLog property. A convenient way to access event log information is with the showdaqevents function.

For example, suppose you create the analog output object ao for a sound card and add one channel to it. ao is configured to output 8,000 samples using the default sampling rate of 8000 Hz.

\[
\text{ao} = \text{analogoutput('winsound');}
\text{addchannel(ao,1);}
\text{data} = \sin(\text{linspace}(0,1,8000))';
\text{putdata(ao,data) start(ao)}
\]

TriggersExecuted returns the number of triggers executed.

\[
\text{ao.TriggersExecuted}
\text{ans = 1}
\]

You can use showdaqevents to return information for all events that occurred while ao was executing.

\[
\text{showdaqevents(ao)}
\begin{align*}
1 & \text{ Start} & (10:43:25, 0) \\
2 & \text{ Trigger} & (10:43:25, 0) \\
3 & \text{ Stop} & (10:43:26, 8000)
\end{align*}
\]

For more information about recording and retrieving event information, refer to “Recording and Retrieving Event Information” on page 6-29.

**When Did the Trigger Occur?**

You can return the absolute time of the trigger with the InitialTriggerTime property. Absolute time is returned as a clock vector in the form

\[[\text{year month day hour minute seconds}]\]

For example, the absolute time of the trigger event for the preceding example is

\[
\text{abstime} = \text{ao.InitialTriggerTime}
\]
abstime =
1.0e+003 *
 1.9990 0.0040 0.0170 0.0100 0.0430 0.0252

To convert the clock vector to a more convenient form, you can use the `sprintf` function.

```matlab
t = fix(abstime);
sprintf('%d:%d:%d', t(4),t(5),t(6))
ans =
10:43:25
```

As shown in the preceding section, you can also evaluate the absolute time of the trigger event with the `showdaqevents` function.

**Device-Specific Hardware Triggers**

Most data acquisition devices possess the ability to accept a hardware trigger. Hardware triggers are processed directly by the hardware and are typically transistor-transistor logic (TTL) signals. Hardware triggers are used when speed is required because a hardware device can process an input signal much faster than software.

The device-specific hardware triggers are presented to you as additional property values. Hardware triggers for National Instruments and Agilent Technologies devices are discussed below and in Chapter 14, “Base Properties — Alphabetical List”.

Note that the available hardware trigger support depends on the board you are using. Refer to your hardware documentation for detailed information about its triggering capabilities.

**National Instruments**

When using National Instruments hardware, there is an additional analog output trigger type available to you — digital triggering.

If `TriggerType` is set to `HwDigital`, the trigger is given by an external TTL signal that is input directly into the hardware device. The following example illustrates how to configure a hardware digital trigger.
Configuring Analog Output Triggers

```matlab
ao = analogoutput('nidaq','Dev1');
addchannel(ao,0:1);
set(ao,'TriggerType','HwDigital')
```

With this trigger configuration, `ao` will not start outputting data until the TTL signal is detected by the hardware on the appropriate pin.

The diagram below illustrates how you can connect a digital trigger signal to an MIO-16E Series board. PFI6/WFTRIG corresponds to pin 5.

![Diagram of MIO-16E Series board](image)

**Agilent Technologies**

When using Agilent Technologies hardware, there are additional analog output trigger types that you must be aware of: digital triggering on a positive edge and digital triggering on a negative edge.

If `TriggerType` is `HwDigitalPos`, the trigger source is the positive edge of a digital signal. If `TriggerType` is `HwDigitalNeg`, the trigger source is the negative edge of a digital signal.

In both cases, the digital signal is an external TTL signal that is input directly into the hardware device. The example below illustrates how to configure such a trigger.

```matlab
ao = analogoutput('hpe1432',8);
addchannel(ao,1);
set(ao,'TriggerType','HwDigitalPos')
```

With this trigger configuration, `ao` will not start outputting data until the TTL signal is detected by the hardware.
Events and Callbacks

In this section...

“Understanding Events and Callbacks” on page 6-26
“Event Types” on page 6-26
“Recording and Retrieving Event Information” on page 6-29
“Examples: Using Callback Properties and Callback Functions” on page 6-32

Understanding Events and Callbacks

You can enhance the power and flexibility of your analog output application by utilizing events. An event occurs at a particular time after a condition is met and might result in one or more callbacks.

While the analog output object is running, you can use events to display a message, display data, analyze data, and so on. Callbacks are controlled through callback properties and callback functions. All event types have an associated callback property. Callback functions are M-file functions that you construct to suit your specific data acquisition needs.

You execute a callback when a particular event occurs by specifying the name of the M-file callback function as the value for the associated callback property. Refer to “Creating and Executing Callback Functions” on page 5-52 to learn how to create callback functions. Note that daqcallback is the default value for some callback properties.

Event Types

The analog output event types and associated callback properties are described below.
### Table 6-8  Analog Output Callback Properties

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run-time error</td>
<td>RunTimeErrorFcn</td>
</tr>
<tr>
<td>Samples output</td>
<td>SamplesOutputFcn</td>
</tr>
<tr>
<td></td>
<td>SamplesOutputFcnCount</td>
</tr>
<tr>
<td>Start</td>
<td>StartFcn</td>
</tr>
<tr>
<td>Stop</td>
<td>StopFcn</td>
</tr>
<tr>
<td>Timer</td>
<td>TimerFcn</td>
</tr>
<tr>
<td></td>
<td>TimerPeriod</td>
</tr>
<tr>
<td>Trigger</td>
<td>TriggerFcn</td>
</tr>
</tbody>
</table>

**Run-time Error Event**

A run-time error event is generated immediately after a run-time error occurs. This event executes the callback function specified for RunTimeErrorFcn. Additionally, a toolbox error message is automatically displayed to the MATLAB workspace. If an error occurs that is not explicitly handled by the toolbox, then the hardware-specific error message is displayed.

The default value for RunTimeErrorFcn is daqcallback, which displays the event type, the time the event occurred, the device object name, and the error message.

Run-time errors include hardware errors and timeouts. Run-time errors do not include configuration errors such as setting an invalid property value.

**Samples Output Event**

A samples output event is generated immediately after the number of samples specified by the SamplesOutputFcnCount property is output for each channel group member. This event executes the callback function specified for SamplesOutputFcn.
**Start Event**

A start event is generated immediately after the start function is issued. This event executes the callback function specified for StartFcn. When the callback function has finished executing, Running is automatically set to On and the device object and hardware device begin executing. The device object is not started if an error occurs while executing the callback function.

**Stop Event**

A stop event is generated immediately after the device object and hardware device stop running. This occurs when

- The stop function is issued.
- The requested number of samples is output.
- A run-time error occurs.

A stop event executes the callback function specified for StopFcn. Under most circumstances, the callback function is not guaranteed to complete execution until sometime after the device object and hardware device stop running, and the Running property is set to Off.

**Timer Event**

A timer event is generated whenever the time specified by the TimerPeriod property passes. This event executes the callback function specified for TimerFcn. Time is measured relative to when the device object starts running.

Some timer events might not be processed if your system is significantly slowed or if the TimerPeriod value is too small. For example, a common application for timer events is to display data. However, because displaying data is a CPU-intensive task, some of these events might be dropped. To guarantee that events are not dropped, you can use the SamplesOutputFcn property.

**Trigger Event**

A trigger event is generated immediately after a trigger occurs. This event executes the callback function specified for TriggerFcn. Under most
circumstances, the callback function is not guaranteed to complete execution until sometime after Sending is set to On.

**Recording and Retrieving Event Information**

While the analog output object is running, certain information is automatically recorded in the EventLog property for some of the event types listed in the preceding section. EventLog is a structure that contains two fields: Type and Data. The Type field contains the event type. The Data field contains event-specific information. Events are recorded in the order in which they occur. The first EventLog entry reflects the first event recorded, the second EventLog entry reflects the second event recorded, and so on.

The event types recorded in EventLog for analog output objects, as well as the values for the Type and Data fields, are as follows:

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Type Field Value</th>
<th>Data Field Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run-time error</td>
<td>Error</td>
<td>AbsTime</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RelSample</td>
</tr>
<tr>
<td></td>
<td></td>
<td>String</td>
</tr>
<tr>
<td>Start</td>
<td>Start</td>
<td>AbsTime</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RelSample</td>
</tr>
<tr>
<td>Stop</td>
<td>Stop</td>
<td>AbsTime</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RelSample</td>
</tr>
<tr>
<td>Trigger</td>
<td>Trigger</td>
<td>AbsTime</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RelSample</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Channel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Trigger</td>
</tr>
</tbody>
</table>

Samples output events and timer events are not stored in EventLog.
Note Unless a run-time error occurs, EventLog records a start event, a trigger event, and stop event for each data acquisition session.

The Data field values are described below.

The AbsTime Field
AbsTime is used by all analog output events stored in EventLog to indicate the absolute time the event occurred. The absolute time is returned using the MATLAB clock format.

day-month-year hour:minute:second

The Channel Field
Channel is used by the input overrange event and the trigger event. For the input overrange event, Channel indicates the index number of the input channel that experienced an overrange signal. For the trigger event, Channel indicates the index number for each input channel serving as a trigger source.

The RelSample Field
RelSample is used by all events stored in EventLog to indicate the sample number that was output when the event occurred. RelSample is 0 for the start event and for the first trigger event regardless of the trigger type. RelSample is NaN for any event that occurs before the trigger executes.

The String Field
String is used by the run-time error event to store the descriptive message that is generated when a run-time error occurs. This message is also displayed at the MATLAB command line.

The Trigger Field
Trigger is used by the trigger event to indicate the trigger number. For example, if three trigger events occur, then Trigger is 3 for the third trigger event. The total number of triggers executed is given by the TriggersExecuted property.
Example: Retrieving Event Information
Suppose you want to examine the events logged for the example given by “Example: Queuing Data with putdata” on page 6-18. You can do this by accessing the EventLog property.

```matlab
events = AO.EventLog
events =
3x1 struct array with fields:
    Type
    Data
```

By examining the contents of the Type field, you can list the events that were recorded while AO was running.

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

To display information about the trigger event, you must access the Data field, which stores the absolute time the trigger occurred and the number of samples output when the trigger occurred.

```matlab
trigdata = events(2).Data
trigdata =
AbsTime: [1999 4 16 9 53 19.9508]
RelSample: 0
```

You can display a summary of the event log with the `showdaqevents` function. For example, to display a summary of the second event contained by the structure `events`:

```matlab
showdaqevents(events,2)
2 Trigger ( 09:53:19, 0 )
```

Alternatively, you can display event summary information via the Workspace browser by right-clicking the device object and selecting Explore > Show DAQ Events from the context menu.
Examples: Using Callback Properties and Callback Functions
Examples showing how to create callback functions and configure callback properties are given below.

Displaying the Number of Samples Output
This example illustrates how to generate samples output events. You can run this example by typing daqdoc6_4 at the MATLAB command line. The local callback function daqdoc6_4disp (not shown below) displays the number of events that were output from the engine whenever the samples output event occurred.

1 Create a device object — Create the analog output object AO for a sound card. The installed adaptors and hardware IDs are found with daqhwinfo.

\[
\begin{align*}
AO &= \text{analogoutput('winsound')} ; \\
%A0 &= \text{analogoutput('nidaq','Dev1')} ; \\
%A0 &= \text{analogoutput('mcc',1)} ;
\end{align*}
\]

2 Add channels — Add two channels to AO.

\[
\begin{align*}
\text{chans} &= \text{addchannel(AO,1:2)} ; \\
%\text{chans} &= \text{addchannel(AO,0:1)} ; \% \text{For NI and MCC}
\end{align*}
\]

3 Configure property values — Configure the trigger to repeat four times, specify daqdoc6_4disp as the M-file callback function to execute whenever 8000 samples are output, generate data to be queued, and queue the data with one call to putdata.

\[
\begin{align*}
\text{set(AO,'SampleRate',8000)} & \\
\text{ActualRate} &= \text{get(AO,'SampleRate')} ; \\
\text{set(AO,'RepeatOutput',4)} & \\
\text{set(AO,'SamplesOutputFcnCount',8000)} & \\
\text{freq} &= \text{get(AO,'SamplesOutputFcnCount')} ; \\
\text{set(AO,'SamplesOutputFcn',@daqdoc6_4disp)} & \\
\text{data} &= \text{sin(linspace(0,2*pi*500,3*freq))'} ; \\
\text{putdata(AO,[data data])}
\end{align*}
\]

4 Output data — Start AO. The wait function blocks the MATLAB command line, and waits for AO to stop running.
Events and Callbacks

5 Clean up — When you no longer need AO, you should remove it from memory and from the MATLAB workspace.

    delete(AO)
    clear AO

Displaying EventLog Information

This example illustrates how callback functions allow you to easily display information stored in the EventLog property. You can run this example by typing daqdoc6_5 at the MATLAB command line. The local callback function daqdoc6_5disp (not shown below) displays the absolute time and relative sample associated with the start, trigger, and stop events.

1 Create a device object — Create the analog output object AO for a sound card. The installed adaptors and hardware IDs are found with daqhwinfo.

    AO = analogoutput('winsound');
    %AO = analogoutput('nidaq','Dev1');
    %AO = analogoutput('mcc',1);

2 Add channels — Add one channel to AO.

    chan = addchannel(AO,1);
    %chan = addchannel(AO,0); % For NI and MCC

3 Configure property values — Specify daqdoc6_5disp as the M-file callback function to execute when the start, trigger, and stop events occur, generate data to be queued, and queue the data with one call to putdata.

    set(AO,'SampleRate',8000)
    ActualRate = get(AO,'SampleRate');
    set(AO,'StartFcn',@daqdoc6_5disp)
    set(AO,'TriggerFcn',@daqdoc6_5disp)
    set(AO,'StopFcn',@daqdoc6_5disp)
    data = sin(linspace(0,2*pi*500,ActualRate));
    data = [data data data];
    time = (length(data)/AO.SampleRate);
putdata(A0, data')

4 **Output data** — Start A0. The `wait` function blocks the MATLAB command line, and waits for A0 to stop running.

```matlab
start(A0)
wait(A0, 5)
```

5 **Clean up** — When you no longer need A0, you should remove it from memory and from the MATLAB workspace.

```matlab
delete(A0)
clear A0
```
**Linearly Scaling the Data**

<table>
<thead>
<tr>
<th>In this section...</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Engineering Units” on page 6-35</td>
</tr>
<tr>
<td>“Example: Performing a Linear Conversion” on page 6-36</td>
</tr>
</tbody>
</table>

**Engineering Units**

Data Acquisition Toolbox provides you with a way to linearly scale data as it is being queued in the engine. You can associate this scaling with specific engineering units such as volts or Newtons that you might want to apply to your data.

The properties associated with engineering units and linearly scaling output data are as follows:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutputRange</td>
<td>Specify the range of the analog output hardware subsystem.</td>
</tr>
<tr>
<td>Units</td>
<td>Specify the engineering units label.</td>
</tr>
<tr>
<td>UnitsRange</td>
<td>Specify the range of data as engineering units.</td>
</tr>
</tbody>
</table>

For many devices, the output range is expressed in terms of the gain and polarity.

**Note** You can set the engineering units properties on a per-channel basis. Therefore, you can configure different engineering unit conversions for each hardware channel.

Linearly scaled output data is given by the formula:

\[
\text{scaled value} = \frac{\text{original value} \times \text{output range}}{\text{units range}}
\]
Analog Output

The units range is given by the UnitsRange property, while the output range is given by the OutputRange property. UnitsRange controls the scaling of data when it is queued in the engine with the putdata function. OutputRange specifies the gain and polarity of your D/A subsystem. You should choose an output range that encompasses the output signal, and that utilizes the maximum dynamic range of your hardware.

For sound cards, you might have to adjust the volume control to obtain the full-scale output range of the device. Refer to “Sound Cards” on page A-12 to learn how to access the volume control for your sound card.

For example, suppose OutputRange is [-10 10], and UnitsRange is [-5 5]. If a queued value is 2.5, then the scaled value is \((2.5)(20/10)\) or 5, in the appropriate units.

**Note** The data acquisition engine always *clips* out-of-range values. Clipping means that an out-of-range value is fixed to either the minimum or maximum value that is representable by the hardware. Clipping is equivalent to saturation.

---

**Example: Performing a Linear Conversion**

This example illustrates how to configure the engineering units properties for an analog output object connected to a National Instruments PCI-6024E board.

The queued data consists of a 4 volt peak-to-peak sine wave. The UnitsRange property is configured so that queued data is scaled to the OutputRange property value, which is fixed at ±10 volts. This scaling utilizes the maximum dynamic range of the analog output hardware.

You can run this example by typing `daqdoc6_6` at the MATLAB command line.

1 **Create a device object** — Create the analog output object AO for a National Instruments board. The installed adaptors and hardware IDs are found with daqhwinfo.

   ```matlab
   AO = analogoutput('nidaq','Dev1');
   ```
2 Add channels — Add one hardware channel to $A_0$.

```matlab
chan = addchannel(A0,0);
```

3 Configure property values — Create the data to be queued.

```matlab
freq = 500;
w = 2*pi*freq;
t = linspace(0,2,20000);
data = 2*sin(w*t)';
```

Configure the sampling rate to 5 kHz, configure the trigger to repeat two times, and scale the data to cover the full output range of the D/A converter. Because the peak-to-peak amplitude of the queued data is 4, UnitsRange is set to $[-2 2]$, which scales the output data to 20 volts peak-to-peak.

```matlab
set(A0,'SampleRate',5000)
set(A0,'RepeatOutput',2)
set(chan,'UnitsRange',[-2 2])
```

Queue the data with one call to putdata.

```matlab
putdata(A0,data)
```

4 Output data — Start $A_0$ and wait until all the data is output.

```matlab
start(A0)
wait(A0,6)
```

5 Clean up — When you no longer need $A_0$, you should remove it from memory and from the MATLAB workspace.

```matlab
delete(A0)
clear AO
```
Starting Multiple Device Objects

With Data Acquisition Toolbox, you can start multiple device objects. You might find this feature useful when simultaneously using your hardware’s analog output (AO) and analog input (AI) subsystems. For example, suppose you create the analog input object `ai` and the analog output object `ao` for a sound card, and add one channel to each device object.

```matlab
ai = analoginput('winsound');
addchannel(ai,1);
ao = analogoutput('winsound');
addchannel(ao,1);
```

You should use manual triggers when starting multiple device objects because this trigger type executes faster than other trigger types with the exception of hardware triggers. Additionally, to synchronize the input and output of data, you should configure the `ManualTriggerHwOn` property to `Trigger` for `ai`.

```matlab
set([ai ao],'TriggerType','Manual')
set(ai,'ManualTriggerHwOn','Trigger')
```

Configure `ai` for continuous acquisition, call the callback function `qmoredata` whenever 1000 samples are output, and call `daqcallback` when `ai` and `ao` stop running.

```matlab
set(ai,'SamplesPerTrigger',inf)
set(ao,'SamplesOutputFcn',{'qmoredata',ai})
set(ao,'SamplesOutputFcnCount',1000)
set([ai ao],'StopFcn',@daqcallback)
```

As shown below, the callback function `qmoredata` extracts data from the engine and then queues it for output.

```matlab
function qmoredata(obj,event,ai)
data = getdata(ai,1000);
putdata(obj,data)
```
Queue data in the engine, start the device objects, and execute the manual triggers.

```matlab
data = zeros(4000,1);
putdata(ao,data)
start([ai ao])
trigger([ai ao])
```

**Note** Device objects cannot execute simultaneously unless you use an external hardware trigger.

You can determine the starting time for each device object with the `InitialTriggerTime` property. The difference, in seconds, between the starting times for `ai` and `ao` is

```matlab
aitime = ai.InitialTriggerTime
aotime = ao.InitialTriggerTime
delta = abs(aotime - aitime);
sprintf('%d',delta(6))
ans =
2.28818e-005
```

Note that this number depends on the specific platform you are using. To stop both device objects:

```matlab
stop([ai ao])
```

The output from `daqcallback` is shown below.

Stop event occurred at 13:00:25 for the object: winsound0-AO.
Stop event occurred at 13:00:25 for the object: winsound0-AI.
Digital Input/Output

Digital I/O (DIO) subsystems are designed to transfer digital values to and from hardware. These values are handled either as single bits or lines, or as a port, which typically consists of eight lines. While most popular data acquisition boards include some DIO capability, it is usually limited to simple operations and special dedicated hardware is required for performing advanced DIO operations. Data Acquisition Toolbox provides access to digital I/O subsystems through a digital I/O object. The DIO object can be associated with a parallel port or with a DIO subsystem on a data acquisition board.

The purpose of this chapter is to show you how to perform data acquisition tasks using your digital I/O hardware. The sections are as follows.

- Digital I/O Objects (p. 7-3) Create a MATLAB object that represents the digital I/O subsystem
- Adding Lines to a Digital I/O Object (p. 7-6) Associate hardware lines with the digital I/O object
- Writing and Reading Digital I/O Line Values (p. 7-15) Write values to digital lines, and read values from digital lines
- Generating Timer Events (p. 7-20) Execute the digital I/O object and configure properties to generate timer events
- Evaluating the Digital I/O Object Status (p. 7-24) Return the values of certain properties in a convenient display format
**Note** Data Acquisition Toolbox does not directly support buffered DIO or handshaking (latching). However, you can write your own M-code to support this functionality. Buffered DIO means that the data is stored in the engine. Handshaking allows the DIO subsystem to input or output values after receiving a digital pulse.
Digital I/O Objects

<table>
<thead>
<tr>
<th>In this section...</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Creating a Digital I/O Object” on page 7-3</td>
</tr>
<tr>
<td>“The Parallel Port” on page 7-4</td>
</tr>
</tbody>
</table>

Creating a Digital I/O Object

You create a digital I/O (DIO) object with the `digitalio` function. `digitalio` accepts the adaptor name and the hardware device ID as input arguments. For parallel ports, the device ID is the port label (LPT1, LPT2, or LPT3). For data acquisition boards, the device ID refers to the number associated with the board when it is installed. Note that some vendors refer to the device ID as the device number or the board number. When using NI-DAQmx, this is usually a string such as `'Dev1'`.) Use the `daqhwinfo` function to determine the available adaptors and device IDs.

Each DIO object is associated with one parallel port or one subsystem. For example, to create a DIO object associated with a National Instruments board:

```matlab
dio = digitalio('nidaq','Dev1');
```

The digital I/O object `dio` now exists in the MATLAB workspace. You can display the class of `dio` with the `whos` command.

```matlab
whos dio
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>dio</td>
<td>1x1</td>
<td>1308</td>
<td>digitalio object</td>
</tr>
</tbody>
</table>

Grand total is 40 elements using 1308 bytes

Once the object is created, the properties listed below are automatically assigned values. These general purpose properties provide descriptive information about the object based on its class type and adaptor.
Table 7-1  Descriptive Digital I/O Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Indicate the device object type.</td>
</tr>
<tr>
<td>Name</td>
<td>Specify a descriptive name for the device object.</td>
</tr>
</tbody>
</table>

You can display the values of these properties for dio with the get function.

```matlab
get(dio,{'Name','Type'})
ans =
    'nidaq1-DIO'    'Digital IO'
```

The Parallel Port
The PC supports up to three parallel ports that are assigned the labels LPT1, LPT2, and LPT3. You can use any of these standard ports as long as they use the usual base addresses, which are (in hex) 378, 278, and 3BC, respectively. The port labels and addresses are typically configured through the PC's BIOS. Additional ports, or standard ports not assigned the usual base addresses, are not accessible by the toolbox.

Most PCs that support MATLAB will include a single parallel port with label LPT1 and base address 378. To create a DIO object for this port,

```matlab
parport = digitalio('parallel','LPT1');
```

**Note** The parallel port is not locked by MATLAB. Therefore, other applications or other instances of MATLAB can access the same parallel port, which can result in a conflict.

Administrator Privileges for Parallel Port Pins
Accessing the individual pins of the parallel port under Windows 2000 and Windows XP is a privileged operation. Data Acquisition Toolbox installs a driver called winio.sys that provides access to the parallel port pins. Normally, only users with administrator privileges can do this.
If you want to allow users without administrator privileges to use the parallel port from Data Acquisition Toolbox, you need to do the following:

1 Log in to your machine as the administrator.

2 Start MATLAB.

3 At the MATLAB command line, type

   `daqhwinfo('parallel');`

4 Minimize the MATLAB window.

5 On the desktop, select **My Computer** and right-click. Choose **Properties** from the menu that appears.

6 In the dialog box that appears, click the **Hardware** tab, and click the **Device Manager** button.

7 In the window that appears, select **View > Show Hidden Devices**, and expand the **Non-Plug and Play Drivers** item in the list.

8 Find the **WINIO** item near the bottom of the list. Double-click it, and click the **Driver** tab in the window that appears.

9 Expand the **Startup Type** drop-down list and change the entry from **Demand to Boot**. This causes the WINIO driver to start up every time the machine is rebooted.

10 Close all the open windows, including MATLAB, and reboot your machine.

Users with standard or power-user privileges can now access the parallel port pins.
Adding Lines to a Digital I/O Object

In this section...

“Using the Addline Function” on page 7-6

“Line and Port Characteristics” on page 7-8

“Referencing Individual Hardware Lines” on page 7-12

Using the Addline Function

After creating the digital I/O (DIO) object, you must add lines to it. As shown by the figure in “Hardware Channels or Lines” on page 3-9, you can think of a device object as a container for lines. The collection of lines contained by the DIO object is referred to as a line group. A line group consists of a mapping between hardware line IDs and MATLAB indices (see below).

When adding lines to a DIO object, you must follow these rules:

- The lines must reside on the same hardware device. You cannot add lines from different devices, or from different subsystems on the same device.
- You can add a line only once to a given digital I/O object. However, a line can be added to as many different digital I/O objects as you desire.
- You can add lines that reside on different ports to a given digital I/O object.

You add lines to a digital I/O object with the addline function. addline requires the device object, at least one hardware line ID, and the direction (input or output) of each added line as input arguments. You can optionally specify port IDs, descriptive line names, and an output argument. For example, to add eight output lines from port 0 to the device object dio created in the preceding section:

```
hwlines = addline(dio,0:7,'out');
```

The output argument hwlines is a column vector that reflects the line group contained by dio. You can display the class of hwlines with the whos command.

```
whos hwlines
```
Adding Lines to a Digital I/O Object

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>hwlines</td>
<td>8x1</td>
<td>536</td>
<td>dioline object</td>
</tr>
</tbody>
</table>

Grand total is 13 elements using 536 bytes

You can use `hwlines` to easily access lines. For example, you can configure or return property values for one or more lines. As described in “Referencing Individual Hardware Lines” on page 7-12, you can also access lines with the `Line` property.

Once you add lines to a DIO object, the properties listed below are automatically assigned values. These properties provide descriptive information about the lines based on their class type and ID.

**Table 7-2 Descriptive Digital I/O Line Properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HwLine</td>
<td>Specify the hardware line ID.</td>
</tr>
<tr>
<td>Index</td>
<td>Indicate the MATLAB index of a hardware line.</td>
</tr>
<tr>
<td>Parent</td>
<td>Indicate the parent (device object) of a line.</td>
</tr>
<tr>
<td>Type</td>
<td>Indicate a line.</td>
</tr>
</tbody>
</table>

You can display the values of these properties for `hwlines` with the `get` function.

```matlab
get(hwlines,{'HwLine','Index','Parent','Type'})
```

```matlab
an =
[0] [1]  [1x1 digitalio] 'Line'
[1] [2]  [1x1 digitalio] 'Line'
[2] [3]  [1x1 digitalio] 'Line'
[3] [4]  [1x1 digitalio] 'Line'
[4] [5]  [1x1 digitalio] 'Line'
[5] [6]  [1x1 digitalio] 'Line'
[6] [7]  [1x1 digitalio] 'Line'
[7] [8]  [1x1 digitalio] 'Line'
```
Line and Port Characteristics

As described in the preceding section, when you add lines to a DIO object, they must be configured for either input or output. You read values from an input line and write values to an output line. Whether a given hardware line is addressable for input or output depends on the port it resides on. You can classify digital I/O ports into these two groups based on your ability to address lines individually:

- **Port-configurable devices** — You cannot address the lines associated with a port-configurable device individually. Therefore, you must configure all the lines for either input or output. If you attempt to mix the two configurations, an error is returned.

You can add any number of available port-configurable lines to a DIO object. However, the engine will address all lines behind the scenes. For example, if one line is added to a DIO object, then you automatically get all lines. Therefore, if a DIO object contains lines from a port-configurable device, and you write a value to one or more of those lines, then all the lines are written to even if they are not contained by the device object.

- **Line-configurable devices** — You can address the lines associated with a line-configurable device individually. Therefore, you can configure individual lines for either input or output. Additionally, you can read and write values on a line-by-line basis. Note that for National Instruments E-Series hardware, port 0 is always line-configurable, while all other ports are port-configurable. Port 0 is line-configurable only for E-Series devices of the traditional National Instruments drivers. Note that NI-DAQmx devices do not support this.

You can return the line and port characteristics with the `daqhwinfo` function. For example, National Instruments AT-MIO-16DE-10 board has four ports with eight lines per port. To return the digital I/O characteristics for this board:

```matlab
hwinfo = daqhwinfo(dio);
```

Display the line characteristics for each port.

```matlab
hwinfo.Port(1)
ans =
    ID: 0
```
Adding Lines to a Digital I/O Object

This information tells you that you can configure all 32 lines for either input or output, and that the first port is line-configurable while the remaining ports are port-configurable.

Parallel Port Characteristics

The parallel port consists of eight data lines, four control lines, five status lines, and eight ground lines. In normal usage, the lines are controlled by the host computer software and the peripheral device following a protocol such as IEEE Standard 1284-1994. The protocol defines procedures for transferring data such as handshaking, returning status information, and so on. However, the toolbox uses the parallel port as a basic digital I/O device, and no protocol is needed. Therefore, you can use the port to input and output digital values just as you would with a typical DIO subsystem.

To access the physical parallel port lines, most PCs come equipped with one 25-pin female connector, which is shown below.
The lines use TTL logic levels. A line is high (true or asserted) when it is a TTL high level, while a line is low (false or unasserted) when it is a TTL low level. The exceptions are lines 1, 11, 14, and 17, which are hardware inverted.

The toolbox groups the 17 nonground lines into three separate ports. The port IDs and the associated pin numbers are given below:

Table 7-3  Parallel Port IDs and Pin Numbers

<table>
<thead>
<tr>
<th>Port ID</th>
<th>Pins</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2-9</td>
<td>Eight I/O lines, with pin 9 being the most significant bit (MSB).</td>
</tr>
<tr>
<td>1</td>
<td>10-13, and 15</td>
<td>Five input lines used for status</td>
</tr>
<tr>
<td>2</td>
<td>1, 14, 16, and 17</td>
<td>Four I/O lines used for control</td>
</tr>
</tbody>
</table>

Note that in some cases, port 0 lines might be unidirectional and only output data. If supported by the hardware, you can configure these lines for both input and output with your PC’s BIOS by selecting a bidirectional mode such as EPP (Enhanced Parallel Port) or ECP (Extended Capabilities Port).

The parallel port characteristics for the DIO object parport are shown below.

```matlab
hwinfo = daqhwinfo(parport);
hwinfo.Port(1)
ans =

    ID: 0
LineIDs: [0 1 2 3 4 5 6 7]
Direction: 'in/out'
Config: 'port'
```

```matlab
hwinfo.Port(2)
```
Adding Lines to a Digital I/O Object

ans =

    ID: 1
    LineIDs: [0 1 2 3 4]
    Direction: 'in'
    Config: 'port'

hwinfo.Port(3)
ans =

    ID: 2
    LineIDs: [0 1 2 3]
    Direction: 'in/out'
    Config: 'port'

This information tells you that all 17 lines are port-configurable, you can input and output values using the 12 lines associated with ports 0 and 2, and that you can only input values from the five lines associated with port 1.

For easy reference, the LineName property is automatically populated with a name that includes the port pin number. For example:

dio = digitalio('parallel', 1)

display Summary of DigitalIO (DIO) Obj Using 'PC Parallel Port Hardware'.

Port Parameters: Port 0 is port configurable for reading and writing.

Port 1 is port configurable for reading.

Port 2 is port configurable for reading and writing.

Engine status: Engine not required.

DIO object contains no lines.

addline(dio, 0:16, 'in')

<table>
<thead>
<tr>
<th>Index</th>
<th>LineName</th>
<th>HwLine</th>
<th>Port</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>'Pin2'</td>
<td>0</td>
<td>0</td>
<td>'In'</td>
</tr>
<tr>
<td>2</td>
<td>'Pin3'</td>
<td>1</td>
<td>0</td>
<td>'In'</td>
</tr>
<tr>
<td>3</td>
<td>'Pin4'</td>
<td>2</td>
<td>0</td>
<td>'In'</td>
</tr>
<tr>
<td>4</td>
<td>'Pin5'</td>
<td>3</td>
<td>0</td>
<td>'In'</td>
</tr>
<tr>
<td>5</td>
<td>'Pin6'</td>
<td>4</td>
<td>0</td>
<td>'In'</td>
</tr>
</tbody>
</table>
Referencing Individual Hardware Lines

As described in the preceding section, you can access lines with the Line property or with a line object. To reference individual lines, you must specify either MATLAB indices or descriptive line names.

MATLAB Indices

Every hardware line contained by a DIO object has an associated MATLAB index that is used to reference the line. When adding lines with the addline function, index assignments are made automatically. The line indices start at 1 and increase monotonically up to the number of line group members. The first line indexed in the line group represents the least significant bit (LSB). Unlike adding channels with the addchannel function, you cannot manually assign line indices with addline.

For example, the digital I/O object dio created in the preceding section has the MATLAB indices 1 through 8 automatically assigned to the hardware lines 0 through 7, respectively. To swap the first two hardware lines so that line ID 1 is the LSB, you can supply the appropriate index to hwlines and use the HwLine property.

```matlab
hwlines(1).HwLine = 1;
hwlines(2).HwLine = 0;
```

Alternatively, you can use the Line property.

```matlab
dio.Line(1).HwLine = 1;
```
Adding Lines to a Digital I/O Object

dio.Line(2).HwLine = 0;

Descriptive Line Names
Choosing a unique, descriptive name can be a useful way to identify and reference lines — particularly for large line groups. You can associate descriptive names with hardware lines with the addline function. For example, suppose you want to add 8 lines to dio, and you want to associate the name TrigLine with the first line added.

```matlab
addline(dio,0,'out','TrigLine');
addline(dio,1:7,'out');
```

Alternatively, you can use the LineName property.

```matlab
addline(dio,0:7,'out');
dio.Line(1).LineName = 'TrigLine';
```

You can now use the line name to reference the line.

```matlab
dio.TrigLine.Direction = 'in';
```

Example: Adding Lines for National Instruments Hardware
This example illustrates various ways you can add lines to a DIO object associated with a National Instruments AT-MIO-16DE-10 board. This board is a multiport device whose characteristics are described in “Line and Port Characteristics” on page 7-8.

To add eight input lines to dio from port 0:

```matlab
addline(dio,0:7,'in');
```

To add four input lines and four output lines to dio from port 0:

```matlab
addline(dio,0:7,{'in','in','in','in','out','out','out','out'});
```

Suppose you want to add the first two lines from port 0 configured for input, and the first two lines from port 2 configured for output. There are four ways to do this. The first way requires only one call to addline because it uses the hardware line IDs, and not the port IDs.

```matlab
addline(dio,[0 1 8 9],{'in','in','out','out'});
```
The second way requires two calls to addline, and specifies one line ID and multiple port IDs for each call.

```plaintext
addline(dio, 0, [0 2], {'in', 'out'});
addline(dio, 1, [0 2], {'in', 'out'});
```

The third way requires two calls to addline, and specifies multiple line IDs and one port ID for each call.

```plaintext
addline(dio, 0:1, 0, 'in');
addline(dio, 0:1, 2, 'out');
```

Lastly, you can use four addline calls — one for each line added.
Writing and Reading Digital I/O Line Values

<table>
<thead>
<tr>
<th>In this section...</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Writing Digital Values” on page 7-15</td>
</tr>
<tr>
<td>“Reading Digital Values” on page 7-17</td>
</tr>
<tr>
<td>“Example: Writing and Reading Digital Values” on page 7-18</td>
</tr>
</tbody>
</table>

Writing Digital Values

Note Unlike analog input and analog output objects, you do not control the behavior of DIO objects by configuring properties. This is because buffered DIO is not supported, and data is not stored in the engine. Instead, you either write values directly to, or read values directly from the hardware lines.

You write values to digital lines with the `putvalue` function. `putvalue` requires the DIO object and the values to be written as input arguments. You can specify the values to be written as a decimal value or as a *binary vector* (binvec). A binary vector is a logical array that is constructed with the least significant bit (LSB) in the first column and the most significant bit (MSB) in the last column. For example, the decimal value 23 is written in binvec notation as \([1 \ 1 \ 1 \ 0 \ 1] = 2^0 + 2^1 + 2^2 + 2^4\). You might find that binvecs are easier to work with than decimal values because there is a clear association between a given line and the value (1 or 0) that is written to it. You can convert decimal values to binvec values with the `dec2binvec` function.

For example, suppose you create the digital I/O object `dio` and add eight output lines to it from port 0.

```matlab
dio = digitalio('nidaq','Dev1');
addline(dio,0:7,'out');
```

To write a value of 23 to the eight lines contained by `dio`, you can write to the device object.

```matlab
data = 23;
putvalue(dio,data)
```
Alternatively, you can write to individual lines through the `Line` property.

```matlab
putvalue(dio.Line(1:8),data)
```

To write a binary vector of values using the device object and the `Line` property:

```matlab
bvdata = dec2binvec(data,8);
putvalue(dio,bvdata)
putvalue(dio.Line(1:8),bvdata)
```

The second input argument supplied to `dec2binvec` specifies the number of bits used to represent the decimal value. Because the preceding commands write to all eight lines contained by `dio`, an eight element binary vector is required. If you do not specify the number of bits, then the minimum number of bits needed to represent the decimal value is used.

Alternatively, you can create the binary vector without using `dec2binvec`.

```matlab
bvdata = logical([1 1 1 0 1 0 0 0]);
putvalue(dio,bvdata)
```

**Rules for Writing Digital Values**

Writing values to digital I/O lines follows these rules:

- If the DIO object contains lines from a port-configurable device, then the data acquisition engine writes to all lines associated with the port even if they are not contained by the device object.

- When writing decimal values,
  - If the value is too large to be represented by the lines contained by the device object, then an error is returned.
  - You can write to a maximum of 32 lines. To write to more than 32 lines, you must use a `binvec` value.

- When writing `binvec` values,
  - You can write to any number of lines.
  - There must be an element in the binary vector for each line you write to.
• You can always read from a line configured for output. Reading values is discussed in “Reading Digital Values” on page 7-17.

• An error is returned if you write a negative value, or if you write to a line configured for input.

**Reading Digital Values**

**Note** Unlike analog input and analog output objects, you do not control the behavior of DIO objects by configuring properties. This is because buffered DIO is not supported, and data is not stored in the engine. Instead, you either write values directly to, or read values directly from the hardware lines.

You can read values from one or more lines with the `getvalue` function. `getvalue` requires the DIO object as an input argument. You can optionally specify an output argument, which represents the returned values as a binary vector. Binary vectors are described in “Writing Digital Values” on page 7-15.

For example, suppose you create the digital I/O object `dio` and add eight input lines to it from port 0.

```matlab
dio = digitalio('nidaq','Dev1');
addline(dio,0:7,'in');
```

To read the current value of all the lines contained by `dio`:

```matlab
portval = getvalue(dio)
portval =
    1   1   1   0   1   0   0   0
```

To read the current values of the first five lines contained by `dio`:

```matlab
lineval = getvalue(dio.Line(1:5))
lineval =
    1   1   1   0   1
```

You can convert a binvec to a decimal value with the `binvec2dec` function. For example, to convert the binary vector `lineval` to a decimal value:

```matlab
out = binvec2dec(lineval)
```
Rules for Reading Digital Values

Reading values from digital I/O lines follows these rules:

- If the DIO object contains lines from a port-configurable device, then all lines are read even if they are not contained by the device object. However, only values from the lines contained by the object are returned.
- You can always read from a line configured for output.
- For National Instruments hardware using the Traditional NI-DAQ interface, lines configured for output return a value of 1 by default.
- `getvalue` always returns a binary vector (binvec). To convert the binvec to a decimal value, use the `binvec2dec` function.

Example: Writing and Reading Digital Values

This example illustrates how to read and write digital values using a line-configurable subsystem. With line-configurable subsystems, you can transfer values on a line-by-line basis.

You can run this example by typing `daqdoc7_1` at the MATLAB command line.

1 Create a device object — Create the digital I/O object `dio` for a National Instruments board. The installed adaptors and hardware IDs are found with `daqhwinfo`.

```matlab
dio = digitalio('nidaq','Dev1');
```

2 Add lines — Add eight output lines from port 0 (line-configurable).

```matlab
addline(dio,0:7,'out');
```

3 Read and write values — Write a value of 13 to the first four lines as a decimal number and as a binary vector, and read back the values.

```matlab
data = 13;
putvalue(dio.Line(1:4),data)
val1 = getvalue(dio);
```
bvdata = dec2binvec(data);
putvalue(dio.Line(1:4), bvdata)
val2 = getvalue(dio);

Write a value of 3 to the last four lines as a decimal number and as a binary vector, and read back the values.

data = 3;
putvalue(dio.Line(5:8), data)
val3 = getvalue(dio.Line(5:8));
bvdata = dec2binvec(data, 4);
putvalue(dio.Line(5:8), bvdata)
val4 = getvalue(dio.Line(5:8));

Read values from the last four lines but switch the most significant bit (MSB) and the least significant bit (LSB).

val5 = getvalue(dio.Line(8:-1:5));

4 Clean up — When you no longer need dio, you should remove it from memory and from the MATLAB workspace.

delete(dio)
clear dio
Generating Timer Events

Overview
The fact that analog input and analog output objects make use of data stored in the engine and clocked I/O leads to the concept of a “running” device object and the generation of events.

However, because Data Acquisition Toolbox does not support buffered digital I/O (DIO) operations, DIO objects do not store data in the engine. Additionally, reading and writing line values are not clocked at a specific rate in the way that data is sampled by an analog input or analog output subsystem. Instead, values are either written directly to digital lines with putvalue, or read directly from digital lines with getvalue.

Therefore, the concept of a running DIO object does not make sense in the same way that it does for analog I/O. However, you can “run” a DIO object to perform one task: generate timer events. You can use timer events to update and display the state of the DIO object. Refer to the diopanel demo for an example.

Timer Events
The only event supported by DIO objects is a timer event. Timer events occur after a specified period of time has passed. Properties associated with generating timer events are given below.
Generating Timer Events

Table 7-4 Digital I/O Timer Event Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running</td>
<td>Indicate if the device object is running.</td>
</tr>
<tr>
<td>TimerFcn</td>
<td>Specify the M-file callback function to execute whenever a predefined period of time passes.</td>
</tr>
<tr>
<td>TimerPeriod</td>
<td>Specify the period of time between timer events.</td>
</tr>
</tbody>
</table>

A timer event is generated whenever the time specified by TimerPeriod passes. This event executes the callback function specified for TimerFcn. Time is measured relative to when the device object starts running (Running is On). Starting a DIO object is discussed in the next section.

Some timer events might not be processed if your system is significantly slowed or if the TimerPeriod value is too small. For example, a common application for timer events is to display data. However, because displaying data can be a CPU-intensive task, some of these events might be dropped. For digital I/O objects, timer events are typically used to display the state of the object.

To see how to construct a callback function, refer to “Creating and Executing Callback Functions” on page 5-52 or the example below.

Starting and Stopping a Digital I/O Object

You use the start function to start a DIO object. For example, to start the digital I/O object dio:

```matlab
start(dio)
```

After start is issued, the Running property is automatically set to On, and timer events can be generated. If you attempt to start a digital I/O object that does not contain any lines or that is already running, an error is returned.

A digital I/O object will stop executing under these conditions:

- The stop function is issued.
• An error occurred in the system.

When the device object stops, Running is automatically set to Off.

**Example: Generating Timer Events**

This example illustrates how to generate timer events for a DIO object. The callback function daqcallback displays the event type and device object name. Note that you must issue a stop command to stop the execution of the object.

You can run this example by typing daqdoc7_2 at the MATLAB command line.

1 **Create a device object** — Create the digital I/O object dio for a National Instruments board. The installed adaptors and hardware IDs are found with daqhwinfo.

   ```matlab
dio = digitalio('nidaq','Dev1');
```

2 **Add lines** — Add eight input lines from port 0 (line-configurable).

   ```matlab
   addline(dio,0:7,'in');
   ```

3 **Configure property values** — Configure the timer event to call daqcallback every five seconds.

   ```matlab
   set(dio,'TimerFcn',@daqcallback)
   set(dio,'TimerPeriod',5.0)
   ```

Start the digital I/O object. You must issue a stop command when you no longer want to generate timer events.

```matlab
start(dio)
```

The pause command ensures that two timer events are generated when you run daqdoc7_2 from the command line.

```matlab
pause(11)
```
4 **Clean up** — When you no longer need `dio`, you should remove it from memory and from the MATLAB workspace.

```matlab
delete(dio)
clear dio
```
Evaluating the Digital I/O Object Status

In this section...

“Running Property” on page 7-24
“The Display Summary” on page 7-24

Running Property
You can evaluate the status of a digital I/O (DIO) object by returning the value of the Running property (this is useful only if timer events are generated).

The Display Summary
You can invoke the display summary by typing a DIO object or a line object at the MATLAB command line, or by excluding the semicolon when

- Creating a DIO object
- Adding lines
- Configuring property values using the dot notation

You can also display summary information via the Workspace browser by right-clicking a toolbox object and selecting Explore > Display Summary from the context menu.

The displayed information is designed so you can quickly evaluate the status of your data acquisition session. The display is divided into two main sections: general summary information and line summary information.

General Summary Information
The general display summary includes the device object type and the hardware device name, followed by the port parameters. The port parameters include the port ID, and whether the associated lines are configurable for reading or writing.

Line Summary Information
The line display summary includes property values associated with
Evaluating the Digital I/O Object Status

- The hardware line mapping
- The line name
- The port ID
- The line direction

The display summary for the example given in “Example: GeneratingTimer
Events” on page 7-22 is shown below.

```
Display Summary of DigitalIO (DIO) Object Using 'PCI-6024E'.

Port Parameters: Port 0 is line configurable for reading and writing.

Engine status: Engine not required.

DIO object contains line(s):

<table>
<thead>
<tr>
<th>Index</th>
<th>LineName</th>
<th>HwLine</th>
<th>Port</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>' '</td>
<td>0</td>
<td>0</td>
<td>'In'</td>
</tr>
<tr>
<td>2</td>
<td>' '</td>
<td>1</td>
<td>0</td>
<td>'In'</td>
</tr>
<tr>
<td>3</td>
<td>' '</td>
<td>2</td>
<td>0</td>
<td>'In'</td>
</tr>
<tr>
<td>4</td>
<td>' '</td>
<td>3</td>
<td>0</td>
<td>'In'</td>
</tr>
<tr>
<td>5</td>
<td>' '</td>
<td>4</td>
<td>0</td>
<td>'In'</td>
</tr>
<tr>
<td>6</td>
<td>' '</td>
<td>5</td>
<td>0</td>
<td>'In'</td>
</tr>
<tr>
<td>7</td>
<td>' '</td>
<td>6</td>
<td>0</td>
<td>'In'</td>
</tr>
<tr>
<td>8</td>
<td>' '</td>
<td>7</td>
<td>0</td>
<td>'In'</td>
</tr>
</tbody>
</table>
```

You can use the Line property to display only the line summary information.

DIO.Line
This chapter describes how to save and load information associated with a data acquisition session. The sections are as follows.

Saving and Loading Device Objects (p. 8-2)

Logging Information to Disk (p. 8-5)
Saving and Loading Device Objects

In this section...

“Saving Device Objects to an M-File” on page 8-2
“Saving Device Objects to a MAT-File” on page 8-4

Saving Device Objects to an M-File

Note For analog input objects, you can also save acquired data, hardware information, and so on to a log file. Refer to “Logging Information to Disk” on page 8-5 for more information.

You can save a device object to an M-file using the obj2mfile function. obj2mfile provides you with these options:

• Save all property values, or save only those property values that differ from their default values.

Read-only property values are not saved. Therefore, read-only properties use their default values when you load the device object into the MATLAB workspace. To determine if a property is read-only, use the propinfo function or examine the property reference pages.

• Save property values using the set syntax, the dot notation, or named referencing (if defined).

If the UserData property is not empty, or if a callback property is set to a cell array of values or a function handle, then the data stored in these properties is written to a MAT-file when the device object is saved. The MAT-file has the same name as the M-file containing the device object code.

For example, suppose you create the analog input object ai for a sound card, add two channels to it, and configure several property values.

    ai = analoginput('winsound');
    addchannel(ai,1:2,{'Temp1';'Temp2'});
    time = now;
    set(ai,'SampleRate',11025,'TriggerRepeat',4)
set(ai,'TriggerFcn',{@mycallback,time})
start(ai)

The following command saves ai and the modified property values to the M-file myai.m. Because the TriggerFcn property is set to a cell array of values, its value is automatically written to the MAT-file myai.mat.

    obj2mfile(ai,'myai.m');

Created: d:\v6\myfiles\myai.m
Created: d:\v6\myfiles\myai.mat

Use the type command to display myai.m at the command line.

**Loading the Device Object**

To load a device object that was saved as an M-file into the MATLAB workspace, type the name of the M-file at the command line. For example, to load ai from the M-file myai.m:

    ai = myai

Note that the read-only properties such as SamplesAcquired and SamplesAvailable are restored to their default values.

    get(ai,{'SamplesAcquired','SamplesAvailable'})
    ans =
    [0]    [0]

When loading ai into the workspace, the MAT-file myai.mat is automatically loaded and the TriggerFcn property value is restored.

    ai.TriggerFcn
    ans =
    [@mycallback]    [7.3071e+005]
Saving Device Objects to a MAT-File

**Note** For analog input objects, you can also save acquired data, hardware information, and so on to a log file. Refer to “Logging Information to Disk” on page 8-5 for more information.

You can save a device object to a MAT-file just as you would any workspace variable — using the `save` command. For example, to save the analog input object `ai` and the variable `time` defined in the preceding section to the MAT-file `myai1.mat`:

```
save myai1 ai time
```

Read-only property values are not saved. Therefore, read-only properties use their default values when you load the device object into the MATLAB workspace. To determine if a property is read-only, use the `propinfo` function or examine the property reference pages.

**Loading the Device Object**

To load a device object that was saved to a MAT-file into the MATLAB workspace, use the `load` command. For example, to load `ai` and `time` from MAT-file `myai1.mat`:

```
load myai1
```
Logging Information to Disk

In this section...

“Analog Input Logging Properties” on page 8-5
“Specifying a Filename” on page 8-6
“Retrieving Logged Information” on page 8-7
“Example: Logging and Retrieving Information” on page 8-9

Analog Input Logging Properties

While an analog input object is running, you can log this information to a disk file:

- Acquired data
- Event information
- Device object and channel information
- Hardware information

Logging information to disk provides a permanent record of your data acquisition session, and is an easy way to debug your application.

As shown below, you can think of the logged information as a stream of data and events.

![Diagram showing a stream of data and events with triggers and stops marked.] 

Data logged to file
The properties associated with logging information to a disk file are as follows:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogFileName</td>
<td>Specify the name of the disk file to which information is logged.</td>
</tr>
<tr>
<td>Logging</td>
<td>Indicate if data is being logged.</td>
</tr>
<tr>
<td>LoggingMode</td>
<td>Specify the destination for acquired data.</td>
</tr>
<tr>
<td>LogToDiskMode</td>
<td>Specify whether data, device object information, and hardware information is saved to one disk file or to multiple disk files.</td>
</tr>
</tbody>
</table>

You can initiate logging by setting LoggingMode to Disk or Disk&Memory. A new log file is created each time you issue the start function, and each different analog input object must log information to a separate log file. Writing to disk is performed as soon as possible after the current data block is filled.

You can choose whether a log file is overwritten or if multiple log files are created with the LogToDiskMode property. If LogToDiskMode is Overwrite, the log file is overwritten. If LogToDiskMode is Index, new log files are created, each with an indexed name based on the value of LogFileName.

**Specifying a Filename**

You specify the name of the log file with the LogFileName property. You can specify any value for LogFileName, including a directory path, provided the filename is supported by your operating system. Additionally, if LogToDiskMode is Index, then the log filename also follows these rules:

- Indexed log filenames are identified by a number. This number precedes the filename extension and is incremented by one for successive log files.
- If no number is specified as part of the initial log filename, then the first log file does not have a number associated with it. For example, if LogFileName is myfile.daq, then myfile.daq is the name of the first log file, myfile01.daq is the name of the second log file, and so on.
- LogFileName is updated after the log file is written (after the stop event occurs).
If the specified log filename already exists, then the existing file is overwritten.

**Retrieving Logged Information**

You retrieve logged information with the `daqread` function. You can retrieve any part of the information stored in a log file with one call to `daqread`. However, you will probably use `daqread` in one of these two ways:

- Retrieving data and time information
- Retrieving event, device object, channel, and hardware information

**Retrieving Data and Time Information**

You can characterize logged data by the sample number or the time the sample was acquired. To retrieve data and time information, you use the syntax shown below:

\[
\text{[data, time, abstime] = daqread('file','P1',V1,'P2',V2,...);} \]

where

- `data` is the retrieved data. Data is returned as an m-by-n matrix where m is the number of samples and n is the number of channels.
- `time` (optional) is the relative time associated with the retrieved data. Time is returned as an m-by-1 matrix where m is the number of samples.
- `abstime` (optional) is the absolute time of the first trigger. Absolute time is returned as a clock vector.
- `file` is the name of the log file.
- `'P1',V1,'P2',V2,...` (optional) are the property name/property value pairs, which allow you to select the amount of data to retrieve, among other things (see below).

`daqread` returns data and time information in the same format as `getdata`. If data from multiple triggers is retrieved, each trigger is separated by a `NaN`.

You select the amount of data returned and the format of that data with the properties given below.
Table 8-1  daqread Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Samples</td>
<td>Specify the sample range.</td>
</tr>
<tr>
<td>Time</td>
<td>Specify the relative time range.</td>
</tr>
<tr>
<td>Triggers</td>
<td>Specify the trigger range.</td>
</tr>
<tr>
<td>Channels</td>
<td>Specify the channel range. Channel names can be specified as a cell array.</td>
</tr>
<tr>
<td>DataFormat</td>
<td>Specify the data format as doubles or native.</td>
</tr>
<tr>
<td>TimeFormat</td>
<td>Specify the time format as vector or matrix.</td>
</tr>
</tbody>
</table>

The Samples, Time, and Triggers properties are mutually exclusive. If none of these three properties is specified, then all the data is returned.

**Retrieving Event, Device Object, Channel, and Hardware Information**

You can retrieve event, device object, channel, and hardware information, along with data and time information, using the syntax shown below.

```plaintext
[data,time,abstime,events,daqinfo] = daqread('file','P1',V1,'P2',V2,...);
```

events is a structure containing event information associated with the logged data. The events retrieved depend on the value of the Samples, Time, or Triggers property. At a minimum, the trigger event associated with the selected data is returned. The entire event log is returned to events only if Samples, Time, or Triggers is not specified.

daqinfo is a structure that stores device object, channel, and hardware information in two fields: ObjInfo and HwInfo. ObjInfo is a structure containing property values for the device object and any channels it contains. The property values are returned in the same format as returned by get. HwInfo is a structure containing hardware information. The hardware information is identical to the information returned by daqhwinfо(obj).
Alternatively, you can return only object, channel, and hardware information with the command

\[
daqinfo = \	ext{daqread}('file','info');
\]

**Note** When you retrieve object information, the entire event log is returned to `daqinfo.ObjInfo.EventLog` regardless of the number of samples retrieved.

### Example: Logging and Retrieving Information

This example illustrates how to log information to a disk file and then retrieve the logged information to MATLAB using various calls to `daqread`.

A sound card is configured for stereo acquisition, data is logged to memory and to a disk file, four triggers are issued, and 2 seconds of data are collected for each trigger at a sampling rate of 8 kHz. You can run this example by typing `daqdoc8_1` at the MATLAB command line.

1. **Create a device object** — Create the analog input object `ai` for a sound card. The installed adaptors and hardware IDs are found with `daqhwinfo`.

   ```matlab
   ai = analoginput('winsound');
   \%ai = analoginput('nidaq','Dev1');
   \%ai = analoginput('mcc',1);
   ```

2. **Add channels** — Add two hardware channels to `ai`.

   ```matlab
   ch = addchannel(ai,1:2);
   \%ch = addchannel(ai,0:1); \% For NI and MCC
   ```

3. **Configure property values** — Define a 2 second acquisition for each trigger, set the trigger to repeat three times, and log information to the file `file00.daq`.

   ```matlab
   duration = 2; \% Two seconds of data for each trigger
   set(ai,'SampleRate',8000)
   ActualRate = get(ai,'SampleRate');
   set(ai,'SamplesPerTrigger',duration*ActualRate)
   set(ai,'TriggerRepeat',3)
   set(ai,'LogFileName','file00.daq')
   ```
set(ai,'LoggingMode','Disk&Memory')

4 Acquire data — Start ai, wait for ai to stop running, and extract all the data stored in the log file as sample-time pairs.

```matlab
start(ai)
[data,time] = daqread('file00.daq');
```

Plot the data and label the figure axes.

```matlab
subplot(211), plot(data)
title('Logging and Retrieving Data')
xlabel('Samples'), ylabel('Signal (Volts)')
subplot(212), plot(time,data)
xlabel('Time (seconds)'), ylabel('Signal (Volts)')
```

Make sure ai has stopped running before cleaning up the workspace.

```matlab
wait(ai,2)
```

5 Clean up — When you no longer need ai, you should remove it from memory and from the MATLAB workspace.

```matlab
delete(ai)
clear ai
```

Retrieving Data Based on Samples

You can retrieve data based on samples using the Samples property. To retrieve samples 1000 to 2000 for both sound card channels:

```matlab
[data,time] = daqread('file00.daq','Samples',[1000 2000]);
```

Plot the data and label the figure axes.

```matlab
subplot(211), plot(data);
xlabel('Samples'), ylabel('Signal (Volts)')
subplot(212), plot(time,data);
xlabel('Time (seconds)'), ylabel('Signal (Volts)')
```
Retrieving Data Based on Channels
You can retrieve data based on channels using the Channels property. To retrieve samples 1000 to 2000 for the second sound card channel:

```
[data,time] = daqread('file00.daq','Samples',[1000 2000],
'Channels',2);
```

Plot the data and label the figure axes.

```
subplot(211), plot(data);
xlabel('Samples'), ylabel('Signal (Volts)')
subplot(212), plot(time,data);
xlabel('Time (seconds)'); ylabel('Signal (Volts)')
```

Alternatively, you can retrieve data for the second sound card channel by specifying the channel name.

```
[data,time] = daqread('file00.daq','Samples',[1000 2000],
'Channels',{ 'Right'});
```

Retrieving Data Based on Triggers
You can retrieve data based on triggers using the Triggers property. To retrieve all the data associated with the second and third triggers for both sound card channels:

```
[data,time] = daqread('file00.daq','Triggers',[2 3]);
```

Plot the data and label the figure axes.

```
subplot(211), plot(data);
xlabel('Samples'), ylabel('Signal (Volts)')
subplot(212), plot(time,data);
xlabel('Time (seconds)'); ylabel('Signal (Volts)')
```

Retrieving Data Based on Time
You can retrieve data based on time using the Time property. Time must be specified in seconds and Time=0 corresponds to the first logged sample. To retrieve the first 25% of the data acquired for the first trigger:

```
[data,time] = daqread('file00.daq','Time',[0 0.5]);
```
Plot the data and label the figure axes.

```matlab
subplot(211), plot(data);
xlabel('Samples'), ylabel('Signal (Volts)')
subplot(212), plot(time, data);
xlabel('Time (seconds)'), ylabel('Signal (Volts)')
```

### Retrieving Event, Object, Channel, and Hardware Information

You can retrieve event, object, channel, and hardware information by specifying the appropriate arguments to `daqread`. For example, to retrieve all event information, you must return all the logged data.

```matlab
[data, time, abstime, events, info] = daqread('file00.daq');
{events.Type}
ans =
'Start' 'Trigger' 'Trigger' 'Trigger' 'Trigger' 'Stop'
```

If you retrieve part of the data, then only the events associated with the requested data are returned.

```matlab
[data, time, abstime, events, info] = daqread('file00.daq',
'Trigger',[1 3]);
{events.Type}
ans =
'Trigger' 'Trigger' 'Trigger'
```

You can retrieve the entire event log as well as object and hardware information by including `info` as an input argument to `daqread`.

```matlab
daqinfo = daqread('file00.daq','info')
daqinfo =
    ObjInfo: [1x1 struct]
    HwInfo: [1x1 struct]
```
To return the event log information:

```matlab
eventinfo = daqinfo.ObjInfo.EventLog
eventinfo =
6x1 struct array with fields:
    Type
    Data
```
softscope: The Data Acquisition Oscilloscope

The data acquisition Oscilloscope is an interactive graphical user interface (GUI) for streaming data into a display. The sections are as follows.

- Oscilloscope Overview (p. 9-3) - Associate hardware with the Oscilloscope and open the application
- Displaying Channels (p. 9-6) - Display hardware, math, and reference channels
- Channel Data and Properties (p. 9-15) - Scale channel data horizontally and vertically
- Triggering the Oscilloscope (p. 9-19) - Control how the data acquisition is initiated
- Making Measurements (p. 9-22) - Make measurements on acquired data using predefined or custom measurement types
- Exporting Data (p. 9-29) - Save channel data or measurements to the workspace, a figure, or a MAT-file
- Saving and Loading the Oscilloscope Configuration (p. 9-32) - Save and load the hardware configuration, the property values, and the state of the Oscilloscope

This examples in this chapter use Measurement Computing’s Demo-Board, which is installed with InstaCal or the Universal Library driver. The Demo-Board is a software simulation of an 8-channel, 16-bit analog input device. You can associate waveforms such as a sine wave or a
square wave, or input from a data file with the analog input channels. You can download InstaCal or the Universal Library driver from http://www.measurementcomputing.com.
Oscilloscope Overview

In this section...

“Opening the Oscilloscope” on page 9-3
“Hardware Configuration” on page 9-4

Opening the Oscilloscope

To open the Oscilloscope, create an analog input object for the Measurement Computing Demo-Board, add two hardware channels, and supply the object to the softscope function.

```matlab
ai = analoginput('mcc',0)
addchannel(ai,0:1)
softscope(ai)
```

As shown below, the Oscilloscope opens with a single display containing a marker for each added hardware channel, a channel scaling pane, and a trigger pane.
Note that you can also open the Oscilloscope by

- Typing `softscope` without any arguments and using the Hardware Configuration GUI to configure the hardware device.

- Supplying a configuration file as an input argument to `softscope`. Refer to “Saving and Loading the Oscilloscope Configuration” on page 9-32 for more information.

**Hardware Configuration**

If you type `softscope` without supplying an analog input object,

```
softscope
```

the Hardware Configuration GUI is opened, which allows you to select the hardware device to be used with the Oscilloscope.

The GUI shown below is configured to display the first two hardware channels of the `mcc` Demo-Board in the Oscilloscope. The channels are sampled at a rate of 5000 Hz and use the default input range. After you click the **OK** button, the Oscilloscope opens.

1. Set the sampling rate to 5000 Hz.
2. Display only the first two channels.
3. Click the **OK** button to open the Oscilloscope.
You can also open the Hardware Configuration GUI by selecting the **Edit > Hardware** menu item. You might want to do this to reconfigure an existing hardware device, or to select a new hardware device. Additionally you can change the sampling rate of the added channels with the New Sample Rate GUI, which is shown below. You open this GUI by selecting the **Edit > Sample Rate** menu item.

![New Sample Rate GUI](image)
Displaying Channels

In this section...

“Creating a Display” on page 9-6
“Creating Additional Displays” on page 9-7
“Configuring Display Properties” on page 9-9
“Math and Reference Channels” on page 9-10
“Removing Channel Displays” on page 9-13

Creating a Display

Click **Trigger** to begin streaming data into the display. The data from each channel defines a unique trace (line). To quickly scale the data, right-click the display and select **Autoscale** from the menu.

Display data tips by placing the mouse cursor over the trace.

Click the **Trigger** button to begin streaming data into the display.
The display area contains this information:

- Labels and markers for each trace. For this example, the traces are labeled CH0 and CH1.
- Labels for the vertical units for each trace, and a label for the horizontal units for the display.

When the acquisition is not running, you can display data tips by moving the mouse cursor over the trace. The data tip is indicated by a red circle, and displays the value of the trace at the selected point. If you press the Control key while the cursor is over the trace, the difference between the first data tip and the last data tip is displayed.

**Creating Additional Displays**

To add additional displays to the Oscilloscope, use the Scope pane of the Scope Editor GUI. To open this GUI, select Scope from the Edit menu. As shown below, the new display is named display2.
To show a trace in a particular display, use the **Channel Display** pane of the Channel Editor GUI. To open this GUI, select **Channel** from the **Edit** menu. As shown below, CH0 is associated with the new display.

The Oscilloscope is now configured so that the CH0 trace is shown in the bottom display, and the CH1 trace is shown in the top display.
Configuring Display Properties

You can change the display characteristics of the Oscilloscope by configuring display properties. You access the display properties these two ways:

- **Property Inspector** — Place the mouse cursor in the display of interest, right-click, and select **Edit Properties** from the menu.
- **Scope Editor GUI** — Select **Scope** from the **Edit** menu, and then choose the **Scope Properties** pane.

For this example, use the Scope Editor GUI to change the color of both displays to white. The steps are

1. Select both displays from the **Select the scope components** list.
2. Open the color picker for the **Color** property.
3. Select **White** from the color picker pop-up menu.

The **Scope Properties** pane and color picker are shown below. For descriptions of all display properties, click the **Help** button.

---

![Scope Editor GUI](image)

- **Select both displays.**
- **Open the color picker and select White from the pop-up menu.**
- **Click Help to view property descriptions.**
Math and Reference Channels

In addition to hardware channels, you can display

- Reference channels — The data associated with a reference channel is defined from a MATLAB variable or expression. You should use reference channel data as a known waveform against which other data is compared.

- Math channels — The data associated with a math channel is calculated in MATLAB using the data from existing hardware channels, math channels, or reference channels.

You use the Channel pane of the Channel Editor GUI to create math and reference channels. You open this GUI by selecting the Edit > Channel menu item. For example, suppose you want to create a reference waveform to compare to the CH0 waveform. The first step is to create the reference data in MATLAB:

```
t = 0:0.0001:0.2;
w = 200*2*pi;
ref = 3.75*sin(w*t);
```
The next step is to define the reference channel in the Channel Editor GUI. The Channel pane shown below is configured to create a reference channel called r1 using the data defined in the variable ref, and to display the reference channel data with CH0 in display2.

Note that instead of creating the variable ref in the workspace, you can specify the expression $3.75 \times \sin(w \times t)$ in the Expression field.

**Note** If the expression returns a complex value, only the real part of the value will be displayed.

Defining a math channel is similar to defining a reference channel. The main difference is in specifying the expression. For a reference channel, you specify a MATLAB variable or expression. For a math channel, you specify

- The channel name — Channel names are given by the Name column in the Defined channels table.
• A valid MATLAB expression — When the expression is evaluated, the channel names are replaced with the associated data that is currently being displayed.

The **Channel** pane shown below is configured to create a math channel called m1 using the CH0 and CH1 data, and to display the math channel data with CH1 in display1.

![Channel Editor](image)

- Define a math channel named m1 using the data from CH0 and CH1.
- Display the math channel data with CH1.
- Click the Add button to include the new channel in the table.
- Click the OK or Apply button to create the math channel.

The traces for the hardware, math, and reference channels are shown below.
Removing Channel Displays

You can remove a channel from a display one of these ways:

- **Channel Editor GUI**
  - The Channel pane — Clear the associated check box in the first column of the Defined channels table.
  - The Channel Display pane — Select <not displayed> from the Display column of the table.
- The On/Off button of the Channel Scaling pane. Refer to “Channel Data and Properties” on page 9-15 for more information about this pane.

The Channel pane is shown below with the math and reference channels cleared from the Oscilloscope displays.
Note that if you clear the check boxes, then in addition to the channels not being displayed:

- For hardware channels, data is not streamed into the Oscilloscope.
- For math and reference channels, the values are not calculated.
Channel Data and Properties

In this section...

“Scaling the Channel Data” on page 9-15
“Configuring Channel Properties” on page 9-16

Scaling the Channel Data
You can scale the defined channels using the Channel Scaling pane. In particular, you can modify

- The horizontal scaling and offset for all display components.
- The vertical scaling and offset for one or more channels. To simultaneously modify the vertical scaling for multiple channels, select the desired channel names in the list box.

Additionally, using the On/Off button, you can add or remove the selected traces from the Oscilloscope.

As shown below, the horizontal scale is changed to approximately 5 ms/div, and the vertical scale is modified to maximize the trace amplitudes. Note that the horizontal and vertical scaling information is shown at the bottom of each display component.
To specify a precise horizontal scale or offset, you modify the associated display properties. To specify a precise vertical scale or offset, you modify the associated channel properties. You can access these properties using the Scope Editor and the Channel Editor, respectively. You open these editors with the Edit menu or a right-click menu. Note that all displays use the same horizontal offset and scale.

**Configuring Channel Properties**

There are two sets of properties associated with the Channel Scaling pane:

- Channel pane properties — Properties associated with the controls and labels that make up the pane
- Channel properties — Properties associated with the hardware, math, and reference channels that are listed in the pane

For descriptions of all channel properties, click the Help button of the appropriate GUI editor.

**Channel Pane Properties**

You can change the characteristics of the controls and labels that make up the pane with the Scope Editor GUI. To open this GUI, select Scope from the
**Edit** menu, choose the **Scope Properties** pane, and select Channel Scaling from the **Select scope components** list box. The **Scope Properties** pane is shown below.

**Channel Properties**

You can change the characteristics of the hardware, math, and reference channels that are listed in the pane by configuring their channel properties. You can access the channel properties these two ways:

- **Property Inspector** — Place the mouse cursor in the Channel Scaling pane, right-click, and select **Edit Properties** from the menu.
- **Channel Editor GUI** — Select **Channel** from the **Edit** menu, and then choose the **Channel Properties** pane.

For this example, use the Channel Editor GUI to modify the marker characteristics for both CH0 and CH1. The steps are

1. Select both hardware channels from the **Select the channels** list box.
2. Specify a circular symbol for the Marker property, and specify an interval of 4 for the MarkerInterval property.

The **Channel Properties** pane is shown below.
Select both channels.

Specify a circular marker symbol.

Specify a marker interval of 4.

Click Help to view property descriptions.
Triggering the Oscilloscope

In this section...

“Acquisition Types” on page 9-19
“Trigger Types” on page 9-19
“Configuring Trigger Properties” on page 9-21

Acquisition Types

To display acquired data in the Oscilloscope, you must click the Trigger button. You control how the data acquisition is initiated by specifying the acquisition type and the trigger type in the Trigger pane.

The Oscilloscope supports three acquisition types, which you select from the Acquire menu:

- **One Shot** — Acquire the specified number of samples once.
- **Continuous** — Continuously acquire the specified number of samples.
- **Sequence** — Continuously acquire the specified number of samples, and use the dependent trigger type each time.

For each acquisition type, you can either fill the display with data or you can acquire a specific number of samples. Additionally, the specified trigger type determines how the acquisition is initiated.

Trigger Types

The Oscilloscope supports two trigger types, which you select from the Type menu:

- **Dependent** — Data acquisition depends on the data. You define this dependency by specifying the hardware channel, trigger condition, trigger condition value, and whether pretrigger data is acquired.

Note that you can specify a dependent trigger for only one channel at a time, and this channel initiates data acquisition for all other channels defined for the Oscilloscope.
- **Independent** — Data acquisition starts immediately after you press the Trigger button, and is independent of the data. Note that the Sequence acquisition does not support this trigger type.

The Oscilloscope shown below is configured for a one-shot acquisition of 1000 samples for CH0 and CH1. The acquisition is dependent on the data, and is initiated when a rising signal level of -3.3 volts is detected on CH0. Additionally, the first 0.02 second of data is defined as pretrigger data.

When you use a dependent trigger type, the display associated with the selected channel contains these two indicators:

- The trigger level on the vertical axis.
- The location of the start of the trigger on the horizontal axis. The start of the trigger corresponds to the first acquired sample at time zero. As shown by the data tips for CH1, data to the left of the indicator is defined as pretrigger data and has negative time values.

Note that you can change the indicator locations graphically by placing the mouse cursor over the indicator and sliding it to the desired location.
Configuring Trigger Properties

You can change the characteristics of the labels associated with the Triggers pane with the Scope Editor GUI. To open this GUI, select Scope from the Edit menu, choose the Scope Properties pane, and select Triggers from the Select the scope components list box. The Scope Properties pane is shown below.

Select Triggers.

Click Help to view property descriptions.
Making Measurements

In this section...

<table>
<thead>
<tr>
<th>Predefined Measurement” on page 9-22</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Defining a Measurement” on page 9-23</td>
</tr>
<tr>
<td>“Defining a New Measurement Type” on page 9-25</td>
</tr>
<tr>
<td>“Configuring Measurement Properties” on page 9-26</td>
</tr>
</tbody>
</table>

Predefined Measurement

You can make measurements on the acquired data with the Measurements pane. The Oscilloscope provides many predefined measurement types such as horizontal and vertical cursors, and basic math calculations such as the mean and standard deviation. Additionally, you can define new measurement types that suit your specific needs.

As shown below, you can list the predefined measurement types and create a new measurement type with the Measurement Type pane of the Measurement Editor GUI.
Defining a Measurement

Measurements that you define for the Oscilloscope are displayed in the Measurements pane. By default, this pane is not included as part of the Oscilloscope. To create the pane, you define one or more initial measurements. There are two ways to do this:

- Right-click in the Channel Scaling pane and select Add Measurement from the menu.
- Use the Measurement Editor GUI, which you open by selecting the Edit > Measurement menu item.

Alternatively, you can create an empty Measurements pane by selecting the Measurement check box in the Scope pane of the Scope Editor.

The Measurement pane shown below is configured to add a vertical cursor measurement for CH0 to the Oscilloscope. Note that the peak-to-peak measurement is already defined for CH0.
Select the channel and the measurement type.

Click Add to add the measurement to the table.

Click OK or Apply to add the measurement to the Oscilloscope.
After you click the **OK** or **Apply** button of the Measurement Editor, the **Measurements** pane is automatically added to the Oscilloscope. You can then click the **Add Measurement** button to define additional measurements.

![Oscilloscope Screenshot](image)

The vertical cursor.

To add a new measurement to the pane, click Add Measurement.

---

**Defining a New Measurement Type**

You define a new measurement type by defining a MATLAB function that takes an array of data as input and returns a scalar value. You can define a new measurement type these two ways:

- If the **Measurements** pane is displayed, select **New** from the **Type** menu.
- Use the **Measurement Type** pane of the Measurement Editor.
As shown below, a new measurement type that calculates the median is defined via the **Measurements** pane. The resulting measurement is the median value of the CH0 data.

### Configuring Measurement Properties

There are two sets of properties associated with measurements:

- **Measurement pane properties** — Properties associated with the pane label
- **Measurement properties** — Properties associated with the measurements that are listed in the pane

For descriptions of all measurement properties, click the **Help** button of the **Scope Properties** pane or the **Measurement Properties** pane.

### Measurement Panel Properties

You can change the characteristics of the pane label with the Scope Editor GUI. To open this GUI, select **Scope** from the **Edit** menu, choose the **Scope**
Properties pane, and select Measurements from the Select the scope components list box. The Scope Properties pane is shown below.

**Measurement Properties**

You can configure measurement properties with the Measurement Properties editor. You can open this editor two ways:

- Right-click menu — Place the mouse cursor in the Measurements pane of interest, right-click, and select Edit Properties from the menu.

- Measurement Editor GUI — Select Measurement from the Edit menu, and then choose the Measurement Properties pane.

For this example, use the Measurement Editor GUI to change the number of measurements stored for CH1 to be identical to the number of samples acquired for each trigger. The steps are:

1. Select CH0 - Pk2Pk in the Select the measurements list box.

2. Edit the BufferSize property to be 1000.
The **Measurement Properties** pane is shown below.

Select the CH0 peak-to-peak measurement.

Store 1000 measurements.
Exporting Data

In this section...

“Channels” on page 9-29
“Measurements” on page 9-30

Channels

You can export this information to the MATLAB workspace, a figure, or a MAT-file.

You export channel data with the Channel Exporter GUI, which you open by selecting the File > Export > Channels menu.

Channel data is data associated with a hardware channel, a math channel, or a reference channel.

The GUI shown below is configured to export 1000 samples for both hardware channels to the workspace as a structure, which contains horizontal and vertical scaling information. The variable name for the CH0 data is c0 and the variable name for the CH1 data is c1.
The saved structure is shown below, where \( t_0 \) is the time of the first stored sample. Note that the time is negative because pretrigger data was acquired.

\[
c_0
\]

\[
c_0 =
\begin{align*}
\text{horizontalScale:} & \ 0.0050 \\
\text{horizontalOffset:} & \ 0 \\
\text{verticalScale:} & \ 2.5730 \\
\text{verticalOffset:} & \ 0 \\
\text{data:} & \ [1000\times1 \ \text{double}] \\
\text{t0:} & \ -0.0200 \\
\text{_samplerate:} & \ 5000
\end{align*}
\]

**Measurements**

You can export this information to the MATLAB workspace, a figure, or a MAT-file.

You export measurement data with the Measurement Exporter GUI, which you open by selecting the `File > Export > Measurement` menu item.

Measurements data is associated with a defined measurement. Note that some measurements such as the horizontal and the vertical cursor have no data to save.

The GUI shown below is configured to export the peak-to-peak and absolute value measurements for CH0 to the workspace. The maximum number of measurements exported depends on the `BufferSize` property value for each measurement type. The variable name for the peak-to-peak measurement is \( m_0 \) and the variable name for the absolute value measurement is \( m_1 \).
Save the measurement data to the workspace.

Save the data for both measurements to the variable names m0 and m1.
Saving and Loading the Oscilloscope Configuration

You can save the Oscilloscope configuration to a softscope file. Softscope files are text-based files that contain this information:

- The hardware configuration
- The property values
- The screen position

You create a softscope file by selecting **Save** or **Save As** from the **File** menu. The Save Softscope dialog box is shown below.

To load a softscope file into the Oscilloscope, provide the file name as an argument to the `softscope` function.

```matlab
softscope('mcc.si')
```
Using the Data Acquisition Blocks in Simulink

Overview (p. 10-2) Describes the four Data Acquisition Toolbox blocks
Opening the Data Acquisition Block Library (p. 10-3) Describes how to access the block library
Building Simulink Models to Acquire Data from a Device (p. 10-6) Provides an example of using an Analog Input block in a model
Overview

This chapter describes how to use the Data Acquisition Toolbox block library. The toolbox block library contains four blocks:

- **Analog Input** — Acquire data from multiple analog channels of a data acquisition device.
- **Analog Output** — Output data to multiple analog channels of a data acquisition device.
- **Digital Input** — Acquire the latest set of values from multiple digital lines of a data acquisition device.
- **Digital Output** — Output data to multiple digital lines of a data acquisition device.

You can use these blocks to acquire analog or digital data in a Simulink® model, or to output analog or digital data from the model to a hardware device. You can interconnect these blocks with blocks in other Simulink libraries to create sophisticated models.

**Note** You must have a license for both Data Acquisition Toolbox and Simulink to use these blocks.

Use of the Data Acquisition Toolbox blocks requires Simulink, a tool for simulating dynamic systems. Simulink is a model definition and simulation environment. Use Simulink blocks to create a block diagram that represents the computations of your system or application. 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.

For more detailed information about the blocks in Data Acquisition Toolbox, see the reference pages for the blocks in Blocks — Alphabetical List.
Opening the Data Acquisition Block Library

In this section...

“Using the daqlib Command from MATLAB” on page 10-3
“Using the Simulink Library Browser” on page 10-3

Using the daqlib Command from MATLAB
To open the Data Acquisition Toolbox block library, enter

daqlib

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

Using the Simulink Library Browser
To open the Data Acquisition Toolbox block library, start the Simulink Library Browser and select 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 lists available block libraries, with the basic Simulink library listed first, followed by other libraries listed in alphabetical order under it. To open the Data Acquisition Toolbox block library, click its icon.
Opening the Data Acquisition Block Library

**Simulink Library Browser**

**Analog Input:** Acquire data from multiple channels of a data acquisition device.
Building Simulink Models to Acquire Data from a Device

In this section...

| “Data Acquisition Toolbox Library” on page 10-6 |
| “Example: Bringing Analog Data into a Model” on page 10-6 |

Data Acquisition Toolbox Library

This section provides an example that builds a simple model using the block in conjunction with a block from another block library. It illustrates how to bring live analog data into Simulink from a data acquisition device, in this case a sound card.

Example: Bringing Analog Data into a Model

Step 1: Open the Data Acquisition Toolbox Block Library

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

To start the Simulink Library Browser, enter

    simulink

at the MATLAB prompt. In the Simulink Library Browser, the left pane lists the available block libraries. To open the Data Acquisition Toolbox block library, click its icon.

Step 2: Create a New Model

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

Create a new model by clicking the Create a new model button in the Simulink Library Browser.
Step 3: Add the Analog Input Block to the Model
To use the Analog Input block in a model, click the block in the library and, holding the left mouse button down, drag it into the Model window. Note how the name on the block changes to reflect the first available analog device connected to your system.
Using the Data Acquisition Blocks in Simulink
Step 4: Add a Scope to the Model

To illustrate using the block, this example creates a simple model that acquires analog data from a microphone, via a sound card (the analog device), and then outputs the data to a scope, where you can see the intensity of the sound waves. To create this model, this example uses a Scope block from the basic Simulink block library.

Expand the Simulink block library by clicking **Simulink** at the top of the library list, if it is not already open. In the library window, open the **Sinks** group. From this group, click the Scope block in the library and, holding the left mouse button down, drag the block into the Model window.
Using the Data Acquisition Blocks in Simulink

Simulink Library Browser

Scope: simulink/Sinks/Scope

Analog Input Model:

winSound 0
NV10/IP(R) n...
8000 samples/sec

Analog Input

Scope
Step 5: Specify Block Parameters
To specify Analog Input block parameter settings, double-click the block's icon in the Model window. This opens the Source Block Parameters dialog box for the Analog Input block, shown in the following figure. Use the various fields to determine the current values of the Analog Input block parameters or to change the values.
Using the Data Acquisition Blocks in Simulink

Source Block Parameters: Analog Input

Acquire data from multiple channels of a data acquisition device.

Parameters:

Acquisition Mode
- Asynchronous - Initiates the acquisition when simulation starts. The simulation runs while data is acquired into a FIFO buffer.
- Synchronous - Initiates the acquisition at each time step. The simulation will not continue until all data is acquired.

Device: winsound 0 (NVIDIA(R) nForce(TM) Audio)

Hardware sample rate [samples/second]: 8000

**Actual rate will be 8000 samples per second.**

Block size: 5

Input type: AC-Coupled

Channels:

<table>
<thead>
<tr>
<th>Hardware Channel</th>
<th>Name</th>
<th>Input Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>1</td>
<td>-1V to +1V</td>
</tr>
<tr>
<td>✔</td>
<td>2</td>
<td>-1V to +1V</td>
</tr>
</tbody>
</table>

Outputs:

Number of ports: 1 for all hardware channels

Signal type: Sample-based

Data type: double

[OK] [Cancel] [Help]
In this example, keep the default settings for everything except Block size. Change the block size setting to 5, which means five samples will be acquired from each channel at every time step. As you can see in the dialog box, the acquisition will be asynchronous, and the left and right channels will both use the same port, since the 1 for all hardware channels option is selected for Number of ports.

After changing the block size to 5, click OK to close the dialog box. For more information on the options and the Analog Input block, see the Analog Input Block reference page.

**Step 6: Connect the Blocks**
Connect the output from the Analog Input block to the Scope. Use the cursor in the model to drag a connection from the port of the Analog Input block to the scope.
**Step 7: Run the Simulation**
Before running the simulation, change the run time to 20 seconds by editing the default of 10 seconds in the Model window toolbar.

Open the scope by double-clicking the Scope block in the model. You will see live sound waves in the scope when the model is running.

Run the simulation by clicking the **Start simulation** toolbar button. During the 20 seconds that the simulation is running, speak into the microphone.

While the simulation is running, the status bar at the bottom of the Model window indicates the progress of the simulation. If you are speaking into the microphone, you will also see the live sound data plotted in the scope.

**Step 8: Look at the Data in the Scope**
When the 20 seconds elapses, the model stops running and you will have 20 seconds of sound data displayed by the scope. Click the **Autoscale** toolbar
button (binoculars icon) in the scope to see the portion of the collected data that has the most contrast or significance. It will look something like this:

Note in the above example that words were spoken into the microphone between the 7th and 8th second, and only ambient sound is picked up between the 9th and 10th second.
In the following example, you can see that the volume of the sound peaked around the 18th second, when shouting was picked up by the microphone.

For more information about the four Data Acquisition Toolbox blocks, including the Analog Input block shown in this chapter, see Blocks — Alphabetical List.
The functions and the device objects they are associated with are categorized according to usage as shown below. The supported device objects include analog input (AI), analog output (AO), and digital I/O (DIO). The tables indicate which object types each function supports.

- **Creating Device Objects (p. 11-2)**: Functions related to creating device objects
- **Adding Channels and Lines (p. 11-3)**: Functions related to adding hardware channels
- **Getting and Setting Properties (p. 11-4)**: Functions related to displaying properties
- **Executing the Object (p. 11-5)**: Functions related to executing device objects
- **Working with Data (p. 11-6)**: Functions related to working with acquired data
- **Getting Information and Help (p. 11-7)**: Functions related to displaying help information
- **General Purpose (p. 11-8)**: Functions related to general Data Acquisition Toolbox functionality
Creating Device Objects

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>AI</th>
<th>AO</th>
<th>DIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>analoginput</td>
<td>Create analog input object</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>analogoutput</td>
<td>Create analog output object</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>digitalio</td>
<td>Create digital I/O object</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
## Adding Channels and Lines

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>AI</th>
<th>AO</th>
<th>DIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>addchannel</td>
<td>Add hardware channels to analog input or analog output object</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>addline</td>
<td>Add hardware lines to digital I/O object</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>addmuxchannel</td>
<td>Add hardware channels when using multiplexer board (National Instruments only)</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Getting and Setting Properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>AI</th>
<th>AO</th>
<th>DIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>get</td>
<td>Return device object properties</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>inspect</td>
<td>Open Property Inspector</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>set</td>
<td>Configure or display device object properties</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>setverify</td>
<td>Configure and return specified property</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
## Executing the Object

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>AI</th>
<th>AO</th>
<th>DIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>islogging</td>
<td>Determine if analog input object is logging data</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>isrunning</td>
<td>Determine if device object is running</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>issending</td>
<td>Determine if analog output object is sending data</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>start</td>
<td>Start device object</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>stop</td>
<td>Stop device object</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>trigger</td>
<td>Manually execute trigger</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>wait</td>
<td>Wait for device object to stop running</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
## Working with Data

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>AI</th>
<th>AO</th>
<th>DIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>flushdata</td>
<td>Remove data from data acquisition engine</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getdata</td>
<td>Extract data, time, and event information from data acquisition engine</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getsample</td>
<td>Immediately acquire one sample</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getvalue</td>
<td>Read values from lines</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>peekdata</td>
<td>Preview most recent acquired data</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>putdata</td>
<td>Queue data in engine for eventual output</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>putsample</td>
<td>Immediately output one sample</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>putvalue</td>
<td>Write values to lines</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
## Getting Information and Help

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>AI</th>
<th>AO</th>
<th>DIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>daqhelp</td>
<td>Display help for device objects, constructors, adaptors, functions, and properties</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>daqhwinfo</td>
<td>Display data acquisition hardware information</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>propinfo</td>
<td>Return property characteristics for device objects, channels, or lines</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
# General Purpose

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
<th>AI</th>
<th>AO</th>
<th>DIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>binvec2dec</td>
<td>Convert binary vector to decimal value</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>clear</td>
<td>Remove device objects from MATLAB workspace</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>daqcallback</td>
<td>Callback function that displays event information for specified event</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>daqfind</td>
<td>Return device objects, channels, or lines from data acquisition engine to MATLAB workspace</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>daqmem</td>
<td>Allocate or display memory resources</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>daqread</td>
<td>Read Data Acquisition Toolbox (.daq) file</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>daqregister</td>
<td>Register or unregister hardware driver adaptor</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>daqreset</td>
<td>Remove device objects, engine MEX-file, and adaptor DLLs from memory</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dec2binvec</td>
<td>Convert decimal value to binary vector</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>delete</td>
<td>Remove device objects, channels, or lines from data acquisition engine</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>disp</td>
<td>Display summary information for device objects, channels, or lines</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ischannel</td>
<td>Check for channels</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>isdioline</td>
<td>Check for lines</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Function</td>
<td>Purpose</td>
<td>AI</td>
<td>AO</td>
<td>DIO</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------</td>
<td>----</td>
<td>----</td>
<td>-----</td>
</tr>
<tr>
<td><code>isvalid</code></td>
<td>Determine if device objects, channels, or lines are valid</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><code>length</code></td>
<td>Return length of device object, channel group, or line group</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><code>load</code></td>
<td>Load device objects, channels, or lines into MATLAB workspace</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><code>makenames</code></td>
<td>Generate list of descriptive channel or line names</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><code>muxchanidx</code></td>
<td>Return multiplexed scanned channel index (National Instruments only)</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>obj2mfile</code></td>
<td>Convert device objects, channels, or lines to MATLAB code</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><code>save</code></td>
<td>Save device objects to MAT-file</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><code>showdaqevents</code></td>
<td>Display event log information</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><code>size</code></td>
<td>Return size of device object, channel group, or line group</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Functions — Alphabetical List
**Purpose**

Add hardware channels to analog input or analog output object

**Syntax**

\[
\text{chans} = \text{addchannel}(\text{obj}, \text{hwch})
\]

\[
\text{chans} = \text{addchannel}(\text{obj}, \text{hwch}, \text{index})
\]

\[
\text{chans} = \text{addchannel}(\text{obj}, \text{hwch}, 'names')
\]

\[
\text{chans} = \text{addchannel}(\text{obj}, \text{hwch}, \text{index}, 'names')
\]

**Arguments**

- **obj**: An analog input or analog output object.
- **hwch**: Specifies the numeric IDs of the hardware channels added to the device object. Any MATLAB vector syntax can be used.
- **index**: The MATLAB indices to associate with the hardware channels. Any MATLAB vector syntax can be used provided the vector elements are monotonically increasing.
- **'names'**: A descriptive channel name or cell array of descriptive channel names.
- **chans**: A column vector of channels with the same length as hwch.

**Description**

\[
\text{chans} = \text{addchannel}(\text{obj}, \text{hwch})
\]

adds the hardware channels specified by hwch to the device object obj. The MATLAB indices associated with the added channels are assigned automatically. chans is a column vector of channels.

\[
\text{chans} = \text{addchannel}(\text{obj}, \text{hwch}, \text{index})
\]

adds the hardware channels specified by hwch to the device object obj. index specifies the MATLAB indices to associate with the added channels.

\[
\text{chans} = \text{addchannel}(\text{obj}, \text{hwch}, 'names')
\]

adds the hardware channels specified by hwch to the device object obj. The MATLAB indices associated with the added channels are assigned automatically. names is a descriptive channel name or cell array of descriptive channel names.

\[
\text{chans} = \text{addchannel}(\text{obj}, \text{hwch}, \text{index}, 'names')
\]

adds the hardware channels specified by hwch to the device object obj. index specifies
the MATLAB indices to associate with the added channels. names is a descriptive channel name or cell array of descriptive channel names.

### Remarks

#### Rules for Adding Channels

- The numeric values you supply for hwch depend on the hardware you access. For National Instruments and Measurement Computing hardware, channels are “zero-based” (begin at zero). For Agilent Technologies hardware and sound cards, channels are “one-based” (begin at one).

- Hardware channel IDs are stored in the HwChannel property and the associated MATLAB indices are stored in the Index property.

- You can add individual hardware channels to multiple device objects.

- For sound cards and Agilent Technologies devices, you cannot add a hardware channel multiple times to the same device object.

- For Agilent Technologies devices, added channels must be in increasing order.

- You can configure sound cards in one of two ways: mono mode or stereo mode. For mono mode, hwch must be 1. For stereo mode, the first hwch value specified must be 1.

---

**Note** If you are using National Instruments AMUX-64T multiplexer boards, you must use the addedchannelfunction to add channels.

### More About MATLAB Indices

Every hardware channel contained by a device object has an associated MATLAB index that is used to reference the channel. Index assignments are made either automatically by addchannel or explicitly with the index argument and follow these rules:

- If index is not specified and no hardware channels are contained by the device object, then the assigned indices automatically start at one
and increase monotonically. If hardware channels have already been added to the device object, then the assigned indices automatically start at the next highest index value and increase monotonically.

- If index is specified but the indices are previously assigned, then the requested assignment takes precedence and the previous assignment is reindexed to the next available values. If the lengths of \( \text{hwch} \) and index are not equal, then an error is returned and no channels are added to the device object.

- The resulting indices begin at one and increase monotonically up to the size of the channel group.

- If you are using scanning hardware, then the indices define the scan order.

- Sound cards cannot be reindexed.

**Rules for Adding Channels to National Instruments 1200 Series Boards**

When using National Instruments 1200 Series hardware, you need to modify the above rules in these ways:

- Channel IDs are given in reverse order with \( \text{addchannel} \). For example, to add eight single-ended channels to the analog input object ai:

\[
\text{addchannel(ai,7:-1:0)};
\]

- The scan order is from the highest ID to the lowest ID (which must be 0).

- There cannot be any gaps in the channel group.

- When channels are configured in differential mode, the hardware IDs are 0, 2, 4, and 6.

**More About Descriptive Channel Names**

You can assign hardware channels descriptive names, which are stored in the \( \text{ChannelName} \) property. Choosing a unique descriptive name can
be a useful way to identify and reference channels. For a single call to
`addchannel`, you can

- Specify one channel name that applies to all channels that are to
  be added
- Specify a different name for each channel to be added

If the number of names specified in a single `addchannel` call is more
than one but not equal to the number of channels to be added, then an
error is returned. If a channel is to be referenced by its name, then that
name must not contain symbols. If you are naming a large number of
channels, then the `makenames` function might be useful. If a channel is
not assigned a descriptive name, then it must be referenced by index.

A sound card configured in mono mode is automatically assigned
the name `Mono`, while a sound card configured in stereo mode is
automatically assigned the names `Left` for the first channel and `Right`
for the second channel. You can change these default channel names
when the device object is created, or any time after the channel is added.

**Examples**

**National Instruments**

Suppose you create the analog input object `AI1` for a National
Instruments board, and add the first four hardware channels (channels
0-3) to it.

```matlab
AI1 = analoginput('nidaq','Dev1');
addchannel(AI1,0:3);
```

The channels are automatically assigned the indices 1-4. If you want
to add the first four hardware channels to `AI1` and assign descriptive
names to the channels,

```matlab
addchannel(AI1,0:3,{'chan1','chan2','chan3','chan4'});
```

Note that you can use the `makenames` function to create a cell array of
channel names. If you add channels 4, 5, and 7 to the existing channel
group,
addchannel(AI1,[4 5 7]);

the new channels are automatically assigned the indices 5-7. Suppose instead you add channels 4, 5, and 7 to the channel group and explicitly assign them indices 1-3.

addchannel(AI1,[4 5 7],1:3);

The new channels are assigned the indices 1-3, and the previously defined channels are reindexed as indices 4-7. However, if you assigned channels 4, 5, and 7 to indices 6-8, an error is returned because there is a gap in the indices (index 5 has no associated hardware channel).

**Sound Card**

Suppose you create the analog input object AI1 for a sound card. Most sound cards have only two channels that can be added to a device object. To configure the sound card to operate in mono mode, you must specify `hwch` as 1.

AI1 = analoginput('winsound');
addchannel(AI1,1);

The `ChannelName` property is automatically assigned the value `Mono`. You can now configure the sound card to operate in stereo mode by adding the second channel.

addchannel(AI1,2);

The `ChannelName` property is assigned the values `Left` and `Right` for the two hardware channels. Alternatively, you can configure the sound card to operate in stereo mode with one call to `addchannel`.

addchannel(AI1,1:2);
See Also

Functions

delete, makenames

Properties

ChannelName, HwChannel, Index
Purpose

Add hardware lines to digital I/O object

Syntax

\[
\begin{align*}
\text{lines} &= \text{addline}(\text{obj}, \text{hwline}, \text{'direction'}) \\
\text{lines} &= \text{addline}(\text{obj}, \text{hwline}, \text{port}, \text{'direction'}) \\
\text{lines} &= \text{addline}(\text{obj}, \text{hwline}, \text{'direction'}, \text{'names'}) \\
\text{lines} &= \text{addline}(\text{obj}, \text{hwline}, \text{port}, \text{'direction'}, \text{'names'})
\end{align*}
\]

Arguments

- **obj**
  - A digital I/O object.

- **hwline**
  - The numeric IDs of the hardware lines added to the device object. Any MATLAB vector syntax can be used.

- **'direction'**
  - The line directions can be In or Out, and can be specified as a single value or a cell array of values.

- **port**
  - The numeric IDs of the digital I/O port.

- **'names'**
  - A descriptive line name or cell array of descriptive line names.

- **lines**
  - A column vector of lines with the same length as hwline.

Description

- \(\text{lines} = \text{addline}(\text{obj}, \text{hwline}, \text{'direction'})\) adds the hardware lines specified by hwline to the digital I/O object obj. *direction* configures the lines for either input or output. lines is a row vector of lines.

- \(\text{lines} = \text{addline}(\text{obj}, \text{hwline}, \text{port}, \text{'direction'})\) adds the hardware lines specified by hwline from the port specified by port to the digital I/O object obj.

- \(\text{lines} = \text{addline}(\text{obj}, \text{hwline}, \text{'direction'}, \text{'names'})\) adds the hardware lines specified by hwline to the digital I/O object obj. names is a descriptive line name or cell array of descriptive line names.

- \(\text{lines} = \text{addline}(\text{obj}, \text{hwline}, \text{port}, \text{'direction'}, \text{'names'})\) adds the hardware lines specified by hwline from the port specified by port to the digital I/O object obj. *direction* configures the lines for either
input or output. names is a descriptive line name or cell array of descriptive line names.

**Remarks**

**Rules for Adding Lines**

- The numeric values you supply for hwline depend on the hardware you access. For National Instruments and Measurement Computing hardware, line IDs are “zero-based” (begin at zero).

- You can add a line only once to a given digital I/O object.

- Hardware line IDs are stored in the HwLine property and the associated MATLAB indices are stored in the Index property.

- For a single call to addline, you can add multiple lines from one port or the same line ID from multiple ports. You cannot add multiple lines from multiple ports.

- If a port ID is not explicitly referenced, lines are added first from port 0, then from port 1, and so on.

- You can specify the line directions as a single value or a cell array of values. If a single direction is specified, then all added lines have that direction. If supported by the hardware, you can configure individual lines by supplying a cell array of directions.

**More About MATLAB Indices**

Every hardware line contained by a device object has an associated MATLAB index that is used to reference the line. Index assignments are made automatically by addline and follow these rules:

- If no hardware lines are contained by the device object, then the assigned indices automatically start at one and increase monotonically. If hardware lines have already been added to the device object, then the assigned indices automatically start at the next highest index value and increase monotonically.

- The resulting indices begin at one and increase monotonically up to the size of the line group.
• The first indexed line represents the least significant bit (LSB) and the highest indexed line represents the most significant bit (MSB).

**More About Descriptive Line Names**

You can assign hardware lines descriptive names, which are stored in the `LineName` property. Choosing a unique descriptive name can be a useful way to identify and reference lines. For a single call to `addline`, you can

- Specify one line name that applies to all lines that are to be added
- Specify a different name for each line to be added

If the number of names specified in a single `addline` call is more than one but differs from the number of lines to be added, then an error is returned. If a line is to be referenced by its name, then that name must not contain symbols. If you are naming a large number of lines, then the `makenames` function might be useful. If a line is not assigned a descriptive name, then it must be referenced by index.

**Example**

Create the digital I/O object `dio` and add the first four hardware lines (line IDs 0-3) from port 0.

```matlab
dio = digitalio('nidaq','Dev1');
addline(dio,0:3,'in');
```

These lines are automatically assigned the indices 1-4. If you want to add the first four hardware lines to `dio` and assign descriptive names to the lines,

```matlab
addline(dio,0:3,'in',{'line1','line2','line3','line4'});
```

Note that you can use the `makenames` function to create a cell array of line names. You can add the first four hardware lines (line IDs 0-3) from port 1 to the existing line group.

```matlab
addline(dio,0:3,1,'out');
```
The new lines are automatically assigned the indices 5-8.

**See Also**

**Functions**
delete, makenames

**Properties**
HwLine, Index, LineName
**addmuxchannel**

**Purpose**
Add hardware channels when using multiplexer board

**Syntax**
```matlab
addmuxchannel(obj)
addmuxchannel(obj,chanids)
chans = addmuxchannel(...)
```

**Arguments**
- `obj` An analog input object associated with a National Instruments Traditional NI-DAQ board.
- `chanids` The hardware channel IDs.
- `chans` The channels that are added to `obj`.

**Description**
`addmuxchannel(obj)` adds as many channels to `obj` as is physically possible based on the number of National Instruments AMUX-64T multiplexer (mux) boards specified by the `NumMuxBoards` property. For one mux board, 64 channels are added. For two mux boards, 128 channels are added. For four mux boards, 256 channels are added.

`addmuxchannel(obj,chanids)` adds the channels specified by `chanids` to `obj`. `chanids` refers to the hardware channel IDs of the data acquisition board.

The actual number of channels added to `obj` depends on the number of mux boards used. For example, suppose you are using a data acquisition board with 16 channels connected to one mux board. If `chanid` is 0, then `addmuxchannel` adds four channels. Refer to the *AMUX-64T User Manual* for more information about adding mux channels based on hardware channel IDs and the number of mux boards used.

`chans = addmuxchannel(...)` returns the channels added to `chans`.

**Remarks**
This function is not available for National Instruments NI-DAQmx boards.

Before using `addmuxchannel`, you must set the `NumMuxBoards` property to the appropriate value. You can use as many as four mux boards with...
one analog input object. `addmuxchannel` deletes all channels contained by `obj` before new channels are added.

See Also

See Also

Functions

`muxchanidx`
analoginput

**Purpose**
Create analog input object

**Syntax**

\[
\begin{align*}
\text{AI} &= \text{analoginput('adaptor')} \\
\text{AI} &= \text{analoginput('adaptor',ID)}
\end{align*}
\]

**Arguments**

- `'adaptor'`  The hardware driver adaptor name. The supported adaptors are advantech, hpe1432, keithley, mcc, nidaq, and winsound.
- `ID`  The hardware device identifier. ID is optional if the device object is associated with a sound card having an ID of 0.
- `AI`  The analog input object.

**Description**

\[
\begin{align*}
\text{AI} &= \text{analoginput('adaptor')} \text{ creates the analog input object AI for a sound card having an ID of 0 (adaptor must be winsound). This is the only case where ID is not required.} \\
\text{AI} &= \text{analoginput('adaptor',ID)} \text{ creates the analog input object AI for the specified adaptor and for the hardware device with device identifier ID. ID can be specified as an integer or a string.}
\end{align*}
\]

**Remarks**

**More About Creating Analog Input Objects**

- When an analog input object is created, it does not contain any hardware channels. To execute the device object, hardware channels must be added with the addchannel function.
- You can create multiple analog input objects that are associated with a particular analog input subsystem. However, you can typically execute only one object at a time.
- The analog input object exists in the data acquisition engine and in the MATLAB workspace. If you create a copy of the device object, it references the original device object in the engine.
• If ID is a numeric value, then you can specify it as an integer or a string. If ID contains any nonnumeric characters, then you must specify it as a string (see the Agilent Technologies example below).

• The Name property is automatically assigned a descriptive name that is produced by concatenating adaptor, ID, and -AI. You can change this name at any time.

**More About the Hardware Device Identifier**

When data acquisition devices are installed, they are assigned a unique number which identifies the device in software. The device identifier is typically assigned automatically and can usually be manually changed using a vendor-supplied device configuration utility. National Instruments refers to this number as the device number while Agilent Technologies refers to it as the device ID.

For sound cards, the device identifier is typically not exposed to you through the Microsoft Windows environment. However, Data Acquisition Toolbox automatically associates each sound card with an integer ID value. There are two cases to consider:

• If you have one sound card installed, then ID is 0. You are not required to specify ID when creating an analog input object associated with this device.

• If you have multiple sound cards installed, the first one installed has an ID of 0, the second one installed has an ID of 1, and so on. You must specify ID when creating analog input objects associated with devices not having an ID of 0.

There are two ways you can determine the ID for a particular device:

• Type daqhwinfo('adaptor').
• Execute the vendor-supplied device configuration utility.
**Example**

### National Instruments

To create an analog input object for a National Instruments board defined as device number 1:

\[
AI = \text{analoginput('nidaq','Dev1')};
\]

### Agilent Technologies

To create an analog input object for an Agilent Technologies module with device identifier 1 residing in VXI chassis 0:

\[
AI = \text{analoginput('hpe1432','vxi0::1::instr')};
\]

Alternatively, you can use the syntax

\[
AI = \text{analoginput('hpe1432',1,0)};
\]

The HP driver allows you to span multiple hardware devices. To create an analog input object that spans two HP devices with device identifiers 1 and 2 residing in VXI chassis 0:

\[
AI = \text{analoginput('hpe1432','vxi0::1,2::instr')};
\]

Alternatively, you can use the syntax

\[
AI = \text{analoginput('hpe1432',[1,2],0)};
\]

**See Also**

### Functions

addchannel, daqhwinf

### Properties

Name
Purpose
Create analog output object

Syntax
AO = analogoutput('adaptor')
AO = analogoutput('adaptor',ID)

Arguments
'adaptor' The hardware driver adaptor name. The supported adaptors are advantech, hpe1432, keithley, mcc, nidaq, and winsound.
ID The hardware device identifier. ID is optional if the device object is associated with a sound card having an ID of 0.
AO The analog output object.

Description
AO = analogoutput('adaptor') creates the analog output object AO for a sound card having an ID of 0 (adaptor must be winsound). This is the only case where ID is not required.
AO = analogoutput('adaptor',ID) creates the analog output object AO for the specified adaptor and for the hardware device with device identifier ID. ID can be specified as an integer or a string.

Remarks
More About Creating Analog Output Objects

- When an analog output object is created, it does not contain any hardware channels. To execute the device object, hardware channels must be added with the addchannel function.
- You can create multiple analog output objects that are associated with a particular analog output subsystem. However, you can typically execute only one object at a time.
- The analog output object exists in the data acquisition engine and in the MATLAB workspace. If you create a copy of the device object, it references the original device object in the engine.
• If ID is a numeric value, then you can specify it as an integer or a string. If ID contains any nonnumeric characters, then you must specify it as a string (Agilent Technologies example).

• The Name property is automatically assigned a descriptive name that is produced by concatenating adaptor, ID, and -AO. You can change this name at any time.

More About the Hardware Device Identifier

When data acquisition devices are installed, they are assigned a unique number which identifies the device in software. The device identifier is typically assigned automatically and can usually be manually changed using a vendor-supplied device configuration utility. National Instruments refers to this number as the device number while Agilent Technologies refers to it as the device ID.

For sound cards, the device identifier is typically not exposed to you through the Microsoft Windows environment. However, Data Acquisition Toolbox automatically associates each sound card with an integer ID value. There are two cases to consider:

• If you have one sound card installed, then ID is 0. You are not required to specify ID when creating an analog output object associated with this device.

• If you have multiple sound cards installed, the first one installed has an ID of 0, the second one installed has an ID of 1, and so on. You must specify ID when creating analog output objects associated with devices not having an ID of 0.

There are two ways you can determine the ID for a particular device:

• Type daqhwinfo('adaptor').

• Execute the vendor-supplied device configuration utility.
**Example**

**National Instruments**

To create an analog output object for a National Instruments board defined as device number 1:

```matlab
AO = analogoutput('nidaq','Dev1');
```

**Agilent Technologies**

To create an analog output object for an Agilent Technologies module with device identifier 1 residing in VXI chassis 0:

```matlab
AO = analogoutput('hpe1432','vxi0::1::instr');
```

Alternatively, you can use the syntax

```matlab
AO = analogoutput('hpe1432',1,0);
```

The HP driver allows you to span multiple hardware devices. To create an analog output object that spans two HP devices with device identifiers 1 and 2 residing in VXI chassis 0:

```matlab
AO = analogoutput('hpe1432','vxi0::1,2::instr');
```

Alternatively, you can use the syntax

```matlab
AO = analogoutput('hpe1432',[1,2],0);
```

**See Also**

**Functions**

addchannel, daqhwinfo

**Properties**

Name
Purpose
Convert binary vector to decimal value

Syntax
out = binvec2dec(bin)

Arguments
bin    A binary vector.
out    A double array.

Description
out = binvec2dec(bin) converts the binary vector bin to the equivalent decimal number and stores the result in out. All nonzero binary vector elements are interpreted as a 1.

Remarks
A binary vector (binvec) is constructed with the least significant bit (LSB) in the first column and the most significant bit (MSB) in the last column. For example, the decimal number 23 is written as the binvec value [1 1 1 0 1].

Note The binary vector cannot exceed 52 values.

Examples
To convert the binvec value [1 1 1 0 1] to a decimal value:

    binvec2dec([1 1 1 0 1])
    ans =
        23

See Also Functions
dec2binvec
**Purpose**
Remove device objects from MATLAB workspace

**Syntax**
clear obj
clear obj.Channel(index)
clear obj.Line(index)

**Arguments**
- obj: A device object or array of device objects.
- obj.Channel(index): One or more channels contained by obj.
- obj.Line(index): One or more lines contained by obj.

**Description**
clear obj removes obj and all associated channels or lines from the MATLAB workspace, but not from the data acquisition engine.
clear obj.Channel(index) removes the specified channels contained by obj from the MATLAB workspace, but not from the data acquisition engine.
clear obj.Line(index) removes the specified lines contained by obj from the MATLAB workspace, but not from the data acquisition engine.

**Remarks**
Clearing device objects, channels, and lines follows these rules:

- clear does not remove device objects, channels, or lines from the data acquisition engine. Use the delete function for this purpose.
- If multiple references to a device object exist in the workspace, clearing one reference will not invalidate the remaining references.
- You can restore cleared device objects to the MATLAB workspace with the daqfind function.

If you use the help command to display the M-file help for clear, then you must supply the pathname shown below.

clear

help daq/private/clear
Examples

Create the analog input object ai, copy ai to a new variable aicopy, and then clear the original device object from the MATLAB workspace.

```matlab
ai = analoginput('winsound');
ch = addchannel(ai,1:2);
aicopy = ai;
clear ai
```

Retrieve ai from the engine with daqfind, and demonstrate that ai is identical to aicopy.

```matlab
ainew = daqfind;
isequal(aicopy,ainew)
ans =
   1
```

See Also

Functions

daqfind, delete
Purpose
Callback function that displays event information for specified event.

Syntax
daqcallback(obj,event)

Arguments
obj A device object.
event A variable that captures the event information contained by the EventLog property.

Description
daqcallback(obj,event) is an example callback function that displays information to the MATLAB command window. For all events, the information includes the event type and the name of the device object that caused the event to occur. For events that record the absolute time in EventLog, the event time is also displayed. For run-time error events, the error message is also displayed.

Remarks
You specify daqcallback as the callback function to be executed for any event by specifying it as the value for the associated callback property. For analog input objects, daqcallback is the default value for the DataMissedFcn and RuntimeErrorFcn properties. For analog output objects, daqcallback is the default value for the RuntimeErrorFcn property.

You can use the showdaqevents function to easily display event information captured by the EventLog property.

Examples
Create the analog input object ai and call daqcallback when a trigger event occurs.

```matlab
ai = analoginput('winsound');
addchannel(ai,1);
set(ai,'TriggerRepeat',3)
set(ai,'TriggerFcn',@daqcallback)
start(ai)
```
See Also

Functions

showdaqevents

Properties

DataMissedFcn, EventLog, RuntimeErrorFcn
Purpose

Return device objects, channels, or lines from data acquisition engine to MATLAB workspace

Syntax

out = daqfind
out = daqfind('PropertyName',PropertyValue,...)
out = daqfind(S)
out = daqfind(obj,'PropertyName',PropertyValue,...)

Arguments

'PropertyName' A device object, channel, or line property name.
PropertyValue A device object, channel, or line property value.
obj A device object, array of device objects, channels, or lines.
S A structure with field names that are property names and field values that are property values.
out An array or cell array of device objects, channels, or lines.

Description

out = daqfind returns all device objects that exist in the data acquisition engine. The output out is an array.

e1 = daqfind('PropertyName',PropertyValue,...) returns all device objects, channels, or lines that exist in the data acquisition engine and have the specified property names and property values. The property name/property value pairs can be specified as a cell array.

e2 = daqfind(S) returns all device objects, channels, or lines that exist in the data acquisition and have the property names and property values specified by S. S is a structure with field names that are property names and field values that are property values.

e3 = daqfind(obj,'PropertyName',PropertyValue,...) returns all device objects, channels, or lines listed by obj that have the specified property names and property values.
Remarks  More About Finding Device Objects, Channels, or Lines

daqfind is particularly useful in these circumstances:

- A device object is cleared from the MATLAB workspace, and it needs to be retrieved from the data acquisition engine.
- You need to locate device objects, channels, or lines that have particular property names and property values.

Rules for Specifying Property Names and Property Values

- You can use property name/property value string pairs, structures, and cell array pairs in the same call to daqfind. However, in a single call to daqfind, you can specify only device object properties or channel/line properties.
- You must use the same format as returned by get. For example, if get returns the ChannelName property value as Left, you must specify Left as the property value in daqfind (case matters). However, case does not matter when you specify enumerated property values. For example, daqfind will find a device object with a Running property value of On or on.

Examples

You can use daqfind to return a cleared device object.

    ai = analoginput('winsound');
    ch = addchannel(ai,1:2);
    set(ch,{'ChannelName'},{'Joe';'Jack'})
    clear ai
    ainew = daqfind;

To return the channel associated with the descriptive name Jack:

    ch2 = daqfind(ainew,'ChannelName','Jack');
To return the device object with a sampling rate of 8000 Hz and the descriptive name `winsound0-AI`, you can pass a structure to `daqfind`.

```matlab
S.Name = 'winsound0-AI';
S.SampleRate = 8000;
daqobj = daqfind(S);
```

**See Also**

**Functions**

`clear`, `get`, `propinfo`
Purpose
Help for device objects, constructors, adaptors, functions, and properties

Syntax
daqhelp
out = daqhelp('name')
out = daqhelp(obj)
out = daqhelp(obj,'name')

Arguments
’name’       A device object, constructor, adaptor, function, or property name.
obj          A device object.
out           Contains the specified help text.

Description
daqhelp displays a complete listing of Data Acquisition Toolbox constructors and functions along with a brief description of each.
out = daqhelp('name') returns help for the device object, constructor, adaptor, function, or property specified by name. The help text is returned to out.
out = daqhelp(obj) returns a complete listing of functions and properties for the device object obj to out. Help for obj’s constructor is also displayed.
out = daqhelp(obj,’name’) returns help for name for the specified device object obj to out. name can be a constructor, adaptor, property, or function name.
Remarks

As shown below, you can also display help via the Workspace browser by right-clicking a device object, and selecting **Explore > DAQ Help** from the context menu.

![Workspace browser with context menu](image)

**Access context (pop up) menus by right-clicking a device object.**

**More About Displaying Help**

- When displaying property help, 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 displaying function help, the “See Also” section contains only function names.

**Rules for Specifying Names**

For the `daqhelp('name')` syntax:

- If `name` is the name of a constructor, a complete listing of the device object’s functions and properties is displayed along with a brief description of each function and property. The constructor help is also displayed.

- You can display object-specific function information by specifying `name` as object/function. For example, to display the help for an analog input object’s `getdata` function, `name` is `analoginput/getdata`. 
You can display object-specific property information by specifying name as obj.property. For example, to display the help for an analog input object’s SampleRate property, name is analoginput.SampleRate.

For the daqhelp(obj,'name') syntax:

- If name is the name of a device object constructor and the .m extension is included, the constructor help is displayed.
- If name is the name of a function or property, the function or property help is displayed.

**Examples**

The following commands are some of the ways you can use daqhelp to obtain help on device objects, constructors, adaptors, functions, and properties.

```matlab
daqhelp('analogoutput');
out = daqhelp('analogoutput.m');
daqhelp set
daqhelp analoginput/peekdata
daqhelp analoginput.TriggerDelayUnits
```

The following commands are some of the ways you can use daqhelp to obtain information about functions and properties for an existing device object.

```matlab
ai = analoginput('winsound');
daqhelp(ai,'InitialTriggerTime')
out = daqhelp(ai,'getsample');
```

**See Also**

**Functions**

propinfo
### Purpose
Data acquisition hardware information

### Syntax
```matlab
out = daqhwinfo
out = daqhwinfo('adaptor')
out = daqhwinfo(obj)
out = daqhwinfo(obj,'FieldName')
```

### Arguments
- **'adaptor'** The hardware driver adaptor name. The supported adaptors are advantech, hpe1432, keithley, mcc, nidaq, parallel, and winsound.
- **obj** A device object or array of device objects.
- **'FieldName'** A single field name or a cell array of field names.
- **out** A structure containing the requested hardware information.

### Description
- **out = daqhwinfo** returns general hardware-related information as a structure to `out`. The returned information includes installed adaptors, the toolbox and MATLAB version, and the toolbox name.

- **out = daqhwinfo('adaptor')** returns hardware-related information for the specified `adaptor`. The returned information includes the adaptor DLL name, the board names and IDs, and the device object constructor syntax.

- **out = daqhwinfo('adaptor','FieldName')** returns the hardware-related information specified by `FieldName` for `adaptor`. `FieldName` must be a single string. `out` is a cell array. You can return a list of valid field names with the daqhwinfo('adaptor') syntax.

- **out = daqhwinfo(obj)** returns hardware-related information for the device object `obj`. If `obj` is an array of device objects, then `out` is a 1-by-n cell array of structures where n is the length of `obj`. The returned information depends on the device object type, and might include the maximum and minimum sampling rates, the channel gains, the hardware channel or line IDs, and the vendor driver version.
out = daqhwinfo(obj,'FieldName') returns the hardware-related information specified by FieldName for the device object obj. FieldName 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 FieldName. You can return a list of valid field names with the daqhwinfo(obj) syntax.

Remarks

As shown below, you can also return hardware information via the Workspace browser by right-clicking a device object, and selecting Explore > Display Hardware Info from the context menu.

Example

Display all installed adaptors. Note that this list might be different for your platform.

    out = daqhwinfo;
    out.InstalledAdaptors

    ans =
        'advantech'
        'hpe1432'
        'keithley'
        'mcc'
To display the device object constructor names for all installed winsound devices:

```matlab
out = daqhwinfo('winsound');
out.ObjectConstructorName
ans =
'naloginput('winsound',0)'  
'analoggoutput('winsound',0)'
```

Create the analog input object `ai` for a sound card. To display the input ranges for `ai`:

```matlab
ai = analoginput('winsound');
out = daqhwinfo(ai);
out.InputRanges
ans =
-1  1
```

To display the minimum and maximum sampling rates for `ai`:

```matlab
out = daqhwinfo(ai,{'MinSampleRate','MaxSampleRate'})
out =
[8000] [44100]
```
Purpose  Allocate or display memory resources

Syntax

\[
\begin{align*}
\text{out} &= \text{daqmem} \\
\text{out} &= \text{daqmem}(\text{obj}) \\
\text{daqmem}(\text{obj}, \text{maxmem})
\end{align*}
\]

Arguments

- \text{obj}  
  A device object or array of device objects.
- \text{maxmem}  
  The amount of memory to allocate.
- \text{out}  
  A structure containing information about memory resources.

Description  
\text{out} = \text{daqmem} \text{ returns the structure out, which contains several fields describing the memory resources associated with your platform and Data Acquisition Toolbox. The fields are described below.}

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{MemoryLoad}</td>
<td>Specifies a number between 0 and 100 that gives a general idea of current memory utilization. 0 indicates no memory use and 100 indicates full memory use.</td>
</tr>
<tr>
<td>\text{TotalPhys}</td>
<td>Indicates the total number of bytes of physical memory.</td>
</tr>
<tr>
<td>\text{AvailPhys}</td>
<td>Indicates the number of bytes of physical memory available.</td>
</tr>
<tr>
<td>\text{TotalPageFile}</td>
<td>Indicates the total number of bytes that can be stored in the paging file. Note that this number does not represent the actual physical size of the paging file on disk.</td>
</tr>
<tr>
<td>\text{AvailPageFile}</td>
<td>Indicates the number of bytes available in the paging file.</td>
</tr>
</tbody>
</table>
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TotalVirtual</td>
<td>Indicates the total number of bytes that can be described in the user mode portion of the virtual address space of the calling process.</td>
</tr>
<tr>
<td>AvailVirtual</td>
<td>Indicates the number of bytes of unreserved and uncommitted memory in the user mode portion of the virtual address space of the calling process.</td>
</tr>
<tr>
<td>UsedDaq</td>
<td>The total memory used by all device objects.</td>
</tr>
</tbody>
</table>

Note that all the above fields, except for UsedDaq, are identical to the fields returned by Windows’ MemoryStatus function.

```plaintext
out = daqmem(obj) returns a 1-by-N structure out containing two fields: UsedBytes and MaxBytes for the device object obj. N is the number of device objects specified by obj. UsedBytes returns the number of bytes used by obj. MaxBytes returns the maximum number of bytes that can be used by obj.

daqmem(obj,maxmem) sets the maximum memory that can be allocated for obj to the value specified by maxmem.
```

### Remarks

**More About Allocating and Displaying Memory Resources**

- For analog output objects, daqmem(obj,maxmem) controls the value of the MaxSamplesQueued property.

- If you manually configure the BufferingConfig property, then this value supersedes the values specified by daqmem(obj,maxmem) and the MaxSamplesQueued property.

### Examples

Create the analog input object aiwin for a sound card and the analog input object aini for a National Instruments board, and add two channels to each device object.

```plaintext
aiwin = analгинput('winsound');
addchannel(aiwin,1:2);
aini = anal/ginput('nidaq','Dev1');
```
addchannel(aini,0:1);

To display the total memory used by all existing device objects:

```
out = daqmem;
out.UsedDaq
ans =
    69120
```

To configure the maximum memory used by aiwin to 640 KB:

```
daqmem(aiwin,640000)
```

To configure the maximum memory used by each device object with one call to daqmem:

```
daqmem([aiwin aini],[640000 480000])
```

**See Also**

**Properties**

BufferingConfig, MaxSamplesQueued
<table>
<thead>
<tr>
<th><strong>Purpose</strong></th>
<th>Read Data Acquisition Toolbox (.daq) file</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>See Also</strong></td>
<td>This function is documented in the MATLAB documentation. See daqread.</td>
</tr>
</tbody>
</table>
Purpose
Register or unregister hardware driver adaptor

Syntax
\[
\text{daqregister('} \text{adaptor'}\text{')}
\]
\[
\text{daqregister('} \text{adaptor'}, '\text{unload'}\text{')}
\]
\[
\text{out} = \text{daqregister(...)}
\]

Arguments
- `'adaptor'` The hardware driver adaptor name. The supported adaptors are advantech, hpe1432, keithley, mcc, nidaq, parallel, and winsound.
- `'unload'` Specifies that the hardware driver adaptor is to be unloaded.
- `out` Captures the message returned by daqregister.

Description
\[
\text{daqregister('} \text{adaptor'}\text{') registers the hardware driver adaptor specified by} \text{adaptor. For third-party adaptors,} \text{adaptor must include the full pathname.}
\]
\[
\text{daqregister('} \text{adaptor'}, '\text{unload'}\text{') unregisters the hardware driver adaptor specified by} \text{adaptor. For third-party adaptors,} \text{adaptor must include the full pathname.}
\]
\[
\text{out} = \text{daqregister(...) captures the resulting message in} \text{out.}
\]

Remarks
A hardware driver adaptor must be registered so the data acquisition engine can make use of its services. Unless an adaptor is unloaded, registration is required only once.

For adaptors that are included with the toolbox, registration occurs automatically when you first create a device object. However, you might need to register third-party adaptors manually. In either case, you must install the associated hardware driver before registration can occur.
Examples

The following command registers the sound card adaptor provided with the toolbox.

```matlab
daqregister('winsound');
```

The following command registers the third-party adaptor `myadaptor.dll`. Note that you must supply the full pathname to `daqregister`.

```matlab
daqregister('D:/MATLABR12/toolbox/daq/myadaptors/myadaptor.dll');
```
Purpose
Remove device objects, engine MEX-file, and adaptor DLLs from memory

Syntax
daqreset

Description
daqreset removes all device objects existing in the engine, and unloads all data acquisition executables loaded by the engine (including the adaptor DLLs and the engine MEX-file).

You should use daqreset to return MATLAB to a known initial state of having no device objects and no data acquisition MEX-file or DLLs loaded in memory. When MATLAB returns to this state, the data acquisition hardware is reset.

See Also
Functions
clear, delete
Purpose Convert decimal value to binary vector

Syntax

\[
\text{out} = \text{dec2binvec}(\text{dec})
\]

\[
\text{out} = \text{dec2binvec}(\text{dec},\text{bits})
\]

Arguments

- \text{dec} A decimal value. \text{dec} must be nonnegative.
- \text{bits} Number of bits used to represent the decimal number.
- \text{out} A logical array containing the binary vector.

Description

\[
\text{out} = \text{dec2binvec}(\text{dec}) \text{ converts the decimal value } \text{dec} \text{ to an equivalent binary vector and stores the result as a logical array in } \text{out}. 
\]

\[
\text{out} = \text{dec2binvec}(\text{dec},\text{bits}) \text{ converts the decimal value } \text{dec} \text{ to an equivalent binary vector consisting of at least the number of bits specified by } \text{bits}. 
\]

Remarks

More About Binary Vectors

A binary vector (binvec) is constructed with the least significant bit (LSB) in the first column and the most significant bit (MSB) in the last column. For example, the decimal number 23 is written as the binvec value \([1 1 1 0 1]\).

More About Specifying the Number of Bits

- If \text{bits} is greater than the minimum number of bits required to represent the decimal value, then the result is padded with zeros.
- If \text{bits} is less than the minimum number of bits required to represent the decimal value, then the minimum number of required bits is used.
- If \text{bits} is not specified, then the minimum number of bits required to represent the number is used.
**Examples**

To convert the decimal value 23 to a binvec value:

```
dec2binvec(23)
an =
1 1 1 0 1
```

To convert the decimal value 23 to a binvec value using six bits:

```
dec2binvec(23,6)
an =
1 1 1 0 1 0
```

To convert the decimal value 23 to a binvec value using four bits, then the result uses five bits. This is the minimum number of bits required to represent the number.

```
dec2binvec(23,4)
an =
1 1 1 0 1
```

**See Also**

**Functions**

`binvec2dec`
**Purpose**  
Remove device objects, channels, or lines from data acquisition engine

**Syntax**  
delete(obj)  
delete(obj.Channel(index))  
delete(obj.Line(index))

**Arguments**  
obj  
A device object or array of device objects.  
obj.Channel(index)  
One or more channels contained by obj.  
obj.Line(index)  
One or more lines contained by obj.

**Description**  
delete(obj) removes the device object specified by obj from the engine. If obj contains channels or lines, they are removed as well. If obj is the last object accessing the driver, then the driver and associated adaptor are unloaded.

delete(obj.Channel(index)) removes the channels specified by index and contained by obj from the engine. As a result, the remaining channels might be reindexed.

delete(obj.Line(index)) removes the lines specified by index and contained by obj from the engine. As a result, the remaining lines might be reindexed.

**Remarks**  
Deleting device objects, channels, and lines follows these rules:

- delete removes device objects, channels, or lines from the data acquisition engine but not from the MATLAB workspace. To remove variables from the workspace, use the clear function.

- If multiple references to a device object exist in the workspace, then removing one device object from the engine invalidates the remaining references. These remaining references should be cleared from the workspace with the clear function.
If you delete a device object while it is running, then a warning is issued before it is deleted. You cannot delete a device object while it is logging or sending data.

You should use delete at the end of a data acquisition session. You can quickly delete all existing device objects with the command delete(daqfind).

If you use the help command to display the M-file help for delete, then you must supply the pathname shown below.

```
help daq/daqdevice/delete
```

### Examples

#### National Instruments

Create the analog input object ai for a National Instruments board, add hardware channels 0-7 to it, and make a copy of hardware channels 0 and 1.

```matlab
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:7);
ch = ai.Channel(1:2);
```

To delete hardware channels 0 and 1:

```matlab
delete(ch)
```

These channels are deleted from the data acquisition engine and are no longer associated with ai. The remaining channels are reindexed such that the indices begin at 1 and increase monotonically to 6. To delete ai:

```matlab
delete(ai)
```

#### Sound Card

Create the analog input object AI1 for a sound card, and configure it to operate in stereo mode.

```matlab
AI1 = analoginput('winsound');
addchannel(AI1,1:2);
```
You can now configure the sound card for mono mode by deleting hardware channel 2.

```matlab
delete(AI1.Channel(2))
```

If hardware channel 1 is deleted instead, an error is returned.

**See Also**

**Functions**

clear, daqreset
**Purpose**  
Create digital I/O object

**Syntax**  
```
DIO = digitalio('adaptor',ID)
```

**Arguments**  
- `'adaptor'`  The hardware driver adaptor name. The supported adaptors are advantech, keithley, mcc, nidaq, and parallel.
- `ID`  The hardware device identifier.
- `DIO`  The digital I/O object.

**Description**  
`DIO = digitalio('adaptor',ID)` creates the digital I/O object `DIO` for the specified `adaptor` and for the hardware device with device identifier `ID`. `ID` can be specified as an integer or a string.

**Remarks**  
**More About Creating Digital I/O Objects**

- When a digital I/O object is created, it does not contain any hardware lines. To execute the device object, hardware lines must be added with the `addline` function.
- You can create multiple digital I/O objects that are associated with a particular digital I/O subsystem. However, you can execute only one of these digital I/O objects at a time for the generation of timing events.
- The digital I/O object exists in the data acquisition engine and in the MATLAB workspace. If you create a copy of the device object, it references the original device object in the engine.
- The Name property is automatically assigned a descriptive name that is produced by concatenating `adaptor`, `ID`, and `-DIO`. You can change this name at any time.
The Parallel Port Adaptor

The toolbox provides basic DIO capabilities through the parallel port. The PC supports up to three parallel ports that are assigned the labels LPT1, LPT2, and LPT3. You can use only these ports. If you add additional ports to your system, or if the standard ports do not use the default memory resources, they will not be accessible by the toolbox. For more information about the parallel port, refer to “Parallel Port Characteristics” on page 7-9.

More About the Hardware Device Identifier

When data acquisition devices are installed, they are assigned a unique number, which identifies the device in software. The device identifier is typically assigned automatically and can usually be manually changed using a vendor-supplied device configuration utility. National Instruments refers to this number as the device number.

There are two ways you can determine the ID for a particular device:

- Type `daqhwinfo('adaptor')`.
- Open the vendor-supplied device configuration utility.

Examples

Create a digital I/O object for a National Instruments device defined as device number 1.

```matlab
DIO = digitalio('nidaq','Dev1');
```

Create a digital I/O object for parallel port LPT1.

```matlab
DIO = digitalio('parallel','LPT1');
```

See Also

Functions

`addline, daqhwinfo`

Properties

`Name`
**Purpose**
Summary information for device objects, channels, or lines

**Syntax**
disp(obj)
disp(obj.Channel(index))
disp(obj.Line(index))

**Arguments**
- obj: A device object.
- obj.Channel(index): One or more channels contained by obj.
- obj.Line(index): One or more lines contained by obj.

**Description**
disp(obj) displays summary information for the specified device object obj, and any channels or lines contained by obj. Typing obj at the command line produces the same summary information.

disp(obj.Channel(index)) displays summary information for the specified channels contained by obj. Typing obj.Channel(index) at the command line produces the same summary information.

disp(obj.Line(index)) displays summary information for the specified lines contained by obj. Typing obj.Line(index) at the command line produces the same summary information.

**Remarks**
You can invoke `disp` by typing the device object at the MATLAB command line or by excluding the semicolon when

- Creating a device object
- Adding channel or lines
- Configuring property values using the dot notation
As shown below, you can also display summary information via the Workspace browser by right-clicking a device object, a channel object, or a line object and selecting **Explore > Display Summary** from the context menu.

**Examples**

All the commands shown below produce summary information for the device object AI or the channels contained by AI.

```matlab
AI = analoginput('winsound')
chans = addchannel(AI,1:2)
AI.SampleRate = 44100
AI.Channel(1).ChannelName = 'CH1'
chans
```
flushdata

Purpose
Remove data from data acquisition engine

Syntax
flushdata(obj)
flushdata(obj,'mode')

Arguments
obj An analog input object or array of analog input objects.
'mode' Specifies how much data is removed from the engine.

Description
flushdata(obj) removes all data from the data acquisition engine and resets the SamplesAvailable property to zero.
flushdata(obj,'mode') removes data from the data acquisition engine depending on the value of mode:

- If mode is all, then flushdata removes all data from the engine and the SamplesAvailable property is set to 0. This is the same as flushdata(obj).
- If mode is triggers, then flushdata removes the data acquired during one trigger. triggers is a valid choice only when the TriggerRepeat property is greater than 0 and the SamplesPerTrigger property is not inf. The data associated with the oldest trigger is removed first.

Examples
Create the analog input object ai for a National Instruments board and add hardware channels 0-7 to it.

    ai = analoginput('nidaq','Dev1');
    addchannel(ai,0:7);
A 2-second acquisition is configured and the device object is executed.

```matlab
set(ai,'SampleRate',2000)
duration = 2;
ActualRate = get(ai,'SampleRate');
set(ai,'SamplesPerTrigger',ActualRate*duration)
start(ai)
wait(ai,duration+1)
```

Four thousand samples will be acquired for each channel group member. To extract 1000 samples from the data acquisition engine for each channel:

```matlab
data = getdata(ai,1000);
```

You can use `flushdata` to remove the remaining 3000 samples from the data acquisition engine.

```matlab
flushdata(ai)
ai.SamplesAvailable
ans =
0
```

**See Also**

**Functions**

`getdata`

**Properties**

`SamplesAvailable, SamplesPerTrigger, TriggerRepeat`
Purpose
Device object properties

Syntax
out = get(obj)
out = get(obj.Channel(index))
out = get(obj.Line(index))
out = get(obj,'PropertyName')
out = get(obj.Channel(index),'PropertyName')
out = get(obj.Line(index),'PropertyName')
get(...) 

Arguments
obj A device object or array of device objects.
obj.Channel(index) One or more channels contained by obj.
obj.Line(index) One or more lines contained by obj.
'PropertyName' A property name or a cell array of property names.

Description
out = get(obj) returns the structure out, where each field name is
the name of a property of obj and each field contains the value of that
property.

out = get(obj.Channel(index)) returns the structure out, where each
field name is the name of a channel property of obj and each field
contains the value of that property.

out = get(obj.Line(index)) returns the structure out, where each
field name is the name of a line property of obj and each field contains
the value of that property.

out = get(obj,'PropertyName') returns the value of the property
specified by PropertyName to out. If PropertyName is replaced by a
1-by-n or n-by-1 cell array of strings containing property names, then
get returns a 1-by-n cell array of values to out. If obj is an array
of data acquisition objects, then out will be 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.
out = get(obj.Channel(index),'PropertyName') returns the value of
PropertyName to out for the specified channels contained by obj. If
multiple channels and multiple property names are specified, then out
is an m-by-n cell array where m is the number of channels and n is
the number of properties.

out = get(obj.Line(index),'PropertyName') returns the value of
PropertyName to out for the specified lines contained by obj. If
multiple lines and multiple property names are specified, then out is an
m-by-n cell array where m is the number of lines and n is the number of
properties.

get(...) displays all property names and their current values for the
specified device object, channel, or line. Base properties are displayed
first followed by device-specific properties.

Remarks
If you use the help command to display the M-file help for get, then
you must supply the pathname shown below.

    help daq/daqdevice/get

Examples
Create the analog input object ai for a sound card and configure it to
operate in stereo mode.

    ai = analoginput('winsound');
    addchannel(ai,1:2);

The commands shown below are some of the ways you can use get to
return property values.

    chan = get(ai,'Channel');
    out = get(ai,{'SampleRate','TriggerDelayUnits'});
    out = get(ai);
    get(chan(1),'Units')
    get(chan,{'Index','HwChannel','ChannelName'})

See Also

Functions

set, setverify
getdata

Purpose
Data, time, and event information from data acquisition engine

Syntax

data = getdata(obj)
data = getdata(obj,samples)
data = getdata(obj,samples,'type')
[data,time] = getdata(...)
[data,time,abstime] = getdata(...)
[data,time,abstime,events] = getdata(...)
[data,...] = getdata(obj, 'P1', V1, 'P2', V2,...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>An analog input object.</td>
</tr>
<tr>
<td>samples</td>
<td>The number of samples to extract. If samples is not specified, the number of samples extracted is given by the SamplesPerTrigger property.</td>
</tr>
<tr>
<td>'type'</td>
<td>Specifies the format of the extracted data as double (the default) or as native.</td>
</tr>
<tr>
<td>data</td>
<td>An m-by-n array, where m is the number of samples extracted and n is the number of channels contained by obj.</td>
</tr>
<tr>
<td>time</td>
<td>An m-by-1 array of relative time values in seconds, where m is the number of samples extracted. time = 0 is defined as the point at which data logging begins, i.e., when the Logging property of obj is set to On. Measurement of time, with respect to 0, continues until the acquisition is stopped, i.e., when the Logging property of obj is set to Off.</td>
</tr>
<tr>
<td>abstime</td>
<td>The absolute time of the first trigger returned as a clock vector. This value is identical to the value stored by the InitialTriggerTime property.</td>
</tr>
<tr>
<td>events</td>
<td>A structure containing a list of events that occurred up to the time of the getdata call.</td>
</tr>
</tbody>
</table>
Description

data = getdata(obj) extracts the number of samples specified by the `SamplesPerTrigger` property for each channel contained by obj. data is an m-by-n array, where m is the number of samples extracted and n is the number of channels.

data = getdata(obj,samples) extracts the number of samples specified by samples for each channel contained by obj.

data = getdata(obj,samples,'type') extracts the number of samples specified by samples in the format specified by type for each channel contained by obj.

[data,time] = getdata(...) returns data as sample-time pairs. time is an m-by-1 array of relative time values, where m is the number of samples returned in data. Each element of time indicates the relative time, in seconds, of the corresponding sample in data, measured with respect to the first sample logged by the engine.

[data,time,abstime] = getdata(...) extracts data as sample-time pairs and returns the absolute time of the trigger. The absolute time is returned as a clock vector and is identical to the value stored by the `InitialTriggerTime` property.

[data,time,abstime,events] = getdata(...) extracts data as sample-time pairs, returns the absolute time of the trigger, and returns a structure containing a list of events that occurred up to the getdata call. The possible events that can be returned are identical to those stored by the `EventLog` property.

[data,...] = getdata(obj, 'P1', V1, 'P2', V2,...) specifies the number of samples to be returned, the format of the data matrix, and whether to return a `tscollection` object.

The following table shows a summary of properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Samples</td>
<td>Specify the number of samples to return.</td>
</tr>
</tbody>
</table>
**Property** | **Description**  
--- | ---  
DataFormat | Specify the data format as double (default) or native.  
OutputFormat | Specify the output format as matrix (default) or tscollection.  

**Remarks**

**More About getdata**

- In most circumstances, `getdata` returns all requested data and does not miss any samples. In the unlikely event that the engine cannot keep pace with the hardware device, it is possible that data is missed. If data is missed, the DataMissedFcn property is called and the device object is stopped.

- `getdata` is a *blocking* function because it returns execution control to the MATLAB workspace only when the requested number of samples are extracted from the engine for each channel group member.

- You can issue `^C` (*Ctrl+C*) while `getdata` is blocking. This will not stop the acquisition but will return control to MATLAB.

- The amount of data that you can extract from the engine is given by the SamplesAvailable property.

- It is a good practice to use a `wait` command before your `getdata` command if the `getdata` is going to get all data returned by the analog input subsystem. For example, if your analog input object is `ai` and you have set `duration` to be the number of seconds for the acquisition, you could add the following line right before the `getdata`:

  ```matlab
  wait(ai,duration+1)
  ```

- Setting the OutputFormat property to tscollection causes `getdata` to return a tscollection object. In this case, only the data left-hand argument is used.

- For more information on using the Time Series functionality, see Time Series Objects and Methods in the MATLAB documentation.
More About Extracting Data From the Engine

- After the requested data is extracted from the engine, the `SamplesAvailable` property value is automatically reduced by the number of samples returned.

- If the requested number of samples is greater than the samples to be acquired, then an error is returned.

- If the requested data is not returned in the expected amount of time, an error is returned. The expected time to return data is given by the time it takes the engine to fill one data block plus the time specified by the `Timeout` property.

- If multiple triggers are included in a single `getdata` call, a `NaN` is inserted into the returned data and time arrays and the absolute time returned is given by the first trigger.

Examples

Create the analog input object `ai` for a National Instruments board and add hardware channels 0 to 3 to it.

```matlab
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:3);
```

Configure a 1-second acquisition with `SampleRate` set to 1000 samples per second and `SamplesPerTrigger` set to 1000 samples per trigger.

```matlab
set(ai,'SampleRate',1000)
set(ai,'SamplesPerTrigger',1000)
start(ai)
```

The following `getdata` command blocks execution control until all sample-time pairs, the absolute time of the trigger, and any events that occurred during the `getdata` call are returned.

```matlab
wait(ai,1)
[data,time,abstime,events] = getdata(ai);
```
data is returned as a 1000-by-4 array of doubles, time is returned as a 1000-by-1 vector of relative times, abstime is returned as a clock vector, and events is returned as a 3-by-1 structure array.

To extract the 1000 data samples from hardware channel 0 only, examine the first column of data.

```matlab
chan0_data = data(:,1);
```

The three events returned are the start event, the trigger event, and the stop event. To return specific event information about the stop event, you must access the Type and Data fields.

```matlab
EventType = events(3).Type;
EventData = events(3).Data;
```

### See Also

**Functions**

flushdata, getsample, peekdata, timeseries, tscollection

**Properties**

DataMissedFcn, EventLog, SamplesAvailable, SamplesPerTrigger, Timeout
**Purpose**
Immediately acquire one sample

**Syntax**
sample = getsample(obj)

**Arguments**
- **obj**  
  An analog input object.
- **sample**  
  A row vector containing one sample for each channel contained by obj.

**Description**
sample = getsample(obj) immediately returns a row vector containing one sample for each channel contained by obj.

**Remarks**
Using `getsample` is a good way to test your analog input configuration. Additionally:

- `getsample` does not store samples in, or extract samples from, the data acquisition engine.
- You can execute `getsample` at any time after channels have been added to obj.
- Except for sound cards, you can use `getsample` on an analog input object that is not running (Running is Off). For sound cards, the device object must be running.

**Note**
Refer to the Vendor Limitations section before you access National Instruments devices with the NI-DAQmx adaptor simultaneously from multiple applications.

**Examples**
Create the analog input object `ai` and add eight channels to it.

```matlab
ai = analoginput('nidaq','Dev1');
ch = addchannel(ai,0:7);
```

The following command returns one sample for each channel.
sample = getsample(ai);

See Also

Functions

getdata, peekdata
**Purpose**
Read values from lines

**Syntax**

```
out = getvalue(obj)
out = getvalue(obj.Line(index))
```

**Arguments**

- `obj` A digital I/O object.
- `obj.Line(index)` One or more lines contained by `obj`.
- `out` A binary vector.

**Description**

- `out = getvalue(obj)` returns the current value from all lines contained by `obj` as a binary vector to `out`.
- `out = getvalue(obj.Line(index))` returns the current value from the lines specified by `obj.Line(index)`.

**Remarks**

**More About Reading Values from Lines**

- By default, `out` is returned as a binary vector (binvec). A binvec value is constructed with the least significant bit (LSB) in the first column and the most significant bit (MSB) in the last column. For example, the decimal number 23 is written as the binvec value `[1 1 1 0 1]`.
- You can convert a binvec value to a decimal value with the `binvec2dec` function.
- If `obj` contains lines from a port-configurable device, the data acquisition engine will automatically read from all the lines even if they are not contained by the device object.

**Note** Refer to the Vendor Limitations section before you access National Instruments devices with the NI-DAQmx adaptor simultaneously from multiple applications.
getvalue

Examples

Create the digital I/O object dio and add eight input lines to it.

```matlab
dio = digitalio('nidaq','Dev1');
lines = addline(dio,0:7,'in');
```

To return the current values from all lines contained by dio as a binvec value:

```matlab
out = getvalue(dio);
```

See Also

Functions

binvec2dec
Purpose
Open Property Inspector

Syntax
inspect(obj)

Arguments
obj An object or an array of objects.

Description
inspect(obj) opens the Property Inspector and allows you to inspect and set properties for the object obj.

Remarks
You can also open the Property Inspector via the Workspace browser by double-clicking an object in the Workspace list.

The Property Inspector does not automatically update its display. To refresh the Property Inspector, open it again.

Examples
Create the analog input object ai for a sound card and add two channels.

   ai = analoginput('winsound');
   addchannel(ai,1:2);

Open the Property Inspector for the object ai.

   inspect(ai)

The Property Inspector is shown below.

You can expand the properties that are arrays of objects. In the following figure, the Channel property is expanded to enumerate the individual channel objects that make up this property.

You can also expand these individual channel objects to display their own properties, as shown for channel 1.
See Also
daqfind, daqhelp, get, propinfo, set
**Purpose**
Check for channels

**Syntax**
\[ \text{out} = \text{ischannel}(	ext{obj.Channel(index)}) \]

**Arguments**
- \text{obj.Channel(index)}: One or more channels contained by \text{obj}.
- \text{out}: A logical value.

**Description**
\( \text{out} = \text{ischannel}(	ext{obj.Channel(index)}) \) returns a logical 1 to \text{out} if \text{obj.Channel(index)} is a channel. Otherwise, a logical 0 is returned.

**Remarks**
\text{ischannel} does not determine if channels are valid (associated with hardware). To check for valid channels, use the \text{isvalid} function.

Typically, you use \text{ischannel} directly only when you are creating your own M-files.

**Examples**
Suppose you create the function \text{myfunc} for use with Data Acquisition Toolbox. If \text{myfunc} is passed one or more channels as an input argument, then the first thing you should do in the function is check if the argument is a channel.

```matlab
function myfunc(chan)
    % Determine if a channel was passed.
    if ~ischannel(chan)
        error('The argument passed is not a channel.');
    end
end
```

You can examine the Data Acquisition Toolbox M-files for examples that use \text{ischannel}.

**See Also**

**Functions**
- \text{isvalid}
Purpose
Check for lines

Syntax
out = isdioline(obj.Line(index))

Arguments
obj.Line(index)  One or more lines contained by obj.
out            A logical value.

Description
out = isdioline(obj.Line(index)) returns a logical 1 to out if
obj.Line(index) is a line. Otherwise, a logical 0 is returned.

Remarks
isdioline does not determine if lines are valid (associated with
hardware). To check for valid lines, use the isvalid function.

Typically, you use isdioline directly only when you are creating your
own M-files.

Examples
Suppose you create the function myfunc for use with Data Acquisition
Toolbox. If myfunc is passed one or more lines as an input argument,
then the first thing you should do in the function is check if the
argument is a line.

    function myfunc(line)
    % Determine if a line was passed.
    if ~isdioline(line)
        error('The argument passed is not a line.');
    end

You can examine the Data Acquisition Toolbox M-files for examples
that use isdioline.

See Also
Functions
isvalid
**Purpose**
Determine whether analog input object is logging data

**Syntax**
bool = islogging(obj)

**Description**
bool = islogging(obj) returns true if the analog input object obj is logging data, otherwise false. An analog input object is logging if the value of its Logging property is set to On.

If obj is an array of analog 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 analog input objects in obj is invalid, islogging returns an error.

**Examples**
Create an analog input object and add a channel.

```matlab
ai = analoginput('winsound');
addchannel(ai, 1)
```

To put the analog input object in a logging state, start acquiring data. The example acquires 10 seconds of data to increase the amount of time that the object remains in the logging state.

```matlab
set(ai,'SamplesPerTrigger', 10*get(ai,'SampleRate'))
start(ai)
```

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

```matlab
bool = islogging(ai)
bool =
   1
```

Create a second analog input object.

```matlab
ai2 = analoginput('winsound');
```

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

```python
start(ai)
bool = islogging([ai ai2])
bool =
    1  0
```

**See Also**

**Functions**

isrunning, issending, start, stop

**Properties**

Logging, LoggingMode
**Purpose**
Determine whether device object is running

**Syntax**

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

**Description**
`bool = isrunning(obj)` returns `true` if the device object `obj` is running, otherwise `false`. A device object is running if the value of its `Running` property is set to `On`.

If `obj` is an array of device 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 device objects in `obj` is invalid, `isrunning` returns an error.

**Examples**
Create an analog input object and add a channel.

```matlab
ai = analoginput('winsound');
addchannel(ai, 1)
```

To put the analog input object in a running state, configure a manual trigger and then start the object.

```matlab
set(ai, 'TriggerType', 'Manual')
start(ai)
```

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

```matlab
bool = isrunning(ai)
bool =
1
```

Create an analog output object.

```matlab
ao = analogoutput('winsound');
```
Use `isrunning` to determine which of the two objects is running.

```plaintext
bool = isrunning([ai ao])
bool =
   1    0
```

**See Also**  
**Functions**  
islogging, issending, start, stop

**Properties**  
Running
**Purpose**

Determine whether analog output object is sending data

**Syntax**

\[ \text{bool} = \text{issending}(\text{obj}) \]

**Description**

\[ \text{bool} = \text{issending}(\text{obj}) \] returns true if the analog output object \( \text{obj} \) is sending data to the hardware device, otherwise false. An analog output object is sending if the value of its Sending property is set to On.

If \( \text{obj} \) is an array of analog output objects, \( \text{bool} \) is a logical array where each element in \( \text{bool} \) represents the corresponding element in \( \text{obj} \). If an object in \( \text{obj} \) is sending, the \text{issending} function sets the corresponding element in \( \text{bool} \) to true, otherwise false. If any of the analog output objects in \( \text{obj} \) is invalid, \text{issending} returns an error.

**Examples**

Create an analog output object and add a channel.

\[
\begin{align*}
\text{ao} &= \text{analogoutput('winsound')}; \\
\text{addchannel(ao, 1)};
\end{align*}
\]

To put the analog output object in a sending state, start acquiring data. The example sends 10 seconds of data to increase the amount of time that the object remains in the sending state.

\[
\begin{align*}
\text{putdata(ao, ones(10*get(ao,'SampleRate'),1))} \\
\text{start(ao)}
\end{align*}
\]

When the call to the \text{start} function returns, and the object is still sending data, use \text{issending} to check the state of the object.

\[
\begin{align*}
\text{bool} &= \text{issending(ao)} \\
\text{bool} &= 1
\end{align*}
\]

Create a second analog output object.

\[
\begin{align*}
\text{ao2} &= \text{analogoutput('winsound')};
\end{align*}
\]
Start one of the analog output objects again, such as ao, and use `issending` to determine which of the two objects is sending.

```plaintext
putdata(ao, ones(10*get(ao,'SampleRate'),1));
start(ao);
bool = issending([ao ao2])
bool =
    1    0
```

### See Also

- Functions
  - `islogging`, `isrunning`, `start`, `stop`

### Properties

- Sending
Purpose
Determine whether device objects, channels, or lines are valid

Syntax
out = isvalid(obj)
out = isvalid(obj.Channel(index))
out = isvalid(obj.Line(index))

Arguments
obj A device object or array of device objects.
obj.Channel(index) One or more channels contained by obj.
obj.Line(index) One or more lines contained by obj.
out A logical array.

Description
out = isvalid(obj) returns a logical 1 to out if obj is a valid device object. Otherwise, a logical 0 is returned.

out = isvalid(obj.Channel(index)) returns a logical 1 to out if the channels specified by obj.Channel(index) are valid. Otherwise, a logical 0 is returned.

out = isvalid(obj.Line(index)) returns a logical 1 to out if the lines specified by obj.Line(index) are valid. Otherwise, a logical 0 is returned.

Remarks
Invalid device objects, channels, and lines are no longer associated with any hardware and should be cleared from the workspace with the clear function.

Typically, you use isvalid directly only when you are creating your own M-files.

Examples
Create the analog input object ai for a National Instruments board and add eight channels to it.

ai = analoginput('nidaq','Dev1');
ch = addchannel(ai,0:7);
To verify the device object is valid:

```matlab
isValid(ai)
ans =
  1
```

To verify the channels are valid:

```matlab
isValid(ch)'
an =
  1 1 1 1 1 1 1 1 1
```

If you delete a channel, then `isValid` returns a logical 0 in the appropriate location:

```matlab
delete(ai.Channel(3))
isValid(ch)'
an =
  1 1 0 1 1 1 1 1 1
```

Typically, you use `isValid` directly only when you are creating your own M-files. Suppose you create the function `myfunc` for use with Data Acquisition Toolbox. If `myfunc` is passed the previously defined device object `ai` as an input argument,

```matlab
myfunc(ai)
```

the first thing you should do in the function is check if `ai` is a valid device object.

```matlab
function myfunc(obj)
    % Determine if an invalid handle was passed.
    if ~isValid(obj)
        error('Invalid data acquisition object passed.');
    end
```

You can examine the Data Acquisition Toolbox M-files for examples that use `isValid`. 
See Also

Functions

clear, delete, ischannel, isdioline
**length**

**Purpose**
Length of device object, channel group, or line group

**Syntax**
- `out = length(obj)`
- `out = length(obj.Channel)`
- `out = length(obj.Line)`

**Arguments**
- `obj` A device object or array of device objects.
- `obj.Channel` The channels contained by `obj`.
- `obj.Line` The lines contained by `obj`.
- `out` A double.

**Description**
- `out = length(obj)` returns the length of the device object `obj` to `out`.
- `out = length(obj.Channel)` returns the length of the channel group contained by `obj`.
- `out = length(obj.Line)` returns the length of the line group contained by `obj`.

**Examples**
Create the analog input object `ai` for a National Instruments board and add eight channels to it.

```matlab
ai = analoginput('nidaq','Dev1');
aich = addchannel(ai,0:7);
```

Create the analog output object `ao` for a National Instruments board, add one channel to it, and create the device object array `aiao`.

```matlab
ao = analogoutput('nidaq','Dev1');
aoch = addchannel(ao,0);
aiao = [ai ao]
```

<table>
<thead>
<tr>
<th>Index</th>
<th>Subsystem</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Analog Input</td>
<td>nidaq1-AI</td>
</tr>
<tr>
<td>2</td>
<td>Analog Output</td>
<td>nidaq1-AO</td>
</tr>
</tbody>
</table>
To find the length of `aiao`:

```
length(aiao)
ans =
    2
```

To find the length of the analog input channel group:

```
length(aich)
ans =
    8
```

See Also

**Functions**

`size`
**Purpose**
Load device objects, channels, or lines into MATLAB workspace

**Syntax**
```
load file
load file obj1 obj2...
out = load('file','obj1','obj2',...)
```

**Arguments**
- `file` The MAT-file name.
- `obj1` ... Device objects, an array of device objects, channels, or lines.
- `out` A structure containing the loaded device objects.

**Description**
- `load file` returns all variables from the MAT-file into the MATLAB workspace.
- `load file obj1 obj2...` returns the specified device objects from the MAT-file into the MATLAB workspace.
- `out = load('file','obj1','obj2',...)` returns the specified device objects from the MAT-file as a structure to `out` instead of directly loading them into the workspace. The field names in `out` match the names of the loaded device objects. If no device objects are specified, then all variables existing in the MAT-file are loaded.

**Remarks**
Loading device objects follows these rules:

- Unique device objects are loaded into the MATLAB workspace as well as the engine.
- If a loaded device object already exists in the engine but not the MATLAB workspace, the loaded device object automatically reconnects to the engine device object.
- If a loaded device object already exists in the workspace or the engine but has different properties than the loaded object, then these rules are followed:
- The read-only properties are automatically reset to their default values.

- All other property values are given by the loaded object and a warning is issued stating that property values of the workspace object have been updated.

- If the workspace device object is running, then it is stopped before loading occurs.

- If identical device objects are loaded, then they point to the same device object in the engine. For example, if you saved the array

\[
x = [\text{ai1} \ \text{ai1} \ \text{ai2}]
\]

only \text{ai1} and \text{ai2} are created in the engine, and \(x(1)\) will equal \(x(2)\).

- Values for read-only properties are restored to their default values upon loading. For example, the EventLog property is restored to an empty vector. Use the \text{propinfo} function to determine if a property is read only.

- Values for the BufferingConfig property when the BufferingMode property is set to Auto, and the MaxSamplesQueued property might not be restored to the same value because both these property values are based on available memory.

---

**Note** load is not used to read in acquired data that has been saved to a log file. You should use the daqread function for this purpose.

If you use the \text{help} command to display the M-file help for load, then you must supply the pathname shown below.

```
help daq/private/load
```

**Examples**

This example illustrates the behavior of load when the loaded device object has properties that differ from the workspace object.
load

ai = analoginput('winsound');
addchannel(ai,1:2);
save ai
ai.SampleRate = 10000;
load ai
Warning: Loaded object has updated property values.

See Also

Functions

daqread, propinfo, save
**Purpose**
List descriptive channel or line names

**Syntax**
\[ \text{names} = \text{makenames}('\text{prefix}', \text{index}) \]

**Arguments**
- **'prefix'**
  A string that constitutes the first part of the name.
- **index**
  Numbers appended to the end of prefix — any MATLAB vector syntax can be used to specify index as long as the numbers are positive.
- **names**
  An m-by-1 cell array of channel names where m is the length of index.

**Description**
\( \text{names} = \text{makenames}('\text{prefix}', \text{index}) \) generates a cell array of descriptive channel or line names by concatenating prefix and index.

**Remarks**
You can pass names as an input argument to the addchannel or addline function.

If names contains more than one descriptive name, then the size of names must agree with the number of hardware channels specified in addchannel, or the number of hardware lines specified in addline.

If the channels or lines are to be referenced by name, then prefix must begin with a letter and contain only letters, numbers, and underscores. Otherwise the names can contain any character.

**Examples**
Create the analog input object AI. You can use makenames to define descriptive names for each channel that is to be added to AI.

\[
\text{AI} = \text{analoginput('nidaq','Dev1')};
\text{names} = \text{makenames('chan',1:8)};
\]
names is an eight-element cell array of channel names chan1, chan2,..., chan8. You can now pass names as an input argument to the addchannel function.

    addchannel(AI,0:7,names);

See Also  Functions

addchannel, addline
Purpose

Multiplexed scanned channel index

Syntax

scanidx = muxchanidx(obj,muxboard,muxidx)
scanidx = muxchanidx(obj,absmuxidx)

Arguments

obj An analog input object associated with a National Instruments board.
muxboard The multiplexer board.
muxidx The index number of the multiplexed channel.
absmuxidx The absolute index number of the multiplexed channel.
scanidx The scanning index number of the multiplexed channel.

Description

scanidx = muxchanidx(obj,muxboard,muxidx) returns the scanning index number of the multiplexed channel specified by muxidx. The multiplexer (mux) board is specified by muxboard. For each mux board, muxidx can range from 0-31 for differential inputs and 0-63 for single-ended inputs. muxboard and muxidx are vectors of equal length.

scanidx = muxchanidx(obj,absmuxidx) returns the scanning index number of the multiplexed channel specified by absmuxidx. absmuxidx is the absolute index of the channel independent of the mux board.

For single-ended inputs, the first mux board has absolute index values that range between 0 and 63, the second mux board has absolute index values that range between 64 and 127, the third mux board has absolute index values that range between 128 and 191, the fourth mux board has absolute index values that range between 192 and 255. For example, the absolute index value of the second single-ended channel on the fourth mux board (muxboard is 4 and muxidx is 1) is 193.

Remarks

scanidx identifies the column number of the data returned by getdata and peekdata.
Refer to the *AMUX-64T User Manual* for more information about adding mux channels based on hardware channel IDs and the number of mux boards used.

**Examples**

Create the analog input object `ai` for a National Instruments board that is connected to four AMUX-64T multiplexers, and add 256 channels to `ai` using `addmuxchannel`.

```matlab
ai = analoginput('nidaq','Dev1');
ai.InputType = 'SingleEnded';
ai.NumMuxBoards = 4;
addmuxchannel(ai);
```

The following two commands return a scanned index value of 14.

```matlab
scanidx = muxchanidx(ai,4,1);
scanidx = muxchanidx(ai,193);
```

**See Also**

**Functions**

`addmuxchannel`
**Purpose**

Convert device objects, channels, or lines to MATLAB code

**Syntax**

```
obj2mfile(obj,'file')
obj2mfile(obj,'file','syntax')
obj2mfile(obj,'file','all')
obj2mfile(obj,'file','syntax','all')
```

**Arguments**

- **obj**
  A device object, array of device objects, channels, or lines.

- **'file'**
  The file that the MATLAB code is written to. The full pathname can be specified. If an extension is not specified, the .m extension is used.

- **'syntax'**
  Syntax of the converted MATLAB code. By default, the set syntax is used. If dot is specified, then the subscripted referencing syntax is used. If named is specified, then named referencing is used (if defined).

- **'all'**
  If all is specified, all properties are written to file. If all is not specified, only properties that are not set to their default values are written to file.

**Description**

```
obj2mfile(obj,'file') converts obj to the equivalent MATLAB code using the set syntax and saves the code to file. By default, only those properties that are not set to their default values are written to file.

obj2mfile(obj,'file','syntax') converts obj to the equivalent MATLAB code using syntax and saves the code to file. The values for syntax can be set, dot, or named. set uses the set syntax, dot uses subscripted assignment (dot notation), and named uses named referencing (if defined).

obj2mfile(obj,'file','all') converts obj to the equivalent MATLAB code using the set syntax and saves the code to file. all specifies that all properties are written to file.
```
obj2mfile(obj, 'file', 'syntax', 'all') converts obj including all of obj's properties to the equivalent MATLAB code using syntax and saves the code to file.

**Remarks**

If the UserData property is not empty or if any of the callback properties are set to a cell array of values or a function handle, then the data stored in those properties is written to a MAT-file when the object is converted and saved. The MAT-file has the same name as the M-file containing the object code (see the example below).

You can recreate the saved device objects by typing the name of the M-file at the command line. You can also recreate channels or lines, by typing the name of the M-file with a device object as the only input.

**Examples**

Create the analog input object ai for a sound card, add two channels, and set values for several properties.

```matlab
ai = analoginput('winsound');
addchannel(ai,1:2);
set(ai,'Tag','myai','TriggerRepeat',4)
set(ai,'StartFcn',{@mycallback,2,magic(10)})
```

The following command writes MATLAB code to the files myai.m and myai.mat.

```matlab
obj2mfile(ai, 'myai.m', 'dot')
```

myai.m contains code that recreates the analog input code shown above using the dot notation for all properties that have their default values changed. Because StartFcn is set to a cell array of values, this property appears in myai.m as

```matlab
ai.StartFcn = startfcn1;
```

and is saved in myai.mat as

```matlab
startfcn1 = {@mycallback,2,magic(10)};
```
To recreate ai and assign the device object to a new variable ainew:

```matlab
ainew = myai;
```

The associated MAT-file, myai.mat, is automatically loaded.
Purpose
Preview most recent acquired data

Syntax
\[
data = \text{peekdata}(\text{obj}, \text{samples}) \\
data = \text{peekdata}(\text{obj}, \text{samples}, 'type')
\]

Arguments
- \text{obj} \quad \text{An analog input object.}
- \text{samples} \quad \text{The number of samples to preview for each channel contained by \text{obj}.}
- 'type' \quad \text{Specifies the format of the extracted data as double (the default) or as native.}
- \text{data} \quad \text{An m-by-n matrix where m is the number of samples and n is the number of channels.}

Description
- \text{data = peekdata(obj, samples) returns the latest number of samples specified by samples to data.}
- \text{data = peekdata(obj, samples, 'type')} returns the number of samples specified by samples in the format specified by \text{type} for each channel contained by \text{obj}. If \text{type} is specified as \text{native}, the data is returned in the native data format of the device. If \text{type} is specified as double (the default), the data is returned as doubles.

Remarks
- **More About Using \text{peekdata}**
  - Unlike \text{getdata}, \text{peekdata} is a \textit{nonblocking} function that immediately returns control to \text{MATLAB}. Because \text{peekdata} does not block execution control, data might be missed or repeated.
  - \text{peekdata} takes a “snapshot” of the most recent acquired data and does not remove samples from the data acquisition engine. Therefore, the \text{SamplesAvailable} property value is not affected when \text{peekdata} is called.
Rules for Using peekdata

- You can call peekdata before a trigger executes. Therefore, peekdata is useful for previewing data before it is logged to the engine or to a disk file.

- In most cases, you will call peekdata while the device object is running. However, you can call peekdata once after the device object stops running.

- If samples is greater than the number of samples currently acquired, all available samples are returned with a warning message stating that the requested number of samples were not available.

Examples

Create the analog input object ai for a National Instruments board, add eight input channels, and configure ai for a two-second acquisition.

```matlab
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:7);
set(ai,'SampleRate',2000)
set(ai,'SamplesPerTrigger',4000)
```

After issuing the start function, you can preview the data.

```matlab
start(ai)
data = peekdata(ai,100);
```

peekdata returns 100 samples to data for all eight channel group members. If 100 samples are not available, then whatever samples are available will be returned and a warning message is issued. The data is not removed from the data acquisition engine.

See Also

Functions

gedata, getsample

Properties

SamplesAvailable
**Purpose**

Property characteristics for device objects, channels, or lines

**Syntax**

```matlab
out = propinfo(obj)
out = propinfo(obj,'PropertyName')
```

**Arguments**

- **obj**
  - A device object, channels, or lines.
- **'PropertyName'**
  - A valid obj property name.
- **out**
  - A structure whose field names are the property names for obj (if PropertyName is not specified).

**Description**

`out = propinfo(obj)` returns the structure `out` whose field names are the property names for `obj`. Each property name in `out` contains the fields shown below.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>The property data type. Possible values are any, callback, double, and string.</td>
</tr>
<tr>
<td>Constraint</td>
<td>The type of constraint on the property value. Possible values are bounded, callback, enum, and none.</td>
</tr>
<tr>
<td>ConstraintValue</td>
<td>The property value constraint. The constraint can be a range of valid values or a list of valid string values.</td>
</tr>
<tr>
<td>DefaultValue</td>
<td>The property default value.</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>Indicates when the property is read-only. Possible values are always, never, and whileRunning.</td>
</tr>
<tr>
<td>DeviceSpecific</td>
<td>If the property is device-specific, a 1 is returned. If a 0 is returned, the property is supported for all device objects of a given type.</td>
</tr>
</tbody>
</table>
out = propinfo(obj,'PropertyName') returns the structure out for the property specified by PropertyName. If PropertyName is a cell array of strings, a cell array of structures is returned for each property.

Examples

Create the analog input object ai for a sound card and configure it to operate in stereo mode.

    ai = analoginput('winsound');
    addchannel(ai,1:2);

To capture all property information for all common ai properties:

    out = propinfo(ai);

To display the default value for the SampleRate property:

    out.SampleRate.DefaultValue
    ans =
          8000

To display all the property information for the InputRange property:

    propinfo(ai.Channel,'InputRange')
    ans =
          Type: 'double'
          Constraint: 'Bounded'
          ConstraintValue: [-1 1]
          DefaultValue: [-1 1]
          ReadOnly: 'whileRunning'
          DeviceSpecific: 0

See Also

Functions

daqhelp
**Purpose**
Queue data in engine for eventual output

**Syntax**
`putdata(obj, data)`

**Arguments**
- **obj**: An analog output object.
- **data**: The data to be queued in the engine.

**Description**
`putdata(obj, data)` queues the data specified by `data` in the engine for eventual output to the analog output subsystem. `data` must consist of a column of data for each channel contained by `obj`. That is, `data` must be an m-by-n matrix, where m rows correspond to the number of samples and n columns correspond to the number of channels in `obj`.

`data` can consist of doubles or native data types but cannot contain NaNs. `data` must contain a column of data for each channel contained in `obj`. If `data` contains any data points that are not within the UnitsRange of the channel it pertains to, the data points will be clipped to the bounds of the UnitsRange property.

`data` can be a `tscollection` object or `timeseries` object. If `data` is a `tscollection` object, there must be one `timeseries` per channel in `obj`. If `data` is a `timeseries` object, there must be only one channel in `obj`. If the `tscollection` or `timeseries` object contains gaps, or is sampled at a different rate than the SampleRate of `obj`, the data will be resampled at the rate of `obj` using a zero order hold.

For more information on using the Time Series functionality, see Time Series Objects and Methods in the MATLAB documentation.

**Remarks**

**More About Queuing Data**

- Data must be queued in the engine before `obj` is executed.
putdata is a blocking function because it returns execution control to the MATLAB workspace only when the requested number of samples are queued in the engine for each channel group member.

If the value of the RepeatOutput property is greater than 0, then all queued data is automatically requeued until the RepeatOutput value is reached. RepeatOutput must be configured before start is issued.

After obj executes, you can continue to queue data unless RepeatOutput is greater than 0.

You can queue data in the engine until the value specified by the MaxSamplesQueued property is reached, or the limitations of your hardware or computer are reached.

You should not modify the BitsPerSample, InputRange, SensorRange, and UnitsRange properties after calling putdata. If these properties are modified, all data is deleted from the data acquisition engine. If you add a channel after calling putdata, all data will be deleted from the buffer.

The timeseries object must contain a single column of data.

More About Outputting Data

Data is output as soon as a trigger occurs.

An error is returned if a NaN is included in the data stream.

You can specify data as the native data type of the hardware. Note that MATLAB supports math operations only for the double data type. Therefore, to use math functions on native data, you must convert it to doubles.

If the output data is not within the range specified by the OutputRange property, then the data is clipped.

The SamplesOutput property keeps a running count of the total number of samples that have been output per channel.

The SamplesAvailable property tells you how many samples are ready to be output from the engine per channel. After data is output,
SamplesAvailable is automatically reduced by the number of samples sent to the hardware.

**Examples**

Create the analog output object ao for a National Instruments board, add two output channels to it, and generate 10 seconds of data to be output.

```matlab
ao = analogoutput('nidaq','Dev1');
ch = addchannel(ao,0:1);
set(ao,'SampleRate',1000)
data = linspace(0,1,10000)';
```

Before you can output data, it must be queued in the engine using `putdata`.

```matlab
putdata(ao,[data data])
start(ao)
```

**See Also**

**Functions**

`putsample`, `timeseries`, `tscollection`

**Properties**

`MaxSamplesQueued`, `OutputRange`, `RepeatOutput`, `SamplesAvailable`, `SamplesOutput`, `Timeout`, `UnitsRange`
Purpose
Immediately output one sample

Syntax
putsample(obj, data)

Arguments
obj An analog output object.
data The data to be queued in the engine.

Description
putsample(obj, data) immediately outputs the row vector data, which consists of one sample for each channel contained by obj.

Remarks
Using putsample is a good way to test your analog output configuration. Additionally:

• putsample does not store samples in the data acquisition engine.
• putsample can be executed at any time after channels have been added to obj.
• putsample is not supported for sound cards.

Note Refer to the Vendor Limitations section before you access National Instruments devices with the NI-DAQmx adaptor simultaneously from multiple applications.

Examples
Create the analog output object ao for a National Instruments board and add two hardware channels to it.

    ao = analogoutput('nidaq', 'Dev1');
    ch = addchannel(ao, 0:1);

To call putsample for ao:

    putsample(ao, [1 1])
See Also

Functions

putdata
**Purpose**  Write values to lines

**Syntax**  

\[
\text{putvalue(obj, data)} \\
\text{putvalue(obj.Line(index), data)}
\]

**Arguments**  

- `obj`  A digital I/O object.  
- `obj.Line(index)`  One or more lines contained by `obj`.  
- `data`  A decimal value or binary vector.

**Description**  

`putvalue(obj, data)` writes data to the hardware lines contained by the digital I/O object `obj`.  

`putvalue(obj.Line(index), data)` writes data to the hardware lines specified by `obj.Line(index)`.

**Remarks**  

More About Writing Values to Lines

- You can specify data as either a decimal value or a binary vector. A binary vector (or `binvec`) is constructed with the least significant bit (LSB) in the first column and the most significant bit (MSB) in the last column. For example, the decimal number 23 is written as the binary vector `[1 1 1 0 1]`.

- If `obj` contains lines from a port-configurable device, then all lines will be written to even if they are not contained by the device object.

- An error will be returned if data is written to an input line.

- An error is returned if you attempt to write a negative value.

- If a decimal value is written to a digital I/O object and the value is too large to be represented by the hardware, then an error is returned.
Note  Refer to the Vendor Limitations section before you access National Instruments devices with the NI-DAQmx adaptor simultaneously from multiple applications.

Examples

Create the digital I/O object dio and add four output lines to it.

    dio = digitalio('nidaq','Dev1');
    lines = addline(dio,0:3,'out');

Write the value 8 as a decimal value and as a binary vector.

    putvalue(dio,8)
    putvalue(dio,[0 0 0 1])
**Purpose**

Save device objects to MAT-file

**Syntax**

```
save file
save file obj1 obj2...
```

**Arguments**

- **file**
  - The MAT-file name.
- **obj1 obj2...**
  - One or more device objects or an array of device objects.

**Description**

`save file` saves all MATLAB variables to the MAT-file `file`. If an extension is not specified for `file`, then a `.MAT` extension is used.

`save file obj1 obj2...` saves the specified device objects to `file`.

**Remarks**

Saving device objects follows these rules:

- You can use `save` in the functional form as well as the command form shown above. When using the functional form, you must specify the filename and device objects as strings.

- Samples associated with a device object are not stored in the MAT-file. You can bring these samples into the MATLAB workspace with the `getdata` function, and then save them to the MAT-file using a separate variable name. You can also log samples to disk by configuring the `LoggingMode` property to `Disk` or `Disk&Memory`.

- Values for read-only properties are restored to their default values upon loading. For example, the `EventLog` property is restored to an empty vector. Use the `propinfo` function to determine if a property is read only.

- Values for the `BufferingConfig` property (if the `BufferingMode` property is set to `Auto`) and the `MaxSamplesQueued` property might not be restored because both these property values are based on available memory.
If you use the `help` command to display the M-file help for `save`, then you must supply this pathname:

```
help daq/private/save
```

### See Also

### Functions

`getdata`, `load`, `propinfo`
set

**Purpose**
Configure or display device object properties

**Syntax**
```
set(obj)
props = set(obj)
set(obj,'PropertyName')
props = set(obj,'PropertyName')
set(obj,'PropertyName',PropertyValue,...)
set(obj,PN,PV)
set(obj,S)
```

**Arguments**
- **obj**
  A device object, array of device objects, channels, or lines.
- **'PropertyName'**
  A property name.
- **PropertyValue**
  A property value.
- **PN**
  A cell array of property names.
- **PV**
  A cell array of property values.
- **S**
  A structure whose field names are device object, channel, or line properties.
- **props**
  A structure array whose field names are the property names for `obj`, or a cell array of possible values.

**Description**
- `set(obj)` displays all configurable properties for `obj`. If a property has a finite list of possible string values, then these values are also displayed.
- `props = set(obj)` returns all configurable properties to `props`. `props` is a structure array with fields given by the property names, and possible property values contained in cell arrays. If the property does not have a finite set of possible values, then the cell array is empty.
- `set(obj,'PropertyName')` displays the valid values for the property specified by `PropertyName`. `PropertyName` must have a finite set of possible values.
set

props = set(obj,'PropertyName') returns the valid values for PropertyName to props. props is a cell array of possible values or an empty cell array if the property does not have a finite set of possible values.

set(obj,'PropertyName',PropertyValue,...) sets multiple property values with a single statement. Note that you can use structures, property name/property value string pairs, and property name/property value cell array pairs in the same call to set.

set(obj,PN,PV) sets the properties specified in the cell array of strings PN to the corresponding values in the cell array PV. PN must be a vector. PV can be m-by-n where m is equal to the specified number of device objects, channels, or lines and n is equal to the length of PN.

set(obj,S) where S is a structure whose field names are device object properties, sets the properties named in each field name with the values contained in the structure.

Remarks

If you use the help command to display the M-file help for set, then you must supply the pathname shown below.

help daq/daqdevice/set

Examples

Create the analog input object ai for a sound card and configure it to operate in stereo mode.

ai = analoginput('winsound');
addchannel(ai,1:2);

To display all of ai’s configurable properties and their valid values:

set(ai)

To set the value for the SampleRate property to 10000:

set(ai,'SampleRate',10000)
The following two commands set the value for the SampleRate and InputType properties using one call to set.

```matlab
set(ai,'SampleRate',10000,'TriggerType','Manual')
set(ai,{'SampleRate','TriggerType'},[10000,'Manual'])
```

You can also set different channel property values for multiple channels.

```matlab
ch = ai.Channel(1:2);
set(ch,{'UnitsRange','ChannelName'},{{[-1 1] 'Name1'; [-2 2] 'Name2'}})
```

**See Also**

**Functions**

get, setverify
Purpose
Configure and return specified property

Syntax
Actual = setverify(obj,'PropertyName',PropertyValue)
Actual = setverify(obj.Channel(index),'PropertyName',PropertyValue)
Actual = setverify(obj.Line(index),'PropertyName',PropertyValue)

Arguments

obj
A device object or array of device objects.

'PropertyName'
A property name.

PropertyValue
A property value.

obj.Channel(index)
One or more channels contained by obj.

obj.Line(index)
One or more lines contained by obj.

Actual
The actual value for the specified property.

Description
Actual = setverify(obj,'PropertyName',PropertyValue) sets
PropertyName to PropertyValue for obj, and returns the actual
property value to Actual.

Actual =
setverify(obj.Channel(index),'PropertyName',PropertyValue) sets
PropertyName to PropertyValue for the channels specified by index,
and returns the actual property value to Actual.

Actual =
setverify(obj.Line(index),'PropertyName',PropertyValue) sets
PropertyName to PropertyValue for the lines specified by index, and
returns the actual property value to Actual.

Remarks
setverify is equivalent to the commands

set(obj,'PropertyName',PropertyValue)
Actual = get(obj,'PropertyName')
Using setverify is not required for setting property values, but it does provide a convenient way to verify the actual property value set by the data acquisition engine.

setverify is particularly useful when setting the SampleRate, InputRange, and OutputRange properties because these properties can only be set to specific values accepted by the hardware. You can use the propinfo function to obtain information about the valid values for these properties.

If a property value is specified but does not match a valid value, then

- If the specified value is within the range of supported values,
  - For the SampleRate and InputRange properties, the value is automatically rounded up to the next highest supported value.
  - For all other properties, the value is automatically selected to be the nearest supported value.
- If the value is not within the range of supported values, an error is returned and the current property value remains unchanged.

**Examples**

Create the analog input object ai for a National Instruments AT-MIO-16DE-10 board, add eight hardware channels to it, and set the sample rate to 10,000 Hz using setverify.

```matlab
ai = analoginput('nidaq','Dev1');
ch = addchannel(ai,0:7);
ActualRate = setverify(ai,'SampleRate',10000);
```

Suppose you use setverify to set the input range for all channels contained by ai to -8 to 8 volts.

```matlab
ActualInputRange = setverify(ai.Channel,'InputRange',[-8 8]);
```

The InputRange value was actually rounded up to -10 to 10 volts.

```matlab
ActualInputRange{1}
an =
```
See Also

Functions
get, propinfo, set

Properties
InputRange, OutputRange, SampleRate
**Purpose**
Event log information

**Syntax**

showdaqevents(obj)
showdaqevents(obj,index)
showdaqevents(struct)
showdaqevents(struct,index)
out = showdaqevents(...)

**Arguments**

obj  An analog input or analog output object.
index  The event index.
struct  An event structure.
out  A one column cell array of event information.

**Description**

showdaqevents(obj) displays a summary of the event log for obj.
showdaqevents(obj,index) displays a summary of the events specified by index for obj.
showdaqevents(struct) displays a summary of the events stored in the structure struct.
showdaqevents(struct,index) displays a summary of the events specified by index stored in the structure struct.
out = showdaqevents(...) outputs the event information to a one column cell array out. Each element of out is a string that contains the event information associated with that index value.

**Remarks**

You can pass a structure of event information to showdaqevents. This structure can be obtained from the getdata function, the daqread function, or the EventLog property.
As shown below, you can also display event information via the Workspace browser by right-clicking a device object and selecting **Explore > Show DAQ Events** from the context menu.

![Workspace browser with Show DAQ Events context menu](image)

### Examples

Create the analog input object `ai` for a sound card, add two channels, and configure `ai` to execute three triggers.

```matlab
ai = analoginput('winsound');
ch = addchannel(ai,1:2);
set(ai,'TriggerRepeat',2)
```

Start `ai` and display the trigger event information with `showdaqevents`.

```matlab
start(ai)
showdaqevents(ai,2:4)
```

<table>
<thead>
<tr>
<th>Trigger#1</th>
<th>Time</th>
<th>Channel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(17:07:06, 0)</td>
<td>N/A</td>
</tr>
<tr>
<td>Trigger#2</td>
<td>(17:07:07, 8000)</td>
<td>N/A</td>
</tr>
<tr>
<td>Trigger#3</td>
<td>(17:07:08, 16000)</td>
<td>N/A</td>
</tr>
</tbody>
</table>
See Also

Functions

daqread, getdata

Properties

EventLog
**Purpose**
Size of device object, channel group, or line group

**Syntax**
```
d = size(obj)
[m1,m2,m3,...,mn] = size(obj)
m = size(obj,dim)
d = size(obj.Channel)
[m1,m2,m3,...,mn] = size(obj.Channel)
m = size(obj.Channel,dim)
d = size(obj.Line)
[m1,m2,m3,...,mn] = size(obj.Line)
m = size(obj.Line,dim)
```

**Arguments**
- `obj`: A device object or array of device objects.
- `dim`: The dimension.
- `obj.Channel`: The channels contained by `obj`.
- `obj.Line`: The lines contained by `obj`.
- `d`: A two-element row vector containing the number of rows and columns in `obj`.
- `m1,m2,m3,...,mn`: Each dimension of `obj` is captured in a separate variable.
- `m`: The length of the dimension specified by `dim`.

**Description**
- `d = size(obj)` returns the two-element row vector `d = [m,n]` containing the number of rows and columns in `obj`.
- `[m1,m2,m3,...,mn] = size(obj)` returns the length of the first `n` dimensions of `obj` to separate output variables. For example, `[m,n] = size(obj)` returns the number of rows to `m` and the number of columns to `n`.
- `m = size(obj,dim)` returns the length of the dimension specified by the scalar `dim`. For example, `size(obj,1)` returns the number of rows.
\[ d = \text{size}(\text{obj.Channel}) \] returns the two-element row vector \( d = [m, n] \) containing the number of rows and columns in the channel group \( \text{obj.Channel} \).

\[ [m_1, m_2, m_3, \ldots, m_n] = \text{size}(\text{obj.Channel}) \] returns the length of the first \( n \) dimensions of the channel group \( \text{obj.Channel} \) to separate output variables. For example, \( [m, n] = \text{size}(\text{obj.Channel}) \) returns the number of rows to \( m \) and the number of columns to \( n \).

\( m = \text{size}(\text{obj.Channel}, \text{dim}) \) returns the length of the dimension specified by the scalar \( \text{dim} \). For example, \( \text{size}(\text{obj.Channel}, 1) \) returns the number of rows.

\[ d = \text{size}(\text{obj.Line}) \] returns the two-element row vector \( d = [m, n] \) containing the number of rows and columns in the line group \( \text{obj.Line} \).

\[ [m_1, m_2, m_3, \ldots, m_n] = \text{size}(\text{obj.Line}) \] returns the length of the first \( n \) dimensions of the line group \( \text{obj.Line} \) to separate output variables. For example, \( [m, n] = \text{size}(\text{obj.Line}) \) returns the number of rows to \( m \) and the number of columns to \( n \).

\( m = \text{size}(\text{obj.Line}, \text{dim}) \) returns the length of the dimension specified by the scalar \( \text{dim} \). For example, \( \text{size}(\text{obj.Line}, 1) \) returns the number of rows.

### Examples

Create the analog input object \( \text{ai} \) for a National Instruments board and add eight channels to it.

\[
\text{ai} = \text{analoginput}('nidaq','Dev1');
\text{ch} = \text{addchannel}(	ext{ai},0:7);
\]

To find the size of the device object:

\[
\text{size}(	ext{ai})
\]

\[
\text{ans} =
\begin{bmatrix}
1 & 1
\end{bmatrix}
\]
To find the size of the channel group:

```matlab
size(ch)
ans =
   8   1
```

**See Also**

**Functions**

- `length`
Purpose

Open data acquisition oscilloscope

Syntax

softscope
softscope(obj)
softscope('fname.si')

Arguments

obj An analog input object.
fname.si Name of the file containing Oscilloscope settings.

Description

softscope opens the Hardware Configuration graphical user interface (GUI), which allows you to configure the hardware device to be used with the Oscilloscope. The Oscilloscope opens when you click the OK button, and at least one hardware channel is selected.

softscope(obj) opens the Oscilloscope configured to display the data acquired from the analog input object, obj. obj must contain at least one hardware channel.

softscope('fname.si') pens the Oscilloscope using the settings saved in the softscope file specified by fname. fname is generated from the Oscilloscope’s File > Save or File > Save As menu item.

Remarks

The Oscilloscope is a graphical user interface (GUI) that allows you to

• Stream acquired data into a display.
• Scale displayed data, and configure triggers and measurements.
• Configure analog input hardware settings.
• Export measurements and acquired data.

To support these tasks, the Oscilloscope includes several helper GUIs, which are described below.
Hardware Configuration

The Hardware Configuration GUI allows you to add channels from a particular hardware device to the Oscilloscope GUI. You can configure the device’s sample rate and input type, as well as the input range for each added channel. The GUI shown below is configured to add both sound card channels using the default sample rate.

![Hardware Configuration GUI](image)

Oscilloscope

The Oscilloscope GUI consists of these panes:

- **Display** pane — The display pane contains the hardware channel data (a trace) and the measurements, if defined. The display area also contains labels for each channel's horizontal and vertical units, and indicators for
  - Each trace
  - The trigger level (if defined)
  - The location of the start of the trigger (used for pretriggers)

- **Channel** pane — The channel pane lists the hardware channels, math channels, and reference channels that are currently being viewed in a display. The Channel Panel also contains knobs for configuring
- The display’s horizontal offset and horizontal scale
- The selected channel’s vertical offset and vertical scale

• **Trigger** pane — The trigger pane allows you to define how data acquisition is initiated. There are three trigger types:
  - One-shot — Acquire the specified number of samples once.
  - Continuous — Continuously acquire the specified number of samples.
  - Sequence — Continuously acquire the specified number of samples, and use the dependent trigger type each time.

For each trigger type, the Oscilloscope begins to acquire data after you press the **Trigger** button.

• **Measurement** pane — The measurement pane lists all measurements that are currently being taken. When defining a measurement, you must specify
  - The hardware, math, or reference channel
  - The measurement type
  - Whether the measurement result is drawn as a cursor in the display
The Oscilloscope GUI shown below is configured to display the sound card channels in separate displays.

Channel Exporter

The Channel Exporter allows you to export the data associated with a hardware channel, a math channel, or a reference channel. You can export the channel data to one of four destinations:

- The MATLAB workspace as an array
- The MATLAB workspace as a structure
- A MATLAB figure window
- A MAT-file
All channels added to the oscilloscope are listed in the GUI.

**Measurement Exporter**

The Measurement Exporter allows you to export the data associated with a measurement. You can export the measurement to one of three destinations:

- The MATLAB workspace
- A MATLAB figure window
- A MAT-file
The number of measurements exported depends on the BufferSize property value. By default, BufferSize is 1 indicating that the last measurement value calculated is available to export.

Scope Editor

The Scope Editor consists of two panes:

- **Scope** — Add and remove displays, the channel pane, the measurement pane, and the trigger pane. Note that you can define as many displays as you want, but there can only be only one channel pane, measurement pane, and trigger pane in the Oscilloscope at a time.
• **Scope Properties** — Configure properties for the displays, the channel pane, the measurement pane, and the trigger pane.

![Scope Editor](image)

**Channel Editor**

The Channel Editor consists of three panes:

- **Channel** — Add or delete math channels and reference channels, and select which defined channels are available to the Oscilloscope.
- **Channel Properties** — Configure properties for defined hardware channels, math channels, and reference channels.
• **Channel Display** — Select the Oscilloscope display for each defined channel, or choose to not display a channel.

![Channel Editor]

**Measurement Editor**

The Measurement Editor consists of three panes:

• **Measurement** — Add or delete measurements, and select which defined measurements are available to the Oscilloscope.

• **Measurement Properties** — Configure properties for the defined measurements.
• **Measurement Type** — Add or delete measurement types, and select which defined measurement types are available to the Oscilloscope.
**Purpose**
Start device object

**Syntax**
start(obj)

**Arguments**
obj    A device object or an array of device objects.

**Description**
start(obj) initiates the execution of the device object obj.

**Remarks**
When start is issued for an analog input or analog output object,

- The M-file callback function specified for StartFcn is executed.
- The Running property is set to On.
- The start event is recorded in the EventLog property.
- Data existing in the engine is flushed.

Although an analog input or analog output object might be executing,
data logging or sending is not necessarily initiated. Data logging or sending requires a trigger event to occur, and depends on the TriggerType property value.

For any device object, you can specify start as the value for a callback property.

ai.StopFcn = @start;

---

**Note**
You typically execute a digital I/O object to periodically update and display its state. Refer to the diopanel demo for an example of this behavior.

If you want to synchronize the input and output of data, or you require more control over when your hardware starts, you should use the ManualTriggerHwOn property.
See Also

Functions

stop, trigger

Properties

EventLog, ManualTriggerHwOn, Running, Sending, TriggerType
**Purpose**  
Stop device object

**Syntax**  
`stop(obj)`

**Arguments**  
`obj`  
A device object or an array of device objects.

**Description**  
`stop(obj)` terminates the execution of the device object `obj`.

**Remarks**  
An analog input object automatically stops when the requested samples are acquired or data is missed. An analog output object automatically stops when the queued data is output. These two device objects can also stop executing under one of these conditions:

- The `Timeout` property value is reached.
- A run-time error occurs.

For analog input objects, `stop` must be used when the `TriggerRepeat` property or `SamplesPerTrigger` property is set to `inf`. For analog output objects, `stop` must be used when the `RepeatOutput` property is set to `inf`. When `stop` is issued for either of these device objects,

- The `Running` property is set to `Off`.
- The `Logging` property or `Sending` property is set to `Off`.
- The M-file callback function specified for `StopFcn` is executed.
- The stop event is recorded in the `EventLog` property.
- All pending callbacks for this object are discarded.

For any device object, you can specify `stop` as the value for a callback property.

```matlab
ao.TimerFcn = @stop;
```
**Note** Issuing `stop` is the only way to stop an executing digital I/O object. You typically execute a digital I/O object to periodically update and display its state. Refer to the `diopanel` demo for an example.

**See Also**

**Functions**

`start`, `trigger`

**Properties**

`EventLog`, `Logging`, `RepeatOutput`, `Running`, `SamplesPerTrigger`, `Sending`, `Timeout`, `TriggerRepeat`
**Purpose**
Manually execute trigger

**Syntax**
trigger(obj)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>An analog input or analog output object or an array of these device objects.</td>
</tr>
</tbody>
</table>

**Description**
trigger(obj) manually executes a trigger.

**Remarks**
After trigger is issued,

- The absolute time of the trigger event is recorded by the InitialTriggerTime property.
- The Logging property or Sending property is set to On.
- The M-file callback function specified by TriggerFcn is executed.
- The trigger event is recorded in the EventLog property.

You can issue trigger only if TriggerType is set to Manual, Running is On, and Logging is Off.

You can specify trigger as the value for a callback property.

```plaintext
ai.StartFcn = @trigger;
```

**See Also**

**Functions**
start, stop

**Properties**

InitialTriggerTime, Logging, Running, Sending, TriggerFcn, TriggerType
Purpose
Wait until device object stops running

Syntax
wait(obj,waittime)

Arguments
obj A device object or an array of device objects.
waittime The maximum time to wait for obj to stop running.

Description
wait(obj,waittime) blocks the MATLAB command line, and waits for obj to stop running. You specify the maximum waiting time, in seconds, with waittime. waittime overrides the value specified for the Timeout property. If obj is an array of device objects, then wait might wait up to the specified time for each device object in the array.

wait is particularly useful if you want to guarantee that the specified data is acquired before another task is performed.

Remarks
If obj is not running when wait is issued, or if an error occurs while obj is running, then wait immediately relinquishes control of the command line.

When obj stops running, its Running property is automatically set to Off. obj can stop running under one of these conditions:

- The requested number of samples is acquired (analog input) or sent out (analog output).
- The stop function is issued on that object.
- A run-time error occurs.
- The Timeout property value is reached (waittime supersedes this value).

All callbacks, including the StopFcn, are executed before wait returns.
Examples

Create the analog input object \( ai \) for a National Instruments board, add eight channels to it, and configure a 25-second acquisition.

\[
ai = \text{analoginput}('nidaq','Dev1');
ch = \text{addchannel}(ai,0:7);
ai.SampleRate = 2000;
ai.TriggerRepeat = 4;
ai.SamplesPerTrigger = 10000;
\]

You can use \text{wait} to block the MATLAB command line until all the requested data is acquired. Because the expected acquisition time is 25 seconds, the \text{wait} time argument is 26. If the acquisition does not complete within this time, then a timeout occurs.

\[
\text{start}(ai) \\
\text{wait}(ai,26)
\]

See Also

Properties

EventLog, Running, StopFcn, Timeout
Base Properties — By Category

Base properties apply to all supported hardware subsystems of a given type (analog input, analog output, or digital I/O). For example, the SampleRate property is supported for all analog input subsystems regardless of the vendor.

The properties are categorized according to these subsystems:
Analog Input Properties (p. 13-3) Analog input base properties are divided into two main categories: common properties and channel properties. Common properties apply to every channel contained by the analog input object, while channel properties can be configured for individual channels.

Analog Output Properties (p. 13-7) Analog output base properties are divided into two main categories: common properties and channel properties. Common properties apply to every channel contained by the analog output object, while channel properties can be configured for individual channels.

Digital I/O Properties (p. 13-11) Digital I/O base properties are divided into two main categories: common properties and line properties. Common properties apply to every line contained by the digital I/O object, while line properties can be configured for individual lines.

Depending on the hardware device you are using, additional property names or property values might be present. The additional property names are described in Chapter 16, “Device-Specific Properties — Alphabetical List”. For example, only analog input and analog output objects associated with a sound card have a BitsPerSample property. The additional property values are also device-specific but are included in this chapter. For example, all supported devices have an InputType property, but the value AC-Coupled is unique to analog input objects associated with a sound card.
## Analog Input Properties

### Common Properties
The analog input common properties are grouped into the following categories based on usage.

### Analog Input Basic Setup Properties
- **SampleRate**: Specify per-channel rate at which analog data is converted to digital data, or vice versa.
- **SamplesPerTrigger**: Specify number of samples to acquire for each channel group member for each trigger that occurs.
- **TriggerType**: Specify type of trigger to execute.

### Analog Input Logging Properties
- **LogFileName**: Specify name of disk file information is logged to.
- **Logging**: Indicate whether data is being logged to memory or disk file.
- **LoggingMode**: Specify destination for acquired data.
- **LogToDiskMode**: Specify whether data, events, and hardware information are saved to one or more disk files.

### Analog Input Trigger Properties
- **InitialTriggerTime**: Absolute time of first trigger.
- **ManualTriggerHwOn**: Specify hardware device starts at manual trigger.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TriggerChannel</td>
<td>Specify channel serving as trigger source</td>
</tr>
<tr>
<td>TriggerCondition</td>
<td>Specify condition that must be satisfied before trigger executes</td>
</tr>
<tr>
<td>TriggerConditionValue</td>
<td>Specify voltage value(s) that must be satisfied before trigger executes</td>
</tr>
<tr>
<td>TriggerDelay</td>
<td>Specify delay value for data logging</td>
</tr>
<tr>
<td>TriggerDelayUnits</td>
<td>Specify units in which trigger delay data is measured</td>
</tr>
<tr>
<td>TriggerFcn</td>
<td>Specify M-file callback function to execute when trigger occurs</td>
</tr>
<tr>
<td>TriggerRepeat</td>
<td>Specify number of additional times trigger executes</td>
</tr>
<tr>
<td>TriggersExecuted</td>
<td>Indicate number of triggers that execute</td>
</tr>
<tr>
<td>TriggerType</td>
<td>Specify type of trigger to execute</td>
</tr>
</tbody>
</table>

**Analog Input Status Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logging</td>
<td>Indicate whether data is being logged to memory or disk file</td>
</tr>
<tr>
<td>Running</td>
<td>Indicate whether device object is running</td>
</tr>
<tr>
<td>SamplesAcquired</td>
<td>Indicate number of samples acquired per channel</td>
</tr>
<tr>
<td>SamplesAvailable</td>
<td>Indicate number of samples available per channel in engine</td>
</tr>
</tbody>
</table>
Analog Input Hardware Configuration Properties

- **ChannelSkew**: Specify time between consecutive scanned hardware channels.
- **ChannelSkewMode**: Specify how channel skew is determined.
- **ClockSource**: Specify clock that governs hardware conversion rate.
- **InputType**: Specify analog input hardware channel configuration.
- **SampleRate**: Specify per-channel rate at which analog data is converted to digital data, or vice versa.

Analog Input Callback Properties

- **DataMissedFcn**: Specify M-file callback function to execute when data is missed.
- **InputOverRangeFcn**: Specify M-file callback function to execute when acquired data exceeds valid hardware range.
- **RuntimeErrorFcn**: Specify M-file callback function to execute when run-time error occurs.
- **SamplesAcquired**: Indicate number of samples acquired per channel.
- **SamplesAcquiredFcn**: Specify M-file callback function to execute when predefined number of samples is acquired for each channel group member.
- **StartFcn**: Specify M-file callback function to execute before device object runs.
- **StopFcn**: Specify M-file callback function to execute after device object runs.
**TimerFcn**
Specify M-file callback function to execute when predefined time period passes

**TimerPeriod**
Specify time period between timer events

**TriggerFcn**
Specify M-file callback function to execute when trigger occurs

### Analog Input General Purpose Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BufferingConfig</td>
<td>Specify per-channel allocated memory</td>
</tr>
<tr>
<td>BufferingMode</td>
<td>Specify how memory is allocated</td>
</tr>
<tr>
<td>Channel</td>
<td>Contain hardware channels added to device object</td>
</tr>
<tr>
<td>EventLog</td>
<td>Store information for specific events</td>
</tr>
<tr>
<td>HwChannel</td>
<td>Specify hardware channel ID</td>
</tr>
<tr>
<td>Name</td>
<td>Specify descriptive name for device object</td>
</tr>
<tr>
<td>Tag</td>
<td>Specify device object label</td>
</tr>
<tr>
<td>Timeout</td>
<td>Specify additional waiting time to extract or queue data</td>
</tr>
<tr>
<td>Type</td>
<td>Indicate device object type, channel, or line</td>
</tr>
<tr>
<td>UserData</td>
<td>Store data to associate with device object</td>
</tr>
</tbody>
</table>

### Channel Properties
The analog input channel properties are given below.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChannelName</td>
<td>Specify descriptive channel name</td>
</tr>
<tr>
<td>HwLine</td>
<td>Specify hardware line ID</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Index</td>
<td>MATLAB index of hardware channel or line</td>
</tr>
<tr>
<td>InputRange</td>
<td>Specify range of analog input subsystem</td>
</tr>
<tr>
<td>NativeOffset</td>
<td>Indicate offset to use when converting between native data format and doubles</td>
</tr>
<tr>
<td>NativeScaling</td>
<td>Indicate scaling to use when converting between native data format and doubles</td>
</tr>
<tr>
<td>Parent</td>
<td>Indicate parent (device object) of channel or line</td>
</tr>
<tr>
<td>SensorRange</td>
<td>Specify range of data expected from sensor</td>
</tr>
<tr>
<td>Type</td>
<td>Indicate device object type, channel, or line</td>
</tr>
<tr>
<td>Units</td>
<td>Specify engineering units label</td>
</tr>
<tr>
<td>UnitsRange</td>
<td>Specify range of data as engineering units</td>
</tr>
</tbody>
</table>

### Analog Output Properties

#### Common Properties

The analog output common properties are grouped into the following categories based on usage.
### Analog Output Basic Setup Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SampleRate</td>
<td>Specify per-channel rate at which analog data is converted to digital data, or vice versa</td>
</tr>
<tr>
<td>TriggerType</td>
<td>Specify type of trigger to execute</td>
</tr>
</tbody>
</table>

### Analog Output Trigger Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitialTriggerTime</td>
<td>Absolute time of first trigger</td>
</tr>
<tr>
<td>TriggerFcn</td>
<td>Specify M-file callback function to execute when trigger occurs</td>
</tr>
<tr>
<td>TriggersExecuted</td>
<td>Indicate number of triggers that execute</td>
</tr>
<tr>
<td>TriggerType</td>
<td>Specify type of trigger to execute</td>
</tr>
</tbody>
</table>

### Analog Output Status Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running</td>
<td>Indicate whether device object is running</td>
</tr>
<tr>
<td>SamplesAvailable</td>
<td>Indicate number of samples available per channel in engine</td>
</tr>
<tr>
<td>SamplesOutput</td>
<td>Indicate number of samples output per channel from engine</td>
</tr>
<tr>
<td>Sending</td>
<td>Indicate whether data is being sent to hardware device</td>
</tr>
</tbody>
</table>
### Analog Output Hardware Configuration Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClockSource</td>
<td>Specify clock that governs hardware conversion rate</td>
</tr>
<tr>
<td>SampleRate</td>
<td>Specify per-channel rate at which analog data is converted to digital data, or vice versa</td>
</tr>
</tbody>
</table>

### Analog Output Data Management Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxSamplesQueued</td>
<td>Indicate maximum number of samples that can be queued in engine</td>
</tr>
<tr>
<td>RepeatOutput</td>
<td>Specify number of additional times queued data is output</td>
</tr>
<tr>
<td>Timeout</td>
<td>Specify additional waiting time to extract or queue data</td>
</tr>
</tbody>
</table>

### Analog Output Callback Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RuntimeErrorFcn</td>
<td>Specify M-file callback function to execute when run-time error occurs</td>
</tr>
<tr>
<td>SamplesOutputFcn</td>
<td>Specify M-file callback function to execute when predefined number of samples is output for each channel group member</td>
</tr>
<tr>
<td>SamplesOutputFcnCount</td>
<td>Specify number of samples to output for each channel group member before samples output event is generated</td>
</tr>
<tr>
<td>StartFcn</td>
<td>Specify M-file callback function to execute before device object runs</td>
</tr>
<tr>
<td>StopFcn</td>
<td>Specify M-file callback function to execute after device object runs</td>
</tr>
</tbody>
</table>
### TimerFcn
Specify M-file callback function to execute when predefined time period passes.

### TimerPeriod
Specify time period between timer events.

### TriggerFcn
Specify M-file callback function to execute when trigger occurs.

## Analog Output General Purpose Properties

- **BufferingConfig**
  Specify per-channel allocated memory.

- **BufferingMode**
  Specify how memory is allocated.

- **Channel**
  Contain hardware channels added to device object.

- **EventLog**
  Store information for specific events.

- **Name**
  Specify descriptive name for device object.

- **OutOfDataMode**
  Specify how value held by analog output subsystem is determined.

- **Tag**
  Specify device object label.

- **Type**
  Indicate device object type, channel, or line.

- **UserData**
  Store data to associate with device object.

## Channel Properties
The analog output channel properties are given below.
### Analog Output Channel Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChannelName</td>
<td>Specify descriptive channel name</td>
</tr>
<tr>
<td>DefaultChannelValue</td>
<td>Specify value held by analog output subsystem</td>
</tr>
<tr>
<td>HwChannel</td>
<td>Specify hardware channel ID</td>
</tr>
<tr>
<td>Index</td>
<td>MATLAB index of hardware channel or line</td>
</tr>
<tr>
<td>NativeOffset</td>
<td>Indicate offset to use when converting between native data format and doubles</td>
</tr>
<tr>
<td>NativeScaling</td>
<td>Indicate scaling to use when converting between native data format and doubles</td>
</tr>
<tr>
<td>OutputRange</td>
<td>Specify range of analog output hardware subsystem</td>
</tr>
<tr>
<td>Parent</td>
<td>Indicate parent (device object) of channel or line</td>
</tr>
<tr>
<td>Type</td>
<td>Indicate device object type, channel, or line</td>
</tr>
<tr>
<td>Units</td>
<td>Specify engineering units label</td>
</tr>
<tr>
<td>UnitsRange</td>
<td>Specify range of data as engineering units</td>
</tr>
</tbody>
</table>

### Digital I/O Properties

#### Common Properties

The digital I/O common properties are given below.
## Digital I/O Common Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line</td>
<td>Contain hardware lines added to device object</td>
</tr>
<tr>
<td>Name</td>
<td>Specify descriptive name for device object</td>
</tr>
<tr>
<td>Running</td>
<td>Indicate whether device object is running</td>
</tr>
<tr>
<td>Tag</td>
<td>Specify device object label</td>
</tr>
<tr>
<td>TimerFcn</td>
<td>Specify M-file callback function to execute when predefined time period passes</td>
</tr>
<tr>
<td>TimerPeriod</td>
<td>Specify time period between timer events</td>
</tr>
<tr>
<td>Type</td>
<td>Indicate device object type, channel, or line</td>
</tr>
<tr>
<td>UserData</td>
<td>Store data to associate with device object</td>
</tr>
</tbody>
</table>

## Line Properties

The digital I/O line properties are given below.

## Digital I/O Line Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Specify whether line is for input or output</td>
</tr>
<tr>
<td>HwLine</td>
<td>Specify hardware line ID</td>
</tr>
<tr>
<td>Index</td>
<td>MATLAB index of hardware channel or line</td>
</tr>
<tr>
<td>LineName</td>
<td>Specify descriptive line name</td>
</tr>
<tr>
<td>Parent</td>
<td>Indicate parent (device object) of channel or line</td>
</tr>
</tbody>
</table>
Getting Command-Line Property Help

To get command-line property help, use the `daqhelp` function. For example, to get help for the `SampleRate` property, type:

```
daqhelp SampleRate
```

**Note** You can use `daqhelp` without creating a device object.

You can also get property characteristics, such as the default property value, using the `propinfo` function. For example, suppose you create the analog input object `ai` for a sound card and you want to find the default value for the `SampleRate` property.

```
ai = analoginput('winsound');
out = propinfo(ai,'SampleRate');
out.DefaultValue
ans =
     8000
```
Base Properties — Alphabetical List
BufferingConfig

**Purpose**
Specify per-channel allocated memory

**Description**
BufferingConfig is a two-element vector that specifies the per-channel allocated memory. The first element of the vector specifies the block size, while the second element of the vector specifies the number of blocks. The total allocated memory (in bytes) is given by

\[(\text{block size})(\text{number of blocks})(\text{number of channels})(\text{native data type})\]

You can determine the native data type with `daqhwinfo`.

You can allocate memory automatically or manually. If `BufferingMode` is `Auto`, the `BufferingConfig` values are automatically set by the engine. If `BufferingMode` is `Manual`, then you must manually set the `BufferingConfig` values. If you change the `BufferingConfig` values, `BufferingMode` is automatically set to `Manual`.

When memory is automatically allocated by the engine, the block-size value depends on the sampling rate and is typically a binary number. The number of blocks is initially set to a value of 30 but can dynamically increase to accommodate the memory requirements. In most cases, the number of blocks used results in a per-channel memory that is somewhat greater than the `SamplesPerTrigger` value. When you manually allocate memory, the number of blocks is not dynamic and care must be taken to ensure there is sufficient memory to store the acquired data. If the number of samples acquired or queued exceeds the allocated memory, then an error is returned.

You can easily determine the memory allocated and available memory for each device object with the `daqmem` function.

**Characteristics**

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Usage</th>
<th>Access</th>
<th>Data type</th>
<th>Read-only when running</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AI, AO, Common</td>
<td>Read/write</td>
<td>Two-element vector of doubles</td>
<td>Yes</td>
</tr>
</tbody>
</table>
**Values**

The default value is determined by the engine, and is based on the number of channels contained by the device object and the sampling rate. The BufferingMode value determines whether the values are automatically updated as data is acquired. For analog output objects, the default number of blocks is zero.

**Examples**

Create the analog input object `ai` for a sound card and add two channels to it.

```matlab
ai = analoginput('winsound');
addchannel(ai,1:2);
```

The block size and number of blocks are given by BufferingConfig, while the native data type for the sound card is given by daqhwinfo.

```matlab
ai.BufferingConfig
ans =
    512   30
out = daqhwinfo(ai);
out.NativeDataType
ans =
    int16
```

With this information, the total allocated memory is calculated to be 61,440 bytes. This number is stored by daqmem.

```matlab
out = daqmem(ai);
out.UsedBytes
ans =
    61440
```

The allocated memory is more than sufficient to store 8000 two-byte samples for two channels. If more memory was required, then the number of blocks would dynamically grow because BufferingMode is set to Auto.
BufferingConfig

See Also

Functions

daqhwinfo, daqmem

Properties

BufferingMode, SampleRate, SamplesPerTrigger
**Purpose**

Specify how memory is allocated

**Description**

BufferingMode can be set to Auto or Manual. If BufferingMode is set to Auto, the data acquisition engine automatically allocates the required memory. If BufferingMode is set to Manual, you must manually allocate memory with the BufferingConfig property.

If BufferingMode is set to Auto and the SampleRate value is changed, then the BufferingConfig values might be recalculated by the engine. Specifically, you can increase (decrease) the block size if SampleRate is increased (decreased). If BufferingMode is set to Auto and you change the BufferingConfig values, then BufferingMode is automatically set to Manual. If BufferingMode is set to Manual, then you cannot set the number of blocks to a value less than three.

For most data acquisition applications, you should set BufferingMode to Auto and have memory allocated by the engine because this minimizes the chance of an out-of-memory condition.

**Characteristics**

- **Usage**: AI, AO, Common
- **Access**: Read/write
- **Data type**: String
- **Read-only when running**: Yes

**Values**

- **{Auto}**: Memory is allocated by the data acquisition engine.
- **Manual**: Memory is allocated manually.
BufferingMode

See Also

Functions

daqmem

Properties

BufferingConfig
Contains hardware channels added to device object

Channel is a vector of all the hardware channels contained by an analog input (AI) or analog output (AO) object. Because a newly created AI or AO object does not contain hardware channels, Channel is initially an empty vector. The size of Channel increases as channels are added with the addchannel function, and decreases as channels are removed using the delete function.

Channel is used to reference one or more individual channels. To reference a channel, you must know its MATLAB index, which is given by the Index property. For example, you must use Channel with the appropriate indices when configuring channel property values.

For scanning hardware, the scan order follows the MATLAB index. Therefore, the hardware channel associated with index 1 is sampled first, the hardware channel associated with index 2 is sampled second, and so on. To change the scan order, you can specify a permutation of the indices with Channel.

**Characteristics**

- **Usage**: AI, AO, Common
- **Access**: Read/write
- **Data type**: Vector of channels
- **Read-only when running**: Yes

**Values**

Values are automatically defined when channels are added to the device object with the addchannel function. The default value is an empty column vector.

**Examples**

Create the analog input object ai for a National Instruments card and add three hardware channels to it.

```matlab
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:2);
```
To set a property value for the first channel added (ID = 0), you must reference the channel by its index using the Channel property.

```matlab
  chans = ai.Channel(1);
  set(chans,'InputRange',[-10 10])
```

Based on the current configuration, the hardware channels are scanned in order from 0 to 2. To swap the scan order of channels 0 and 1, you can specify the appropriate permutation of the MATLAB indices with Channel.

```matlab
  ai.Channel([1 2 3]) = ai.Channel([2 1 3]);
```

See Also  

Functions  

addchannel, delete

Properties  

HwChannel, Index
Purpose  Specify descriptive channel name

Description  `ChannelName` specifies a descriptive name for a hardware channel. If a channel name is defined, then you can reference that channel by its name. If a channel name is not defined, then the channel must be referenced by its index. Channel names are not required to be unique.

You can also define descriptive channel names when channels are added to a device object with the `addchannel` function.

Characteristics

<table>
<thead>
<tr>
<th>Usage</th>
<th>AI, AO, Channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>String</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Values  The default value is an empty string. To reference a channel by name, it must contain only letters, numbers, and underscores and must begin with a letter.

Examples  Create the analog input object `ai` for a sound card and add two channels to it.

```matlab
ai = analoginput('winsound');
addchannel(ai,1:2);
```

To assign a descriptive name to the first channel contained by `ai`:

```matlab
Chan1 = ai.Channel(1)
set(Chan1,'ChannelName','Joe')
```

You can now reference this channel by name instead of by index.

```matlab
set(ai.Joe,'Units','Decibels')
```
<table>
<thead>
<tr>
<th>See Also</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>addchannel</code></td>
</tr>
</tbody>
</table>
Purpose
Specify time between consecutive scanned hardware channels

Description
ChannelSkew applies only to scanning hardware and not to simultaneous sample and hold (SS/H) hardware.

If ChannelSkewMode is set to Minimum or Equisample, then ChannelSkew is automatically set to the appropriate device-specific read-only value. For SS/H hardware, the only valid ChannelSkew value is zero. For some vendors, ChannelSkewMode is automatically set to Manual if you first set ChannelSkew to a valid value.

Characteristics

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Usage</th>
<th>AI, Common</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Read/write</td>
<td>(depends on ChannelSkewMode value)</td>
</tr>
<tr>
<td>Data type</td>
<td>Double</td>
<td></td>
</tr>
<tr>
<td>Read-only when running</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Values
For SS/H hardware, the only valid value is zero. For scanning hardware, the value depends on ChannelSkewMode. ChannelSkew is specified in seconds.

See Also
Properties
ChannelSkewMode
**Purpose**
Specify how channel skew is determined

**Description**
For simultaneous sample and hold (SS/H) hardware, `ChannelSkewMode` is None. For scanning hardware, `ChannelSkewMode` can be Minimum, Equisample, or Manual (Keithley and National Instruments only). SS/H hardware includes Agilent Technologies devices and sound cards, while scanning hardware includes most Measurement Computing, Keithley, and NI boards. Note that some supported boards from these vendors are SS/H, such as Measurement Computing’s PCI-DAS4020/12.

If `ChannelSkewMode` is Minimum, then the minimum channel skew supported by the hardware is used. Some vendors refer to this as burst mode. If `ChannelSkewMode` is Equisample, the channel skew is given by \([(\text{sampling rate})(\text{number of channels})]^{-1}\). If `ChannelSkewMode` is Manual, then you must specify the channel skew with the `ChannelSkew` property. For some vendors, `ChannelSkewMode` is automatically set to Manual if you first set `ChannelSkew` to a valid value.

**Note** If you want to use the maximum sampling rate of your hardware, you should set `ChannelSkewMode` to Equisample.

**Characteristics**

<table>
<thead>
<tr>
<th>Usage</th>
<th>AI, Common</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>String</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Values**

**Advantech**

{Equisample} The channel skew is given by \([(\text{sampling rate})(\text{number of channels})]^{-1}\).
**Agilent Technologies and Sound Cards**

{None}  
This is the only supported value for SS/H hardware.

**Keithley and National Instruments**

{Minimum}  
The channel skew is set to the minimum supported value.

Equisample  
The channel skew is given by \([(\text{sampling rate})(\text{number of channels})]^{-1}\).

Manual  
The channel skew is given by \text{ChannelSkew}.

**Note**  
Keithley and VXI Technology adaptors will be deprecated in a future version of the Toolbox. If you create a Data Acquisition Toolbox object for the 'keithley' or 'hp1432' adaptors in version R2007a, you will receive a warning stating that these adaptors will be removed in a future release.

**Measurement Computing**

{Minimum}  
The channel skew is set to the minimum supported value.

Equisample  
The channel skew is given by \([(\text{sampling rate})(\text{number of channels})]^{-1}\).

**Examples**  
Create an analog input object for Keithley’s KPCI-3108 board and add eight channels.

```matlab
ai = analoginput('keithley',10);
addchannel(ai,0:7);
```
Using the default ChannelSkewMode value of Min and the default SampleRate value of 1000, the corresponding ChannelSkew value is

```matlab
ai.ChannelSkew
ans =
    1.0000e-005
```

To use the maximum sampling rate, set ChannelSkewMode to Equisample.

```matlab
ai.ChannelSkewMode = 'Equi';
ai.Samplerate = 100000/8;
```

**See Also**

**Properties**

ChannelSkew, SampleRate
**Purpose**
Specify clock that governs hardware conversion rate

**Description**
For all supported hardware except Measurement Computing analog output subsystems, ClockSource can be set to Internal, which specifies that the acquisition rate is governed by the internal hardware clock.

For subsystems without a hardware clock, you must use software clocking to govern the sampling rate. Software clocking allows a maximum sampling rate of 500 Hz and a minimum sampling rate of 0.0002 Hz. An error is returned if more than 1 sample of jitter is detected. Note that you might not be able to attain rates over 100 Hz on all systems.

**Characteristics**
- **Usage**: AI, AO, Common
- **Access**: Read/write
- **Data type**: String
- **Read-only when running**: Yes

**Values**

**Advantech and Measurement Computing**

- **{Internal}**: The internal hardware clock is used (AI only).
- **External**: Externally control the channel clock (AI only).
- **Software**: The computer clock is used.

**Agilent Technologies**

- **{Internal}**: The internal hardware clock is used.
- **External**: The external sample clock, positive true.
### ClockSource

| Inverted External | The external sample clock, negative true. |
| VXIBus/3          | The VXI bus clock, divided by 3, provided by some other clock master. |
| VXIBusSample      | The VXI bus sample clock. |

#### Keithley

| {Internal}       | The internal hardware clock is used. |
| External         | Externally control the channel clock. Note that the ChannelSkew property value is honored. |
| Software         | The computer clock is used. |

**Note** Keithley and VXI Technology adaptors will be deprecated in a future version of the Toolbox. If you create a Data Acquisition Toolbox object for the 'keithley' or 'hp1432' adaptors in version R2007a, you will receive a warning stating that these adaptors will be removed in a future release.

#### National Instruments

| {Internal}       | The internal hardware clock is used. |
| External         | Externally control the channel clock (AO only). |
ExternalSampleCtrl

Externally control the channel clock. This value overrides the ChannelSkew property value (AI only). This value does not apply to cards with simultaneous sample and hold.

ExternalScanCtrl

Externally control the scan clock. This value overrides the SampleRate property value (AI only).

ExternalSampleAndScanCtrl

Externally control the channel and scan clocks. This value overrides the ChannelSkew and SampleRate property values (AI only). This value does not apply to cards with simultaneous sample and hold.

Sound Cards

{Internal}

The internal hardware clock is used.

See Also

Properties

ChannelSkew, SampleRate
Purpose
Specify M-file callback function to execute when data is missed

Description
A data missed event is generated immediately after acquired data is missed. This event executes the callback function specified for DataMissedFcn. The default value for DataMissedFcn is daqcallback, which displays the event type and the device object name.

In most cases, data is missed because:

- The engine cannot keep up with the rate of acquisition.
- The driver wrote new data into the hardware’s FIFO buffer before the previously acquired data was read. You can usually avoid this problem by increasing the size of the memory block with the BufferingConfig property.

Data missed event information is stored in the Type and Data fields of the EventLog property. The Type field value is DataMissed. The Data field values are given below.

<table>
<thead>
<tr>
<th>Data Field Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsTime</td>
<td>The absolute time (as a clock vector) the event occurred.</td>
</tr>
<tr>
<td>RelSample</td>
<td>The acquired sample number when the event occurred.</td>
</tr>
</tbody>
</table>

When a data missed event occurs, the analog input object is automatically stopped.
DataMissedFcn

Characteristics

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Usage</th>
<th>Access</th>
<th>Data type</th>
<th>Read-only when running</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AI, Common</td>
<td>Read/write</td>
<td>String</td>
<td>No</td>
</tr>
</tbody>
</table>

Values

The default value is `daqcallback`.

See Also

Functions

`daqcallback`

Properties

`EventLog`
DefaultChannelValue

**Purpose** Specify value held by analog output subsystem

**Description** DefaultChannelValue specifies the value to write to the analog output (AO) subsystem when data is finished being output from the engine.

DefaultChannelValue is used only when OutOfDataMode is set to DefaultValue. This property guarantees that a known value is held by the AO subsystem if a run-time error occurs. Note that sound cards do not have an OutOfDataMode property.

**Characteristics**

- **Usage**: AO, Channel
- **Access**: Read/write
- **Data type**: Double
- **Read-only when running**: Yes

**Values** The default value is zero.

**Examples** Create the analog output object ao and add two channels to it.

```matlab
ao = analogoutput('nidaq','Dev1');
addchannel(ao,0:1);
```

You can configure ao so that when it stops outputting data, a value of 1 volt is held for both channels.

```matlab
ao.OutOfDataMode = 'DefaultValue';
ao.Channel.DefaultChannelValue = 1.0;
```

**See Also** Properties

OutOfRangeDataMode
Purpose
Specify whether line is for input or output

Description
When adding hardware lines to a digital I/O object with addline, you must configure the line direction. The line direction can be In or Out, and is automatically stored in Direction. If a line direction is In, you can only read a value from that line. If a line direction is Out, you can write or read a line value.

For line-configurable devices, you can change individual line directions using Direction. For port-configurable devices, you cannot change individual line directions.

Characteristics

<table>
<thead>
<tr>
<th>Usage</th>
<th>DIO, Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>String</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Values

<table>
<thead>
<tr>
<th>In</th>
<th>The line can be read from.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>The line can be read from or written to.</td>
</tr>
</tbody>
</table>

Examples
Create the digital I/O object dio and add two input lines and two output lines to it.

```plaintext
dio = digitalio('nidaq','Dev1');
addline(dio,0:3,{'In','In','Out','Out'});
```

To configure all lines for output:

```plaintext
dio.Line(1:2).Direction = 'Out';
```
### Direction

<table>
<thead>
<tr>
<th>See Also</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>addline</td>
</tr>
</tbody>
</table>
EventLog

Purpose

Store information for specific events

Description

EventLog is a structure array that stores information related to specific analog input (AI) or analog output (AO) events. Event information is stored in the Type and Data fields of EventLog. Type stores the event type. The logged event types are shown below.

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
<th>AI</th>
<th>AO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data missed</td>
<td>Data is missed by the engine.</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Input overrange</td>
<td>A signal exceeds the hardware input range.</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Run-time error</td>
<td>A run-time error is encountered. Run-time errors include timeouts and hardware errors.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Start</td>
<td>The start function is issued.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Stop</td>
<td>The device object stops executing.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Trigger</td>
<td>A trigger executes.</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Timer events, samples available events (AI), and samples output events (AO) are not logged.

Data stores event-specific information associated with the event type in several fields. For all stored events, Data contains the RelSample field, which returns the input or output sample number at the time the event occurred. For the start, stop, run-time error, and trigger events, Data contains the AbsTime field, which returns the absolute time (as a clock vector) the event occurred. Other event-specific fields are included in Data. For a description of these fields, refer to “Events and Callbacks” on page 5-45 for analog input objects, “Events and Callbacks” on page 6-26 for analog output objects, or the appropriate reference pages in this chapter.
EventLog

EventLog can store a maximum of 1000 events. If this value is exceeded, then the most recent 1000 events are stored. You can use the showdaqevents function to easily display stored event information.

**Characteristics**

- **Usage**: AI, AO, Common
- **Access**: Read-only
- **Data type**: Structure array
- **Read-only when running**: N/A

**Values**

Values are automatically added as events occur. The default value is an empty structure array.

**Examples**

Create the analog input object `ai` and add four channels to it.

```matlab
ai = analoginput('nidaq','Dev1');
chans = addchannel(ai,0:3);
```

Acquire 1 second of data and display the logged event types.

```matlab
start(ai)
events = ai.EventLog;
{events.Type}
anans =
   'Start'    'Trigger'    'Stop

To examine the data associated with the trigger event:

```matlab
events(2).Data
ans =
   AbsTime: [1999 2 12 14 54 52.5456]
   RelSample: 0
   Channel: []
   Trigger: 1
```
See Also: Functions

showdaqevents
HwChannel

**Purpose**
Specify hardware channel ID

**Description**
All channels contained by a device object have a hardware channel ID and an associated MATLAB index. The channel ID is given by `HwChannel` and the MATLAB index is given by the `Index` property. The `HwChannel` value is defined when hardware channels are added to a device object with the `addchannel` function.

The beginning channel ID value depends on the hardware device. For National Instruments hardware, channel IDs are zero-based (begin at zero). For Agilent Technologies hardware and sound cards, channel IDs are one-based (begin at one).

For scanning hardware, the scan order follows the MATLAB index. Therefore, the hardware channel associated with index 1 is sampled first, the hardware channel associated with index 2 is sampled second, and so on. To change the scan order, you can assign the channel IDs to different indices using `HwChannel`.

**Characteristics**

<table>
<thead>
<tr>
<th>Usage</th>
<th>AI, AO, Channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>Double</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Values**

Values are automatically defined when channels are added to the device object with the `addchannel` function. The default value is one.

**Examples**
Create the analog input object `ai` for a National Instruments board and add the first three hardware channels to it.

```matlab
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:2);
```
Based on the current configuration, the hardware channels are scanned in order from 0 to 2. To swap the scan order of channels 0 and 1, you can assign these channels to the appropriate indices using `HwChannel`.

```matlab
ai.Channel(1).HwChannel = 1;
ai.Channel(2).HwChannel = 0;
```

**See Also**

**Functions**

- `addchannel`

**Properties**

- `Channel`, `Index`
**HwLine**

**Purpose**
Specify hardware line ID

**Description**
All lines contained by a digital I/O object have a hardware ID and an associated MATLAB index. The hardware ID is given by `HwLine` and the MATLAB index is given by the `Index` property. The `HwLine` value is defined when hardware lines are added to a digital I/O object with the `addline` function.

The beginning line ID value depends on the hardware device. For National Instruments hardware, line IDs are zero-based (begin at zero).

**Characteristics**

- **Usage**: DIO, Line  
- **Access**: Read/write  
- **Data type**: Double  
- **Read-only when running**: Yes

**Values**

Values are automatically defined when lines are added to the digital I/O object with the `addline` function. The default value is one.

**Examples**
Suppose you create the digital I/O object `dio` and add four hardware lines to it.

```matlab
dio = digitalio('nidaq','Dev1');
addline(dio,0:3,'out');
```

`addline` automatically assigns the indices 1-4 to these hardware lines. You can swap the hardware lines associated with index 1 and index 2 with `HwLine`.

```matlab
dio.Line(1).HwLine = 1;
dio.Line(2).HwLine = 0;
```
## HwLine

### See Also

### Functions
- addline

### Properties
- Line, Index
Index

Purpose

MATLAB index of hardware channel or line

Description

Every hardware channel (line) contained by a device object has an associated MATLAB index that is used to reference that channel (line). For example, to configure property values for an individual channel, you must reference the channel through the `Channel` property using the appropriate `Index` value. Likewise, to configure property values for an individual line, you must reference the line through the `Line` property using the appropriate `Index` value.

For channels (lines), you can assign indices automatically with the `addchannel` (or `addline`) function. Channel (line) indices always begin at 1 and increase monotonically up to the number of channels (lines) contained by the device object. For channels, index assignments can also be made manually with the `addchannel` function.

For scanning hardware, the scan order follows the MATLAB index. Therefore, the hardware channel associated with index 1 is sampled first, the hardware channel associated with index 2 is sampled second, and so on. To change the scan order, you can assign the channel IDs to different indices using the `HwChannel` or `Channel` property.

Index provides a convenient way to access channels and lines programmatically.

Characteristics

Usage: AI, AO, Channel; DIO, Line
Access: Read-only
Data type: Double
Read-only when running: N/A

Values

Values are automatically defined when channels (lines) are added to the device object with the `addchannel` (or `addline`) function. The default value is one.
Examples

Create the analog input object `ai` for a sound card and add two hardware channels to it.

```matlab
ai = analoginput('winsound');
chans = addchannel(ai,1:2);
```

You can access the MATLAB indices for these channels with `Index`.

```matlab
Index1 = chans(1).Index;
Index2 = chans(2).Index;
```

See Also

Functions

`addchannel`, `addline`

Properties

`Channel`, `HwChannel`, `HwLine`, `Line`
**InitialTriggerTime**

**Purpose**
Absolute time of first trigger

**Description**
For all trigger types, InitialTriggerTime records the time when Logging or Sending is set to On. The absolute time is recorded as a clock vector.

You can return the InitialTriggerTime value with the getdata function, or with the Data.AbsTime field of the EventLog property.

**Characteristics**
- **Usage**: AI, AO, Common
- **Access**: Read-only
- **Data type**: Six-element vector of doubles
- **Read-only when running**: N/A

**Values**
The value is automatically updated when the trigger executes. The default value is a vector of zeros.

**Examples**
Create the analog input object ai for a sound card and add two hardware channels to it.

```matlab
ai = analoginput('winsound');
chans = addchannel(ai,1:2);
```

After starting ai, the trigger immediately executes and the trigger time is recorded.

```matlab
start(ai)
abstime = ai.InitialTriggerTime
abstime =
    1.0e+03 *
   1.9990  0.0020  0.0190  0.0130  0.0260  0.0208
```

To convert the clock vector to a more convenient form:
t = fix(abstime);
sprintf('%d:%d:%d', t(4),t(5),t(6))
ans =
13:26:20

See Also

Functions
getdata

Properties
EventLog, Logging, Sending
**InputOverRangeFcn**

**Purpose**
Specify M-file callback function to execute when acquired data exceeds valid hardware range.

**Description**
An input overrange event is generated immediately after an overrange condition is detected for any channel group member. This event executes the callback function specified for `InputOverRangeFcn`.

An overrange condition occurs when an input signal exceeds the range specified by the `SensorRange` property. Overrange detection is enabled only if the analog input object is running and a callback function is specified for `InputOverRangeFcn`.

Input overrange event information is stored in the `Type` and `Data` fields of the `EventLog` property. The `Type` field value is `OverRange`. The `Data` field values are given below.

**Note**
The input overrange event is not generated if a signal begins outside the range and then goes into the range.

<table>
<thead>
<tr>
<th><strong>Data Field Value</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsTime</td>
<td>The absolute time (as a clock vector) the event occurred.</td>
</tr>
<tr>
<td>Channel</td>
<td>The index of the channel that experienced an overrange signal.</td>
</tr>
<tr>
<td>OverRange</td>
<td>The <code>OverRange</code> value, <code>Off</code> indicates that the channel went from overrange to in range, and <code>On</code> indicates that it went from in range to overrange.</td>
</tr>
<tr>
<td>RelSample</td>
<td>The acquired sample immediately before the moment when the overrange transition occurs.</td>
</tr>
</tbody>
</table>
### Characteristics

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>AI, Common</td>
</tr>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>String</td>
</tr>
<tr>
<td>Read-only when</td>
<td>No</td>
</tr>
<tr>
<td>running</td>
<td></td>
</tr>
</tbody>
</table>

### Values

The default value is an empty string.

### See Also

**Properties**

EventLog, SensorRange
**InputRange**

**Purpose**
Specify range of analog input subsystem

**Description**
InputRange is a two-element vector that specifies the range of voltages that can be accepted by the analog input (AI) subsystem. You should configure InputRange so that the maximum dynamic range of your hardware is utilized.

If an input signal exceeds the InputRange value, then an overrange condition occurs. Overrange detection is enabled only if the analog input object is running and a value is specified for the InputOverRangeFcn property. For many devices, the input range is expressed in terms of the gain and polarity.

AI subsystems have a finite number of InputRange values that you can set. If an input range is specified but does not match a valid range, then the next highest supported range is automatically selected by the engine. If InputRange exceeds the range of valid values, then an error is returned. Use the daqhwinfo function to return the input ranges supported by your board.

Because the engine can set the input range to a value that differs from the value you specify, you should return the actual input range for each channel using the get function or the device object display summary. Alternatively, you can use the setverify function, which sets the InputRange value and then returns the actual value that is set.

**Note** If your hardware supports a channel gain list, then you can configure InputRange for individual channels. Otherwise, InputRange must have the same value for all channels contained by the analog input object.

You should use InputRange in conjunction with the SensorRange property. These two properties should be configured such that the maximum precision is obtained and the full dynamic range of the sensor signal is covered.
<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Usage</th>
<th>AI, Channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>Data type</td>
<td>Two-element vector of doubles</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>Read-only when running</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Values**
The default value is supplied by the hardware driver.

**Examples**
Create the analog input object `ai` for a National Instruments board, and add two hardware channels to it.

```matlab
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:1);
```

You can return the input ranges supported by the board with the `InputRanges` field of the `daqhwinfo` function.

```matlab
out = daqhwinfo(ai);
out.InputRanges
ans =
    -0.0500    0.0500
    -0.5000    0.5000
    -5.0000    5.0000
   -10.0000   10.0000
```

To configure both channels contained by `ai` to accept input signals between -10 volts and 10 volts:

```matlab
ai.Channel.InputRange = [-10 10];
```

Alternatively, you can use the `setverify` function.

```matlab
ActualRange = setverify(ai.Channel,'InputRange',[-10 10]);
```
InputRange

See Also

Functions
daqhwinfo, setverify

Properties
InputOverRangeFcn, SensorRange, Units, UnitsRange
### Purpose
Specify analog input hardware channel configuration

### Description
For National Instruments devices, `InputType` can be `SingleEnded`, `Differential`, `NonReferencedSingleEnded`, or `PseudoDifferential`. For Measurement Computing devices, `InputType` can be `SingleEnded`, or `Differential`, for Agilent Technologies devices, `InputType` can only be `Differential`. For sound cards, `InputType` can only be `AC-Coupled`.

If channels have been added to a National Instruments or Measurement Computing analog input object and you change the `InputType` value, then the channels are automatically deleted if the hardware reduces the number of available channels.

### Characteristics
- **Usage**: AI, Common
- **Access**: Read/write
- **Data type**: String
- **Read-only when running**: Yes

### Values

#### Advantech and Measurement Computing
- **Differential**: Channels are configured for differential input.
- **SingleEnded**: Channels are configured for single-ended input.

The value for `InputType` on Advantech and MCC boards is always read-only in MATLAB. For Advantech boards, the setting is made in the Advantech Device Manager. For Measurement Computing boards, the setting is made in InstaCal.

#### Agilent Technologies
- **{Differential}**: Channels are configured for differential input.
**Keithley**

<table>
<thead>
<tr>
<th>InputType</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Differential}</td>
<td>Channels are configured for differential input.</td>
</tr>
<tr>
<td>SingleEnded</td>
<td>Channels are configured for single-ended input.</td>
</tr>
</tbody>
</table>

**National Instruments**

<table>
<thead>
<tr>
<th>InputType</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Differential}</td>
<td>Channels are configured for differential input.</td>
</tr>
<tr>
<td>SingleEnded</td>
<td>Channels are configured for single-ended input.</td>
</tr>
<tr>
<td>NonReferencedSingleEnded</td>
<td>This channel configuration is used when the input signal has its own ground reference, which is tied to the negative input of the instrumentation amplifier.</td>
</tr>
<tr>
<td>PseudoDifferential</td>
<td>Channels are configured for pseudodifferential input, which are all referred to a common ground but this ground is not connected to the computer ground.</td>
</tr>
</tbody>
</table>

**Note** Keithley and VXI Technology adaptors will be deprecated in a future version of the Toolbox. If you create a Data Acquisition Toolbox object for the 'keithley' or 'hp1432' adaptors in version R2007a, you will receive a warning stating that these adaptors will be removed in a future release.
Sound Cards

{AC-Coupled} The input is coupled so that constant (DC) signal levels are suppressed.
Line

**Purpose**

Contain hardware lines added to device object

**Description**

Line is a vector of all the hardware lines contained by a digital I/O (DIO) object. Because a newly created DIO object does not contain hardware lines, Line is initially an empty vector. The size of Line increases as lines are added with the addline function, and decreases as lines are removed with the delete function.

You can use Line to reference one or more individual lines. To reference a line, you must know its MATLAB index and hardware ID. The MATLAB index is given by the Index property, while the hardware ID is given by the HwLine property.

**Characteristics**

<table>
<thead>
<tr>
<th>Usage</th>
<th>DIO, Common</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>Vector of lines</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Values**

Values are automatically defined when lines are added to the DIO object with the addline function. The default value is an empty column vector.

**Examples**

Create the digital I/O object dio and add four input lines to it.

```matlab
dio = digitalio('nidaq','Dev1');
addline(dio,0:3,'In');
```

To set a property value for the first line added (ID = 0), you can reference the line by its index using the Line property.

```matlab
line1 = dio.Line(1);
set(line1,'Direction','Out')
```
See Also

Functions
addline, delete

Properties
HwLine, Index
**LineName**

**Purpose**
Specify descriptive line name

**Description**
`LineName` specifies a descriptive name for a hardware line. If a line name is defined, then you can reference that line by its name. If a line name is not defined, then the line must be referenced by its index. Line names are not required to be unique.

You can also define descriptive line names when lines are added to a digital I/O object with the `addline` function.

**Characteristics**

<table>
<thead>
<tr>
<th>Usage</th>
<th>DIO, Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>String</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Values**
The default value is an empty string. To reference a line by name, it must contain only letters, numbers, and underscores and must begin with a letter.

**Examples**
Create the digital I/O object `dio` and add four hardware lines to it.

```plaintext
dio = digitalio('nidaq','Dev1');
addline(dio,0:3,'out');
```

To assign a descriptive name to the first line contained by `dio`:

```plaintext
line1 = dio.Line(1);
set(line1,'LineName','Joe')
```

You can now reference this line by name instead of index.

```plaintext
set(dio.Joe,'Direction','In')
```
<table>
<thead>
<tr>
<th>See Also</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>addline</td>
</tr>
</tbody>
</table>
**Purpose**
Specify name of disk file information is logged to

**Description**
You can log acquired data, device object property values and event information, and hardware information to a disk file by setting the LoggingMode property to Disk or Disk&Memory.

You can specify any value for LogFileName as long as it conforms to the MATLAB naming conventions: the name cannot start with a number and cannot contain spaces. If no extension is specified as part of LogFileName, then daq is used. The default value for LogFileName is logfile.daq.

You can choose whether an output file is overwritten or if multiple log files are created with the LogToDiskMode property. Setting LogToDiskMode to Overwrite causes the output file to be overwritten. Setting LogToDiskMode to Index causes new data files to be created, each with an indexed name based on the value of LogFileName.

**Characteristics**

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Usage</th>
<th>Access</th>
<th>Data type</th>
<th>Read-only when running</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AI, Common</td>
<td>Read/write</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Values**
The default value is logfile.daq.

**See Also**

**Properties**
Logging, LoggingMode, LogToDiskMode
**Purpose**
Indicate whether data is being logged to memory or disk file

**Description**
Along with the Running property, Logging reflects the state of an analog input object. Logging can be On or Off.

Logging is automatically set to On when a trigger occurs. When Logging is On, acquired data is being stored in memory or to a disk file.

Logging is automatically set to Off when the requested samples are acquired, an error occurs, or a stop function is issued. When Logging is Off, you can still preview data with the peekdata function provided Running is On. However, peekdata does not guarantee that all the requested data is returned.

To guarantee that acquired data contains no gaps, it must be logged to memory or to a disk file. Data stored in memory is extracted with the getdata function, while data stored to disk is returned with the daqread function. The destination for logged data is controlled with the LoggingMode property.

**Characteristics**

<table>
<thead>
<tr>
<th>Usage</th>
<th>AI, Common</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Read-only</td>
</tr>
<tr>
<td>Data type</td>
<td>String</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Values**

<table>
<thead>
<tr>
<th>{Off}</th>
<th>Data is not logged to memory or a disk file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>On</td>
<td>Data is logged to memory or a disk file.</td>
</tr>
</tbody>
</table>
Logging

See Also

Functions
daqread, getdata, peekdata, stop

Properties
LoggingMode, Running
### Purpose
Specify destination for acquired data

### Description
LoggingMode can be set to Disk, Memory, or Disk&Memory. If LoggingMode is set to Disk, then acquired data (as well as device object and hardware information) is streamed to a disk file. If LoggingMode is set to Memory, then acquired data is stored in the data acquisition engine. If LoggingMode is set to Disk&Memory, then acquired data is stored in the data acquisition engine and is streamed to a disk file.

When logging to the engine, you must extract the data with the `getdata` function. If the data is not extracted, it might be overwritten.

When logging to disk, you can specify the log filename with the `LogFileName` property, and you can control the number of log files created with the `LogToDiskMode` property. You can return data stored in a disk file to the MATLAB workspace with the `daqread` function.

### Characteristics

<table>
<thead>
<tr>
<th>Usage</th>
<th>AI, Common</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>String</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk</td>
<td>Acquired data is logged to a disk file.</td>
</tr>
<tr>
<td>{Memory}</td>
<td>Acquired data is logged to memory.</td>
</tr>
<tr>
<td>Disk&amp;Memory</td>
<td>Acquired data is logged to a disk file and to memory.</td>
</tr>
</tbody>
</table>
LoggingMode

See Also

Functions

daqread, getdata

Properties

LogFileName, LogToDiskMode
Purpose

Specify whether data, events, and hardware information are saved to one or more disk files.

Description

LogToDiskMode can be set to Overwrite or Index. If LogToDiskMode is set to Overwrite, then the log file is overwritten each time start is issued. If LogToDiskMode is set to Index, a different disk file is created each time start is issued and these rules are followed:

- The first log filename is specified by the initial value of LogFileName.
- If the specified file already exists, it is overwritten and no warning is issued.
- LogFileName is automatically updated with a numeric identifier after each file is written. For example, if LogFileName is initially specified as data.daq, then data.daq is the first filename, data01.daq is the second filename, and so on.

Separate analog input objects are logged to separate files. You can return data stored in a disk file to the MATLAB workspace with the daqread function. If an error occurs during data logging, an error message is returned and data logging is stopped.

Characteristics

Usage: AI, Common
Access: Read/write
Data type: String
Read-only when running: Yes

Values

Index: Multiple log files are written, each with an indexed filename based on the LogFileName property.

{Overwrite}: The log file is overwritten.
LogToDiskMode

See Also

Functions
daqread

Properties
LogFileName, LoggingMode
**ManualTriggerHwOn**

**Purpose**
Specify hardware device starts at manual trigger

**Description**
You can set `ManualTriggerHwOn` to `Start` or `Trigger`, and it has an effect only when the `TriggerType` property value is `Manual`. If `ManualTriggerHwOn` is `Start`, then the hardware device associated with your device object starts running after you issue the `start` function. If `ManualTriggerHwOn` is `Trigger`, then the hardware device associated with your device object starts running after you issue the `start` function and you execute a manual trigger with the `trigger` function. You can use `trigger` only when you configure the `TriggerType` property to `Manual`.

You should configure `ManualTriggerHwOn` to `Trigger` when you want to synchronize the input and output of data, or you require more control over when your hardware starts. Note that you cannot use `peekdata` or acquire pretrigger data when you use this value. Additionally, you should not use this value with repeated triggers because the subsequent behavior is undefined.

**Characteristics**

<table>
<thead>
<tr>
<th>Usage</th>
<th>AI, Common</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>String</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Values**

- **{Start}**
  Start the hardware after the `start` function is issued.

- **Trigger**
  Start the hardware after the `trigger` function is issued.
**Examples**

Create the analog input object `ai` and the analog output object `ao` for a sound card and add two channels to each device object.

```matlab
ai = analoginput('winsound');
addchannel(ai,1:2);

ao = analogoutput('winsound');
addchannel(ao,1:2);
```

To operate the sound card in full duplex mode, and to minimize the time between when `ai` starts and `ao` starts, you configure `ManualTriggerHwOn` to `Trigger` for `ai` and `TriggerType` to `Manual` for both `ai` and `ao`.

```matlab
set([ai ao],'TriggerType','Manual');
ai.ManualTriggerHwOn = 'Trigger';
```

The analog input and analog output hardware devices will both start after you issue the `trigger` function. For a detailed example that uses `ManualTriggerHwOn`, refer to “Starting Multiple Device Objects” on page 6-38.

**See Also**

**Functions**

`peekdata`, `start`, `trigger`

**Properties**

`TriggerType`
Purpose
Indicate maximum number of samples that can be queued in engine

Description
MaxSamplesQueued indicates the maximum number of samples allowed in the analog output queue.

If the BufferingMode is set to Auto, the default value is calculated by the engine, and is based on the memory resources of your system. You can override the default value of MaxSamplesQueued with the daqmem function.

If the BufferingMode is set to Manual, MaxSamplesQueued is updated to indicate the maximum number of samples allowed in the analog output queue based on the number of buffers selected in BufferingConfig.

The value of MaxSamplesQueued can affect the behavior of putdata. For example, if the queued data exceeds the value of MaxSamplesQueued, then putdata becomes a blocking function until there is enough space in the queue to add the additional data.

Characteristics
Usage
AO, Common
Access
Read-only
Data type
Double
Read-only when running
N/A

Values
The value is calculated by the data acquisition engine.

See Also
Functions
daqmem, putdata
### Purpose
Specify descriptive name for device object

### Description
When a device object is created, a descriptive name is automatically generated and stored in `Name`. This name is produced by concatenating the name of the adaptor, the device ID, and the device object type. You can change the value of `Name` at any time.

### Characteristics
<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Usage</th>
<th>Access</th>
<th>Data type</th>
<th>Read-only when running</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AI, AO, DIO, Common</td>
<td>Read/write</td>
<td>String</td>
<td>No</td>
</tr>
</tbody>
</table>

### Values
The value is defined after the device object is created.

### Examples
Create the analog input object `ai` for a sound card.

```matlab
ai = analoginput('winsound');
```

The descriptive name for `ai` is given by

```matlab
ai.Name
ans =
winsound0-AI
```
**Purpose**
Indicate offset to use when converting between native data format and doubles.

**Description**
NativeOffset, along with NativeScaling, is used to convert data between the native hardware format and doubles.

For analog input objects, you return native data from the engine with the `getdata` function. Additionally, if you log native data to a .daq file, then you can read back that data using the `daqread` function. The formula for converting from native data to doubles is

\[
\text{doubles data} = (\text{native data})(\text{native scaling}) + \text{native offset}
\]

For analog output objects, you queue native data in the engine with the `putdata` function. The formula for converting from doubles to native data is

\[
\text{native data} = (\text{doubles data})(\text{native scaling}) + \text{native offset}
\]

You return the native data type of your hardware device with the `daqhwinfo` function. Note that the `NativeScaling` value for a given channel might change if you change its `InputRange` (AI) or `OutputRange` (AO) property value.

You might want to return or queue data in native format to conserve memory and to increase data acquisition or data output speed.

**Characteristics**
- **Usage**: AI, AO, Channel
- **Access**: Read-only
- **Data type**: Double
- **Read-only when running**: N/A

**Values**
The default value is device-specific.
Examples

Create the analog input object ai for a National Instruments board, and add eight channels to it.

    ai = analoginput('nidaq','Dev1');
    addchannel(ai,0:7);

Start ai, collect one second of data for each channel, and extract the data from the engine using the native format of the device.

    start(ai)
    nativedata = getdata(ai,1000,'native');

You can return the native data type of the board with the daqhwinfo function.

    out = daqhwinfo(ai);
    out.NativeDataType
    ans =
        int16

Convert the data to doubles using the NativeScaling and NativeOffset properties.

    scaling = get(ai.Channel(1),'NativeScaling');
    offset = get(ai.Channel(1),'NativeOffset');
    data = double(nativedata)*scaling + offset;

See Also

Functions

daqhwinfo, daqread, getdata, putdata

Properties

InputRange, NativeScaling, OutputRange
Purpose
Indicate scaling to use when converting between native data format and doubles

Description
NativeScaling, along with NativeOffset, is used to convert data between the native hardware format and doubles.

For analog input objects, you return native data from the engine with the getdata function. Additionally, if you log native data to a .daq file, then you can read back that data using the daqread function. The formula for converting from native data to doubles is

doubles data = (native data)(native scaling) + native offset

For analog output objects, you queue native data in the engine with the putdata function. The formula for converting from doubles to native data is

native data = (doubles data)(native scaling) + native offset

You return the native data type of your hardware device with the daqhwinfo function. Note that the NativeScaling value for a given channel might change if you change its InputRange (AI) or OutputRange (AO) property value.

You might want to return or queue data in native format to conserve memory and to increase data acquisition or data output speed.

Characteristics

- **Usage**: AI, AO, Channel
- **Access**: Read-only
- **Data type**: Double
- **Read-only when running**: N/A

Values
The default value is device-specific.
NativeScaling

See Also

Functions

daqhwinfo, daqread, getdata, putdata

Properties

InputRange, NativeOffset, OutputRange
### Purpose

Specify range of analog output hardware subsystem

### Description

`OutputRange` is a two-element vector that specifies the range of voltages that can be output by the analog output (AO) subsystem. You should configure `OutputRange` so that the maximum dynamic range of your hardware is utilized. For many devices, the output range is expressed in terms of the gain and polarity.

AO subsystems have a finite number of `OutputRange` values that you can set. If an output range is specified but does not match a valid range, then the next highest supported range is automatically selected by the engine. If `OutputRange` exceeds the range of valid values, then an error is returned. Use the `daqhwinfo` function to return the output ranges supported by your board.

Because the engine can set the output range to a value that differs from the value you specify, you should return the actual output range for each channel using the `get` function or the device object display summary. Alternatively, you can use the `setverify` function, which sets the `OutputRange` value and then returns the actual value that is set.

### Characteristics

- **Usage**: AO, Channel
- **Access**: Read/write
- **Data type**: Two-element vector of doubles
- **Read-only when running**: Yes

### Values

The default value is determined by the hardware driver.

### Examples

Create the analog output object `ao` for a National Instruments board and add two hardware channels to it.

```matlab
ao = analogoutput('nidaq','Dev1');
addchannel(ao,0:1);
```
You can return the output ranges supported by the board with the `OutputRanges` field of the `daqhwinfo` function.

```matlab
out = daqhwinfo(ao);
out.OutputRanges
ans =
    0.0000   10.0000
   -10.0000   10.0000
```

To configure both channels contained by `ao` to output signals between -10 volts and 10 volts:

```matlab
ao.Channel.OutputRange = [-10 10];
```

Alternatively, you can use the `setverify` function to configure and return the `OutputRange` value.

```matlab
ActualRange = setverify(ao.Channel,'OutputRange',[-10 10]);
```

**See Also**

- **Functions**
  - `daqhwinfo`, `setverify`

- **Properties**
  - `Units`, `UnitsRange`
Parent

Purpose
Indicate parent (device object) of channel or line

Description
The parent of a channel (line) is defined as the device object that contains the channel (line).

You can create a copy of the device object containing a particular channel or line by returning the value of Parent. You can treat this copy like any other device object. For example, you can configure property values, add channels or lines to it, and so on.

Characteristics
Usage        AI, AO, Channel; DIO, Line
Access       Read-only
Data type    Device object
Read-only when running  N/A

Values
The value is defined when channels or lines are added to the device object.

Examples
Create the analog input object ai for a National Instruments board and add three hardware channels to it.

```
ai = analoginput('nidaq','Dev1');
chans = addchannel(ai,0:2);
```

To return the parent for channel 2:

```
parent = ai.Channel(2).Parent;
```

```
parent is an exact copy of ai.
```

```
isequal(ai,parent)
an =
1
```
Specify port ID

Hardware lines are often grouped together as a port. Digital I/O subsystems can consist of multiple ports and typically have eight lines per port. When adding hardware lines to a digital I/O object with addline, you can specify the port ID. The port ID is stored in the Port property. If the port ID is not specified, then the smallest port ID value is automatically used.

Usage DIO, Line

Access Read-only

Data type Double

Read-only when running N/A

The port ID is defined when line are added to the digital I/O object with addline.

Create the digital I/O object dio and add two hardware channels to it.

```matlab
dio = digitalio('nidaq','Dev1');
addline(dio,0:1,'In');
```

You can use Port property to return the port IDs associated with the lines contained by dio.

```matlab
dio.Line.Port
ans =
    [0]
    [0]
```

See Also Functions

addline
RepeatOutput

**Purpose**
Specify number of additional times queued data is output

**Description**
To send data to an analog output subsystem, it must first be queued in the data acquisition engine with the `putdata` function. If you want to continuously output the same data, you can use multiple calls to `putdata`. However, because each `putdata` call consumes memory, a long output sequence can quickly bring your system to halt.

As an alternative to `putdata`, you can continuously output previously queued data using `RepeatOutput`. Because `RepeatOutput` requeues the data, additional memory resources are not consumed. While the data is being output, you cannot add additional data to the queue.

**Characteristics**

<table>
<thead>
<tr>
<th>Usage</th>
<th>AO, Common</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>Double</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Values**
The default value is zero.

**Examples**
Create the analog output object `ao` for a sound card and add one channel to it.

```matlab
ao = analogoutput('winsound');
chans = addchannel(ao,1);
```

To queue one second of data:

```matlab
data = sin(linspace(0,10,8000))';
putdata(ao,data)
```

To continuously output data for 10 seconds:

```matlab
set(ao,'RepeatOutput',9)
```
RepeatOutput

See Also

Functions

putdata
**Purpose**
Indicate whether device object is running

**Description**
Along with the Logging or Sending property, Running reflects the state of an analog input or analog output object. Running can be On or Off.

Running is automatically set to On once the start function is issued. When Running is On, you can acquire data from an analog input device or send data to an analog output device after the trigger occurs. For digital I/O objects, Running is typically used to indicate if time-based events are being generated.

Running is automatically set to Off once the stop function is issued, the specified data is acquired or sent, or a run-time error occurs. When Running is Off, you cannot acquire or send data. However, you can acquire one sample with the getsample function, or send one sample with the putsample function.

**Characteristics**

<table>
<thead>
<tr>
<th>Usage</th>
<th>AI, AO, DIO, Common</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Read-only</td>
</tr>
<tr>
<td>Data type</td>
<td>String</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Values**

- **{Off}** The device object is not running.
- **On** The device object is running.

**See Also**

**Functions**
getsample, putsample, start

**Properties**
Logging, Sending
**Purpose**

Specify M-file callback function to execute when run-time error occurs

**Description**

A run-time error event is generated immediately after a run-time error occurs. This event executes the callback function specified for `RuntimeErrorFcn`. Additionally, a toolbox error message is automatically displayed to the MATLAB workspace. If an error occurs that is not explicitly handled by the toolbox, then the hardware-specific error message is displayed.

The default value for `RuntimeErrorFcn` is `daqcallback`, which displays the event type, the time the event occurred, and the device object name along with the error message.

Run-time error event information is stored in the `Type` and `Data` fields of the `EventLog` property. The `Type` field value is `Error`. The `Data` field values are given below.

<table>
<thead>
<tr>
<th>Data Field Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsTime</td>
<td>The absolute time (as a clock vector) the event occurred.</td>
</tr>
<tr>
<td>RelSample</td>
<td>The acquired (AI) or output (AO) sample number when the event occurred.</td>
</tr>
<tr>
<td>String</td>
<td>The descriptive error message.</td>
</tr>
</tbody>
</table>

Run-time errors include hardware errors and timeouts. Run-time errors do not include configuration errors such as setting an invalid property value.
### Characteristics

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Usage</th>
<th>Access</th>
<th>Data type</th>
<th>Read-only when running</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AI, AO, Common</td>
<td>Read/write</td>
<td>String</td>
<td>No</td>
</tr>
</tbody>
</table>

### Values

The default value is `daqcallback`.

### See Also

**Functions**

`daqcallback`

**Properties**

`EventLog`, `Timeout`
SampleRate

### Purpose
Specify per-channel rate at which analog data is converted to digital data, or vice versa.

### Description
**SampleRate** specifies the per-channel rate (in samples/second) that an analog input (AI) or analog output (AO) subsystem converts data. AI subsystems convert analog data to digital data, while AO subsystems convert digital data to analog data.

AI and AO subsystems have a finite (though often large) number of valid sampling rates. If you specify a sampling rate that does not match one of the valid values, the data acquisition engine automatically selects the nearest available sampling rate.

Because the engine can set the sampling rate to a value that differs from the value you specify, you should return the actual sampling rate using the `get` function or the device object display summary. Alternatively, you can use the `setverify` function, which sets the `SampleRate` value and then returns the actual value that is set. To find out the range of sampling rates supported by your board, use the `propinfo` function.

Additionally, because the actual sampling rate depends on the number of channels contained by the device object and the `ChannelSkew` property value (AI only), `SampleRate` should be the last property you set before starting the device object.

### Characteristics
- **Usage**: AI, AO, Common
- **Access**: Read/write
- **Data type**: Double
- **Read-only when running**: Yes

### Values
The default value is obtained from the hardware driver.
Examples

Create the analog input object `ai` for a sound card and add two channels to it.

```matlab
ai = analoginput('winsound');
addchannel(ai,1:2);
```

You can find out the range of valid sampling rates with the `ConstraintValue` field of the `propinfo` function.

```matlab
rates = propinfo(ai,'SampleRate');
rates.ConstraintValue
ans =
   8000   48000
```

To configure the per-channel sampling rate to 48 kHz:

```matlab
set(ai,'SampleRate',48000)
```

Alternatively, you can use the `setverify` function to configure and return the `SampleRate` value.

```matlab
ActualRate = setverify(ai,'SampleRate',48000);
```

See Also

Functions

`propinfo`, `setverify`

Properties

`ChannelSkew`
**Purpose**
Indicate number of samples acquired per channel

**Description**
SamplesAcquired is continuously updated to reflect the current number of samples acquired by an analog input object. It is reset to zero after a start function is issued.

Use the SamplesAvailable property to find out how many samples are available to be extracted from the engine.

**Characteristics**

<table>
<thead>
<tr>
<th>Usage</th>
<th>AI, Common</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Read-only</td>
</tr>
<tr>
<td>Data type</td>
<td>Double</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Values**
The value is continuously updated to reflect the current number of samples acquired. The default value is zero.

**See Also**

**Functions**
start

**Properties**
SamplesAvailable
SamplesAcquiredFcn

**Purpose**
Specify M-file callback function to execute when predefined number of samples is acquired for each channel group member.

**Description**
A samples acquired event is generated immediately after the number of samples specified by the SamplesAcquiredFcnCount property is acquired for each channel group member. This event executes the callback function specified for SamplesAcquiredFcn.

The samples acquired event is executed regardless of its waiting time in the queue.

Use SamplesAcquiredFcn if you need access each sample that is acquired. Otherwise you can use the TimerFcn property.

Samples acquired event information is not stored in the EventLog property. When the callback function is executed, the second argument is a structure containing two fields. The Type field value is set to the string 'SamplesAcquired', and the Data field values are given below.

<table>
<thead>
<tr>
<th>Data Field Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsTime</td>
<td>The absolute time (as a clock vector) the event occurred.</td>
</tr>
<tr>
<td>RelSample</td>
<td>The acquired sample number when the event occurred.</td>
</tr>
</tbody>
</table>

**Characteristics**
Usage: AI, Common
Access: Read/write
Data type: String
Read-only when running: No

**Values**
The default value is an empty string.
See Also

Properties

EventLog, SamplesAcquiredFcnCount, TimerFcn
**Purpose**
Specify number of samples to acquire for each channel group member before samples acquired event is generated.

**Description**
A samples acquired event is generated immediately after the number of samples specified by `SamplesAcquiredFcnCount` is acquired for each channel group member. This event executes the callback function specified by the `SamplesAcquiredFcn` property.

**Characteristics**
- Usage: AI, Common
- Access: Read/write
- Data type: Double
- Read-only when running: Yes

**Values**
The default value is 1024.

**See Also**
- Properties
  
- `SamplesAcquiredFcn`
### Purpose
Indicate number of samples available per channel in engine

### Description
For analog input (AI) objects, `SamplesAvailable` indicates the number of samples that can be extracted from the engine for each channel group member with the `getdata` function. For analog output (AO) objects, `SamplesAvailable` indicates the number of samples that have been queued with the `putdata` function, and can be sent (output) to each channel group member.

After data has been extracted (AI) or output (AO), the `SamplesAvailable` value is reduced by the appropriate number of samples. For AI objects, `SamplesAvailable` is reset to zero after a `start` function is issued.

For AI objects, use the `SamplesAcquired` property to find out how many samples have been acquired since the `start` function was issued. For AO objects, use the `SamplesOutput` property to find out how many samples have been output since the `start` function was issued.

### Characteristics
<table>
<thead>
<tr>
<th>Usage</th>
<th>AI, AO, Common</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Read-only</td>
</tr>
<tr>
<td>Data type</td>
<td>Double</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Values
The value is automatically updated based on the number of samples acquired (analog input) or sent (analog output). The default value is zero.

### See Also
- **Functions**
  - start
- **Properties**
  - `SamplesAcquired`, `SamplesOutput`
**Purpose**
Indicate number of samples output per channel from engine

**Description**
SamplesOutput is continuously updated to reflect the current number of samples output by an analog output object. It is reset to zero after the device objects stops and data has been queued with the putdata function.

Use the SamplesAvailable property to find out how many samples are available to be output from the engine.

**Characteristics**

<table>
<thead>
<tr>
<th>Usage</th>
<th>AO, Common</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Read-only</td>
</tr>
<tr>
<td>Data type</td>
<td>Double</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Values**
The value is continuously updated to reflect the current number of samples output. The default value is zero.

**See Also**

**Functions**
putdata

**Properties**

SamplesAvailable
SamplesOutputFcn

**Purpose**
Specify M-file callback function to execute when predefined number of samples is output for each channel group member.

**Description**
A samples output event is generated immediately after the number of samples specified by the `SamplesOutputFcnCount` property is output for each channel group member. This event executes the callback function specified for `SamplesOutputFcn`.

Samples output event information is not stored in the `EventLog` property. When the callback function is executed, the second argument is a structure containing two fields. The `Type` field value is set to the string 'SamplesOutput', and the event `Data` field values are given below.

<table>
<thead>
<tr>
<th>Data Field Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsTime</td>
<td>The absolute time (as a clock vector) the event occurred.</td>
</tr>
<tr>
<td>RelSample</td>
<td>The acquired sample number when the event occurred.</td>
</tr>
</tbody>
</table>

**Characteristics**
- **Usage**: AO, Common
- **Access**: Read/write
- **Data type**: String
- **Read-only when running**: No

**Values**
The default value is an empty string.

**See Also**
- **Properties**
  - `EventLog`, `SamplesOutputFcnCount`
**SamplesOutputFcnCount**

**Purpose**
Specify number of samples to output for each channel group member before samples output event is generated.

**Description**
A samples output event is generated immediately after the number of samples specified by `SamplesOutputFcnCount` is output for each channel group member. This event executes the callback function specified by the `SamplesOutputFcn` property.

**Characteristics**
- **Usage**: AO, Common
- **Access**: Read/write
- **Data type**: Double
- **Read-only when running**: Yes

**Values**
The default value is 1024.

**See Also**
- **Properties**
  - `SamplesOutputFcn`
**Purpose**
Specify number of samples to acquire for each channel group member for each trigger that occurs.

**Description**
SamplesPerTrigger specifies the number of samples to acquire for each analog input channel group member for each trigger that occurs. If SamplesPerTrigger is set to Inf, then the analog input object continually acquires data until a stop function is issued or an error occurs.

The default value of SamplesPerTrigger is calculated by the data acquisition engine such that one second of data is acquired. This calculation is based on the value of SampleRate.

**Characteristics**
- **Usage**: AI, Common
- **Access**: Read/write
- **Data type**: Double
- **Read-only when running**: Yes

**Values**
The default value is set by the engine such that one second of data is acquired.

**Examples**
Create the analog input object ai for a sound card and add two channels to it.

```matlab
ai = analoginput('winsound');
addchannel(ai,1:2);
```

By default, a one second acquisition in which 8000 samples are acquired for each channel is defined. To define a two second acquisition at the same sampling rate:

```matlab
set(ai,'SamplesPerTrigger',16000)
```
See Also

Functions
stop

Properties
SampleRate
Sending

Purpose
Indicate whether data is being sent to hardware device

Description
Along with the Running property, Sending reflects the state of an analog output object. Sending can be On or Off.

Sending is automatically set to On when a trigger occurs. When Sending is On, queued data is being output to the analog output subsystem. Sending is automatically set to Off when the queued data has been output, an error occurs, or a stop function is issued. When Sending is Off, data is not being output to the analog output subsystem although you can output a single sample with the putsample function.

Characteristics
- Usage: AO, Common
- Access: Read-only
- Data type: String
- Read-only when running: N/A

Values
- {Off} Data is not being sent to the analog output hardware.
- On Data is being sent to the analog output hardware.

See Also
Functions
putsample

Properties
Running
### Purpose
Specify range of data expected from sensor

### Description
You use `SensorRange` to scale your data to reflect the range you expect from your sensor. You can find the appropriate sensor range from your sensor’s specification sheet. For example, an accelerometer might have a sensor range of ±5 volts, which corresponds to ±50 g’s (1 g = 9.80 m/s/s).

The data is scaled while it is extracted from the engine with the `getdata` function according to the formula

\[
\text{scaled value} = \frac{(A/D \text{ value})(\text{units range})}{(\text{sensor range})}
\]

The A/D value is constrained by the `InputRange` property, which reflects the gain and polarity of your hardware channels. The units range is given by the `UnitsRange` property.

### Characteristics

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Usage</th>
<th>Access</th>
<th>Data type</th>
<th>Read-only when running</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>AI, Channel</td>
<td>Read/write</td>
<td>Two-element vector of doubles</td>
<td>No</td>
</tr>
</tbody>
</table>

### Values
The default value is determined by the default value of the `InputRange` property.

### See Also
**Functions**
- `getdata`

**Properties**
- `InputRange`
- `Units`
- `UnitsRange`
**StartFcn**

**Purpose**
Specify M-file callback function to execute before device object runs

**Description**
A start event is generated immediately after the start function is issued. This event executes the callback function specified for StartFcn. When the callback function has finished executing, Running is automatically set to On and the device object and hardware device begin executing. Note that the device object is not started if an error occurs while executing the callback function.

Start event information is stored in the Type and Data fields of the EventLog property. The Type field value is Start. The Data field values are given below.

<table>
<thead>
<tr>
<th>Data Field Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsTime</td>
<td>The absolute time (as a clock vector) the event occurred.</td>
</tr>
<tr>
<td>RelSample</td>
<td>The acquired (AI) or output (AO) sample number when the event occurred.</td>
</tr>
</tbody>
</table>

**Characteristics**

<table>
<thead>
<tr>
<th>Usage</th>
<th>AI, AO, Common</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>String</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>No</td>
</tr>
</tbody>
</table>

**Values**
The default value is an empty string.
See Also

Functions
start

Properties
EventLog, Running
**Purpose** Specify M-file callback function to execute after device object runs.

**Description** A stop event is generated immediately after the device object and hardware device stop executing. This occurs when:

- A stop function is issued.
- For analog input (AI) objects, the requested number of samples to acquire was reached or data was missed. For analog output (AO) objects, the requested number of samples to output was reached.
- A run-time error occurred.

A stop event executes the callback function specified for StopFcn. Under most circumstances, the callback function is not guaranteed to complete execution until sometime after the device object and hardware device stop, and the Running property is set to Off.

Stop event information is stored in the Type and Data fields of the EventLog property. The Type field value is Stop. The Data field values are given below.

<table>
<thead>
<tr>
<th>Data Field Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsTime</td>
<td>The absolute time (as a clock vector) the event occurred.</td>
</tr>
<tr>
<td>RelSample</td>
<td>The acquired (AI) or output (AO) sample number when the event occurred.</td>
</tr>
</tbody>
</table>

**Characteristics**

- **Usage**: AI, AO, Common
- **Access**: Read/write
- **Data type**: String
- **Read-only when running**: No
<table>
<thead>
<tr>
<th>Values</th>
<th>The default value is an empty string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>See Also</td>
<td>Functions</td>
</tr>
<tr>
<td></td>
<td>stop</td>
</tr>
<tr>
<td></td>
<td>Properties</td>
</tr>
<tr>
<td></td>
<td>EventLog, Running</td>
</tr>
</tbody>
</table>
**Tag**

**Purpose**
Specify device object label

**Description**
Tag provides a means to identify device objects with a label. Using the `daqfind` function and the Tag value, you can identify and retrieve a device object that was cleared from the MATLAB workspace.

**Characteristics**
- **Usage**: AI, AO, DIO, Common
- **Access**: Read/write
- **Data type**: String
- **Read-only when running**: No

**Values**
The default value is an empty string.

**Examples**
Create the analog input object ai for a sound card and add two channels to it.

```matlab
ai = analoginput('winsound');
addchannel(ai,1:2);
```

Assign ai a label using Tag.

```matlab
set(ai,'Tag','Sound')
```

If ai is cleared from the workspace, you can use `daqfind` and the Tag value to identify and retrieve the device object.

```matlab
clear ai
aicell = daqfind('Tag','Sound');
ai = aicell{1};
```

**See Also**

**Functions**
daqfind
Purpose
Specify additional waiting time to extract or queue data

Description
The `Timeout` value (in seconds) is added to the time required to extract data from the engine or queue data to the engine. Because data is extracted with the `getdata` function, and queued with the `putdata` function, `Timeout` is associated only with these two "blocking" functions.

If the requested data is not extracted or queued after waiting the required time, then a timeout condition occurs and control is immediately returned to MATLAB. A timeout is one of the conditions for stopping an acquisition. When a timeout occurs, the callback function specified by `RuntimeErrorFcn` is called.

`Timeout` is not associated with hardware timeout conditions. Possible hardware timeout conditions include

- Triggering on a voltage level and that level never occurs
- Externally clocking an acquisition and the external clock signal never occurs
- Losing the hardware connection

To check for hardware timeouts, you might need to poll the appropriate property value.

Characteristics

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Usage</th>
<th>Access</th>
<th>Data type</th>
<th>Read-only when running</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AI, AO, Common</td>
<td>Read/write</td>
<td>Double</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Values
The default value is one second.
See Also

Functions
getdata, putdata

Properties
RuntimeErrorFcn
**Purpose**

Specify M-file callback function to execute when predefined time period passes.

**Description**

A timer event is generated whenever the time specified by the `TimerPeriod` property passes. This event executes the callback function specified for `TimerFcn`. Time is measured relative to when the device object starts running.

Some timer events might not be processed if your system is significantly slow or if the `TimerPeriod` value is too small. For example, displaying data, which is a common application for timer events can be dropped as it is a CPU-intensive task. A timer event can be dropped if there is a newer event in the queue. To guarantee that events are not dropped, you should use the `SamplesAcquiredFcn` property (analog input) or the `SamplesOutputFcn` property (analog output).

**Note** Use the `SamplesAvailable` property to see the number of samples in the queue before you execute the timer event.

For digital I/O objects, timer events are typically used to update and display the state of the device object.

Timer event information is not stored in the `EventLog` property. When the callback function is executed, the second argument is a structure containing two fields. The `Type` field value is set to the string 'Timer', and the event `Data` field value is given below.

<table>
<thead>
<tr>
<th>Data Field Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsTime</td>
<td>The absolute time (as a clock vector) the event occurred.</td>
</tr>
</tbody>
</table>
### TimerFcn

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Usage</th>
<th>AI, AO, DIO, Common</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td></td>
<td>Data type</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Read-only when running</td>
<td>No</td>
</tr>
</tbody>
</table>

### Values

The default value is an empty string.

### See Also

**Properties**

- EventLog
- SamplesAcquiredFcn
- SamplesOutputFcn
- TimerPeriod
**Purpose**

Specify time period between timer events

**Description**

`TimerPeriod` specifies the time, in seconds, that must pass before the callback function specified for `TimerFcn` is called. Time is measured relative to when the hardware device starts running.

Some timer events might not be processed if your system is significantly slowed or if the `TimerPeriod` value is too small. For example, a common application for timer events is to display data. However, because displaying data is a CPU-intensive task, some of these events might be dropped.

**Characteristics**

- **Usage**: AI, AO, DIO, Common
- **Access**: Read/write
- **Data type**: Double
- **Read-only when running**: No

**Values**

The default value is 0.1 second.

**See Also**

Properties

`TimerFcn`
## TriggerChannel

**Purpose**
Specify channel serving as trigger source

**Description**
TriggerChannel specifies the channel serving as the trigger source. The trigger channel must be specified before the trigger type. You might need to configure the TriggerCondition and TriggerConditionValue properties in conjunction with TriggerChannel.

For all supported vendors, if TriggerType is Software, then you must acquire data from the channel being used for the trigger source. For National Instruments hardware, if TriggerType is HwAnalogChannel, then TriggerChannel must be the first element of the channel group. The exception is if you are using simultaneous acquisition devices such as the S-series boards, with which you can specify any channel for the TriggerChannel value.

**Characteristics**
- **Usage**: AI, Common
- **Access**: Read/write
- **Data type**: Vector of channels
- **Read-only when running**: Yes

**Values**
The default value is an empty vector.

**Examples**
Create the analog input object `ai`, add two channels, and define the trigger source as channel 2.

```matlab
ai = analoginput('winsound');
ch = addchannel(ai,1:2);
set(ai,'TriggerChannel',ch(2))
set(ai,'TriggerType','Software')
```

**See Also**
Properties
- TriggerCondition
- TriggerConditionValue
- TriggerType
### Purpose
Specify condition that must be satisfied before trigger executes

### Description
The available trigger conditions depend on the value of TriggerType. If TriggerType is Immediate or Manual, the only available TriggerCondition is None. If TriggerType is Software, then TriggerCondition can be Rising, Falling, Leaving, or Entering. These trigger conditions require one or more voltage values to be specified for the TriggerConditionValue property.

Based on the hardware you are using, additional trigger conditions might be available.

### Characteristics

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Usage</th>
<th>Access</th>
<th>Data type</th>
<th>Read-only when running</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>AI, Common</td>
<td>Read/write</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Values

#### All Supported Hardware

The following trigger condition is used when TriggerType is Immediate or Manual.

- **{None}**
  - No trigger condition is required.

The following trigger conditions are available when TriggerType is Software.

- **{Rising}**
  - The trigger occurs when the signal has a positive slope when passing through the specified value.

- **Falling**
  - The trigger occurs when the signal has a negative slope when passing through the specified value.
# Trigger Condition

<table>
<thead>
<tr>
<th>Leaving</th>
<th>The trigger occurs when the signal leaves the specified range of values.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entering</td>
<td>The trigger occurs when the signal enters the specified range of values.</td>
</tr>
</tbody>
</table>

## Agilent Technologies

The following trigger conditions are available when TriggerType is HwDigital.

- **{PositiveEdge}**
  - The trigger occurs when the positive (rising) edge of a digital signal is detected.

- **NegativeEdge**
  - The trigger occurs when the negative (falling) edge of a digital signal is detected.

The following trigger conditions are available when TriggerType is HwAnalog.

- **{Rising}**
  - The trigger occurs when the analog signal has a positive slope when passing through the specified range of values.

- **Falling**
  - The trigger occurs when the analog signal has a negative slope when passing through the specified range of values.

- **Leaving**
  - The trigger occurs when the analog signal leaves the specified range of values.

- **Entering**
  - The trigger occurs when the analog signal enters the specified range of values.

Note that when TriggerType is HwAnalog, the trigger condition values are all specified as two-element vectors. Setting two trigger levels prevents the module from triggering repeatedly because of a noisy signal.
**Keithley**

The following trigger conditions are available when TriggerType is HwDigital.

- **{PositiveEdge}** The trigger occurs when the positive (rising) edge of the digital signal is detected.
- **NegativeEdge** The trigger occurs when the negative (falling) edge of the digital signal is detected.
- **GateHigh** Gated acquisition on a TTL high level on TGIN or the 8254 gate input. Note that gated acquisition mode ignores all stop trigger properties.
- **GateLow** Gated acquisition on a TTL low level on TGIN. This option is invalid and causes an error if the device’s analog input gating is set to 8254 Gate in the DriverLINX Configuration Analog Input Subsystem panel.

To utilize simultaneous gated and triggered acquisition, set the analog input gating to 8254 Gate in the DriverLINX Configuration Panel Analog Input subsystem, and use GateHigh and HwDigital triggering through the TGIN connection pin.

**Note** Keithley and VXI Technology adaptors will be deprecated in a future version of the Toolbox. If you create a Data Acquisition Toolbox object for the 'keithley' or 'hp1432' adaptors in version R2007a, you will receive a warning stating that these adaptors will be removed in a future release.

**Measurement Computing**

The following trigger conditions are available when TriggerType is HwDigital.
<table>
<thead>
<tr>
<th>Trigger Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GateHigh</td>
<td>The trigger occurs as long as the digital signal is high.</td>
</tr>
<tr>
<td>GateLow</td>
<td>The trigger occurs as long as the digital signal is low.</td>
</tr>
<tr>
<td>TrigHigh</td>
<td>The trigger occurs when the digital signal is high.</td>
</tr>
<tr>
<td>TrigLow</td>
<td>The trigger occurs when the digital signal is low.</td>
</tr>
<tr>
<td>TrigPosEdge</td>
<td>The trigger occurs when the positive (rising) edge of the digital signal is detected.</td>
</tr>
<tr>
<td>TrigNegEdge</td>
<td>The trigger occurs when the negative (falling) edge of the digital signal is detected.</td>
</tr>
</tbody>
</table>

The following trigger conditions are available when `TriggerType` is `HwAnalog`:

<table>
<thead>
<tr>
<th>Trigger Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{TrigAbove}</td>
<td>The trigger occurs when the analog signal makes a transition from below the specified value to above.</td>
</tr>
<tr>
<td>TrigBelow</td>
<td>The trigger occurs when the analog signal makes a transition from above the specified value to below.</td>
</tr>
<tr>
<td>GateNegHys</td>
<td>The trigger occurs when the analog signal is more than the specified high value. The acquisition stops if the analog signal is less than the specified low value.</td>
</tr>
<tr>
<td>GatePosHys</td>
<td>The trigger occurs when the analog signal is less than the specified low value. The acquisition stops if the analog signal is more than the specified high value.</td>
</tr>
<tr>
<td>GateAbove</td>
<td>The trigger occurs as long as the analog signal is more than the specified value.</td>
</tr>
</tbody>
</table>
GateBelow  The trigger occurs as long as the analog signal is less than the specified value.
GateInWindow The trigger occurs as long as the analog signal is within the specified range of values.
GateOutWindow The trigger occurs as long as the analog signal is outside the specified range of values.

National Instruments
The following trigger conditions are available for AI objects when TriggerType is HwDigital. (For AO objects, TriggerCondition is fixed at NegativeEdge.)

PositiveEdge The trigger occurs when the positive (rising) edge of a digital signal is detected.
{NegativeEdge} The trigger occurs when the negative (falling) edge of a digital signal is detected.

The following trigger conditions are available when TriggerType is HwAnalogChannel or HwAnalogPin.

{AboveHighLevel} The trigger occurs when the analog signal is above the specified value.
BelowLowLevel The trigger occurs when the analog signal is below the specified value.
InsideRegion The trigger occurs when the analog signal is inside the specified region.
LowHysteresis The trigger occurs when the analog signal is less than the specified low value with hysteresis given by the specified high value.
HighHysteresis The trigger occurs when the analog signal is greater than the specified high value with hysteresis given by the specified low value.
TriggerCondition

See Also
Properties
TriggerChannel, TriggerConditionValue, TriggerType
### Purpose
Specify voltage value(s) that must be satisfied before trigger executes

### Description
TriggerConditionValue is used when TriggerType is Software, and is ignored when TriggerCondition is None.

To execute a software trigger, the values specified for TriggerCondition and TriggerConditionValue must be satisfied. When TriggerCondition is Rising or Falling, TriggerConditionValue accepts a single value. When TriggerCondition is Entering or Leaving, TriggerConditionValue accepts a two-element vector of values.

### Characteristics
<table>
<thead>
<tr>
<th>Usage</th>
<th>AI, Common</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>Double (or a two-element vector of doubles)</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Values
The default value is zero.

### Examples
Create the analog input object ai and add one channel to it.

```matlab
ai = analoginput('winsound');
ch = addchannel(ai,1);
```

The trigger executes when a signal with a negative slope passing through 0.2 volts is detected on channel 1.

```matlab
set(ai,'TriggerChannel',ch)
set(ai,'TriggerType','Software')
set(ai,'TriggerCondition','Falling')
set(ai,'TriggerConditionValue',0.2)
```
TriggerConditionValue

See Also

Properties

TriggerCondition, TriggerType
**Purpose**

Specify delay value for data logging

**Description**

You can define both pretriggers and postriggers. Pretriggers are specified with a negative TriggerDelay value while postriggers are specified with a positive TriggerDelay value. You can delay a trigger in units of time or samples with the TriggerDelayUnits property. Pretriggers are not defined for hardware triggers or when TriggerType is Immediate.

Pretrigger samples are included as part of the total samples acquired per trigger as specified by the SamplesPerTrigger property. If sample-time pairs are returned to the workspace with the getdata function, then the pretrigger samples are identified with negative time values.

**Characteristics**

- **Usage**: AI, Common
- **Access**: Read/write
- **Data type**: Double
- **Read-only when running**: Yes

**Values**

The default value is zero.

**Examples**

Create the analog input object ai and add one channel to it.

```matlab
ai = analoginput('winsound');
ch = addchannel(ai,1);
```

Configure ai to acquire 44,100 samples per trigger with 11,025 samples (0.25 seconds) acquired as pretrigger data.

```matlab
set(ai,'SampleRate',44100)
set(ai,'TriggerType','Manual')
set(ai,'SamplesPerTrigger',44100)
set(ai,'TriggerDelay',-0.25)
```
See Also

Properties

SamplesPerTrigger, TriggerDelayUnits
**TriggerDelayUnits**

**Purpose**
Specify units in which trigger delay data is measured

**Description**
TriggerDelayUnits can be Seconds or Samples. If TriggerDelayUnits is Seconds, then data logging is delayed by the specified time for each channel group member. If TriggerDelayUnits is Samples, then data logging is delayed by the specified number of samples for each channel group member.

The trigger delay value is given by the TriggerDelay property.

**Characteristics**

<table>
<thead>
<tr>
<th>Usage</th>
<th>AI, Common</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>String</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Values**

- **{Seconds}**
  The trigger is delayed by the specified number of seconds.

- **Samples**
  The trigger is delayed by the specified number of samples.

**See Also**

**Properties**

- TriggerDelay
Purpose

Specify M-file callback function to execute when trigger occurs

Description

A trigger event is generated immediately after a trigger occurs. This event executes the callback function specified for `TriggerFcn`. Under most circumstances, the callback function is not guaranteed to complete execution until sometime after `Logging` is set to `On` for analog input (AI) objects, or `Sending` is set to `On` for analog output (AO) objects.

Trigger event information is stored in the `Type` and `Data` fields of the `EventLog` property. The `Type` field value is `Trigger`. The `Data` field values are given below.

<table>
<thead>
<tr>
<th>Data Field Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AbsTime</code></td>
<td>The absolute time (as a clock vector) the event occurred.</td>
</tr>
<tr>
<td><code>RelSample</code></td>
<td>The acquired (AI) or output (AO) sample number when the event occurred.</td>
</tr>
<tr>
<td><code>Channel</code></td>
<td>The index number for each input channel serving as a trigger source (AI only).</td>
</tr>
<tr>
<td><code>Trigger</code></td>
<td>The trigger number.</td>
</tr>
</tbody>
</table>

Characteristics

- **Usage**: AI, AO, Common
- **Access**: Read/write
- **Data type**: String
- **Read-only when running**: No

Values

The default value is an empty string.
See Also

Functions
trigger

Properties
EventLog, Logging
**Purpose**
Specify number of additional times trigger executes

**Description**
You can configure a trigger to occur once (one-shot acquisition) or multiple times. If `TriggerRepeat` is set to its default value of zero, then the trigger executes once. If `TriggerRepeat` is set to a positive integer value, then the trigger executes once, and is repeated the specified number of times. For example, if the value is set to 2, you will get a total of 3 triggers. If `TriggerRepeat` is set to `inf` then the trigger executes continuously until a stop function is issued or an error occurs.

You can quickly evaluate how many triggers have executed by examining the `TriggersExecuted` property or by invoking the display summary for the device object. The display summary is invoked by typing the device object name at the MATLAB command line.

**Characteristics**
- **Usage**: AI, Common
- **Access**: Read/write
- **Data type**: Double
- **Read-only when running**: Yes

**Values**
The default value is zero.

**See Also**
- **Functions**: `disp`, `stop`

**Properties**
- `TriggersExecuted`, `TriggerType`
**Purpose**
Indicate number of triggers that execute

**Description**
You can find out how many triggers executed by returning the value of TriggersExecuted. The trigger number for each trigger executed is also recorded by the Data.Trigger field of the EventLog property.

**Characteristics**

- **Usage**: AI, AO, Common
- **Access**: Read-only
- **Data type**: Double
- **Read-only when running**: N/A

**Values**
The default value is zero.

**Examples**
Create the analog input object `ai` and add one channel to it.

```matlab
ai = analoginput('winsound');
ch = addchannel(ai,1);
```

Configure `ai` to acquire 40,000 samples with five triggers using the default sampling rate of 8000 Hz.

```matlab
set(ai,'TriggerRepeat',4)
start(ai)
```

TriggersExecuted returns the number of triggers executed.

```matlab
ai.TriggersExecuted
ans =
    5
```

**See Also**
Properties

- EventLog
**TriggerType**

**Purpose**
Specify type of trigger to execute

**Description**
TriggerType can be Immediate, Manual, or Software. If TriggerType is Immediate, the trigger occurs immediately after the start function is issued. If TriggerType is Manual, the trigger occurs immediately after the trigger function is issued. If TriggerType is Software, the trigger occurs when the associated trigger condition is satisfied (AI only).

For a given hardware device, additional trigger types might be available. Some trigger types require trigger conditions and trigger condition values. Trigger conditions are specified with the TriggerCondition property, while trigger condition values are specified with the TriggerConditionValue property.

When a trigger occurs for an analog input object, data logging is initiated and the Logging property is automatically set to On. When a trigger occurs for an analog output object, data sending is initiated and the Sending property is automatically set to On.

**Characteristics**

<table>
<thead>
<tr>
<th>Usage</th>
<th>AI, AO, Common</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>String</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### Values

<table>
<thead>
<tr>
<th>All Supported Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Immediate}</td>
</tr>
<tr>
<td>Manual</td>
</tr>
<tr>
<td>Software</td>
</tr>
</tbody>
</table>

### Agilent Technologies

| HwDigital              | The trigger source is an external digital signal (AI only). Pretrigger data cannot be captured. |
| HwAnalog               | The trigger source is an external analog signal (AI only). |
| HwDigitalPos           | The trigger source is the positive edge of an external digital signal (AO only). |
| HwDigitalNeg           | The trigger source is the negative edge of an external digital signal (AO only). |

### Keithley

| HwDigital              | The trigger source is an external digital signal (AI only). Pretrigger data cannot be captured. |

---

**Note** Keithley and VXI Technology adaptors will be deprecated in a future version of the Toolbox. If you create a Data Acquisition Toolbox object for the 'keithley' or 'hp1432' adaptors in version R2007a, you will receive a warning stating that these adaptors will be removed in a future release.
TriggerType

Measurement Computing

HwDigital
The trigger source is an external digital signal (AI only). Pretrigger data cannot be captured.

HwAnalog
The trigger source is an external analog signal (AI only).

National Instruments

HwDigital
The trigger source is an external digital signal. Pretrigger data cannot be captured.

HwAnalogChannel
The trigger source is an external analog signal (AI only).

HwAnalogPin
The trigger source is a low-range external analog signal (AI only). Note that HwAnalogPin is supported only for traditional NIDAQ devices. It is not supported for NIDAQmx devices.

For 1200 Series hardware, HwDigital is the only device-specific TriggerType value for analog input subsystems. Analog output subsystems do not support any device-specific TriggerType values.

See Also

Functions
start, trigger

Properties
Logging, Sending, TriggerChannel, TriggerCondition, TriggerConditionValue
**Type**

**Purpose**
Indicate device object type, channel, or line

**Description**
Type is associated with device objects, channels, and lines. For device objects, Type can be Analog Input, Analog Output, or Digital I/O. Once a device object is created, the value of Type is automatically defined.

For channels, the only value of Type is Channel. For lines, the only value of Type is Line. The value is automatically defined when channels or lines are added to the device object.

**Characteristics**

<table>
<thead>
<tr>
<th>Usage</th>
<th>AI, AO, Common, Channel; DIO, Common, Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Read-only</td>
</tr>
<tr>
<td>Data type</td>
<td>String</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Values**

**Device Objects**
For device objects, Type has these possible values:

- **Analog Input**: The device object type is analog input.
- **Analog Output**: The device object type is analog output.
- **Digital I/O**: The device object type is digital I/O.

The value is automatically defined after the device object is created.

**Channels and Lines**
For channels, the only value of Type is Channel. For lines, the only value of Type is Line. The value is automatically defined when channels or lines are added to the device object.
Units

**Purpose**
Specify engineering units label

**Description**
Units is a string that specifies the engineering units label to associate with your data. You should use Units in conjunction with the UnitsRange property.

**Characteristics**
- **Usage**: AI, AO, Channel
- **Access**: Read/write
- **Data type**: String
- **Read-only when running**: No

**Values**
The default value is Volts.

**See Also**
Properties
- UnitsRange
**Purpose**

Specify range of data as engineering units.

**Description**

You use `UnitsRange` to scale your data to reflect particular engineering units.

For analog input objects, the data is scaled while it is extracted from the engine with the `getdata` function according to the formula

\[
\text{scaled value} = \frac{(A/D \text{ value})(\text{units range})}{(\text{sensor range})}
\]

The A/D value is constrained by the `InputRange` property, which reflects the gain and polarity of your analog input channels. The sensor range is given by the `SensorRange` property, which reflects the range of data you expect from your sensor.

For analog output objects, the data is scaled when it is queued in the engine with the `putdata` function according to the formula

\[
\text{scaled value} = \frac{(\text{original value})(\text{output range})}{(\text{units range})}
\]

The output range is constrained by the `OutputRange` property, which specifies the gain and polarity of your analog output channels.

For both objects, you can also use the `Units` property to associate a meaningful label with your data.

**Characteristics**

- **Usage**: AI, AO, Channel
- **Access**: Read/write
- **Data type**: Two-element vector of doubles
- **Read-only when running**: No

**Values**

The default value is determined by the default value of the `InputRange` or the `OutputRange` property.
See Also

Functions

gedata, putdata

Properties

InputRange, OutputRange, SensorRange, Units
**Purpose**

Store data to associate with device object

**Description**

`UserData` stores data that you want to associate with the device object. Note that if you return analog input object information to the MATLAB workspace using the `daqread` function, the `UserData` value is not restored.

**Characteristics**

- **Usage**: AI, AO, DIO, Common
- **Access**: Read/write
- **Data type**: Any type
- **Read-only when running**: No

**Values**

The default value is an empty vector.

**Examples**

Create the analog input object `ai` and add two channels to it.

```matlab
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:1);
```

Suppose you want to access filter coefficients during the acquisition. You can create a structure to store these coefficients, which can then be stored in `UserData`.

```matlab
coeff.a = 1.0;
coeff.b = -1.25;
set(ai,'UserData',coeff)
```
Device-Specific Properties — By Vendor

Device-specific properties apply only to hardware devices of a specific type or from a specific vendor. For example, the BitsPerSample property is supported only for sound cards, while the NumMuxBoards property is supported only for National Instruments devices.

The properties are grouped according to these supported vendors:

- **Advantech (p. 15-2)**
  Device-specific Advantech properties for analog input (AI) and analog output (AO) objects

- **VXI Technology (p. 15-2)**
  Device-specific VXI Technology (formerly Agilent Technologies) properties for analog input (AI) and analog output (AO) objects

- **Keithley (p. 15-3)**
  Device-specific Keithley properties for analog input (AI) and analog output (AO) objects

- **Measurement Computing (p. 15-4)**
  Device-specific Measurement Computing properties for analog input (AI) and analog output (AO) objects

- **National Instruments (p. 15-5)**
  Device-specific National Instruments properties for analog input (AI) and analog output (AO) objects
Parallel Port (p. 15-5)           Device-specific parallel port properties
Sound Card (p. 15-6)            Device-specific sound card properties for analog input (AI) and analog output (AO) objects

You can display device-specific properties with the set function. The device-specific properties are displayed after the base properties.

**Note** Some device-specific property values are not available for all devices. Refer to your hardware documentation for detailed information about device-specific behavior.

### Advantech

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Device Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransferMode</td>
<td>Specify how data is transferred from data acquisition device to system memory</td>
<td>AI, AO</td>
</tr>
</tbody>
</table>

### VXI Technology

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Device Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLA</td>
<td>Specify whether source constant-level output is enabled or disabled</td>
<td>AO</td>
</tr>
<tr>
<td>Coupling</td>
<td>Specify input coupling mode</td>
<td>AI</td>
</tr>
<tr>
<td>GroundingMode</td>
<td>Specify input channel grounding mode</td>
<td>AI</td>
</tr>
<tr>
<td>Property Name</td>
<td>Description</td>
<td>Device Objects</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>InputMode</td>
<td>Specify channel input mode</td>
<td>AI</td>
</tr>
<tr>
<td>InputSource</td>
<td>Specify input to A/D converter</td>
<td>AI</td>
</tr>
<tr>
<td>RampRate</td>
<td>Specify source ramp-up and ramp-down rate</td>
<td>AO</td>
</tr>
<tr>
<td>SourceMode</td>
<td>Specify source mode</td>
<td>AO</td>
</tr>
<tr>
<td>SourceOutput</td>
<td>Specify source output</td>
<td>AO</td>
</tr>
<tr>
<td>Span</td>
<td>Specify measurement bandwidth in Hz</td>
<td>AI, AO</td>
</tr>
<tr>
<td>Sum</td>
<td>Specify whether source sum input is enabled or disabled</td>
<td>AO</td>
</tr>
</tbody>
</table>

**Note**  Keithley and VXI Technology adaptors will be deprecated in a future version of the toolbox. If you create a Data Acquisition Toolbox object for the ’keithley’ or ’hp1432’ adaptors in R2007a, you will receive a warning stating that these adaptors will be removed in a future release.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Device Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutOfDataMode</td>
<td>Specify how value held by analog output subsystem is determined</td>
<td>AO</td>
</tr>
<tr>
<td>StopTriggerChannel</td>
<td>Specify analog input channel serving as a hardware stop trigger source</td>
<td>AI</td>
</tr>
</tbody>
</table>
### Device-Specific Properties — By Vendor

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Device Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>StopTriggerCondition</td>
<td>Specify condition that must be satisfied before stop trigger executes</td>
<td>AI, AO</td>
</tr>
<tr>
<td>StopTriggerConditionValue</td>
<td>Specify value for stop trigger condition</td>
<td>AI</td>
</tr>
<tr>
<td>StopTriggerDelay</td>
<td>Specify delay value for stop trigger</td>
<td>AI</td>
</tr>
<tr>
<td>StopTriggerDelayUnits</td>
<td>Specify units in which stop trigger delay is measured</td>
<td>AI</td>
</tr>
<tr>
<td>StopTriggerType</td>
<td>Specify type of stop trigger to execute</td>
<td>AI</td>
</tr>
<tr>
<td>TransferMode</td>
<td>Specify how data is transferred from data acquisition device to system memory</td>
<td>AI, AO</td>
</tr>
</tbody>
</table>

### Measurement Computing

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Device Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutOfDataMode</td>
<td>Specify how value held by analog output subsystem is determined</td>
<td>AO</td>
</tr>
<tr>
<td>TransferMode</td>
<td>Specify how data is transferred from data acquisition device to system memory</td>
<td>AI, AO</td>
</tr>
</tbody>
</table>
### National Instruments

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Device Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coupling</td>
<td>Specify input coupling mode</td>
<td>AI</td>
</tr>
<tr>
<td><strong>ExternalSampleClockSource</strong></td>
<td>Specify which signal provides clock for sample conversions across channels</td>
<td>AI</td>
</tr>
<tr>
<td><strong>ExternalScanClockSource</strong></td>
<td>Specify which signal starts series of conversions across channels</td>
<td>AI</td>
</tr>
<tr>
<td><strong>HwDigitalTriggerSource</strong></td>
<td>Specify which signal initiates data acquisition</td>
<td>AI, AO</td>
</tr>
<tr>
<td>NumMuxBoards</td>
<td>Specify number of external multiplexer devices connected</td>
<td>AI</td>
</tr>
<tr>
<td><strong>OutOfDataMode</strong></td>
<td>Specify how value held by analog output subsystem is determined</td>
<td>AO</td>
</tr>
<tr>
<td>TransferMode</td>
<td>Specify how data is transferred from data acquisition device to system memory</td>
<td>AI, AO</td>
</tr>
</tbody>
</table>

### Parallel Port

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Device Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BiDirectionalBit</strong></td>
<td>Specify BIOS control register bit that determines bidirectional operation</td>
<td>DIO</td>
</tr>
<tr>
<td><strong>PortAddress</strong></td>
<td>Indicate base address of parallel port</td>
<td>DIO</td>
</tr>
</tbody>
</table>
Sound Card

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>Device Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>BitsPerSample</td>
<td>Specify number of bits sound card uses to represent each sample</td>
<td>AI, AO</td>
</tr>
<tr>
<td>StandardSampleRates</td>
<td>Specify whether valid sample rates snap to small set of standard values, or if you can set sample rate to any value within allowed bounds</td>
<td>AI, AO</td>
</tr>
</tbody>
</table>

Getting Command-Line Property Help

To get command-line property help, you should use the `daqhelp` function. For example, to get help for the sound card’s `BitsPerSample` property:

```
daqhelp BitsPerSample
```

**Note** You can use `daqhelp` without creating a device object.

You can also get property characteristics, such as the default property value, using the `propinfo` function. For example, suppose you create the analog input object `ai` for a sound card and want to find the default value for the `BitsPerSample` property.

```
ai = analoginput('winsound');
out = propinfo(ai,'BitsPerSample');
out.DefaultValue
ans =
16
```
Device-Specific Properties
— Alphabetical List
### Purpose
Specify BIOS control register bit that determines bidirectional operation.

### Description
BiDirectionalBit can be 5, 6, or 7. The default value is 5 because most parallel port hardware uses bit 5 of the BIOS control register to determine the direction (input or output) of port 0.

If port 0 is unable to input data, you need to configure the BiDirectionalBit value to 6 or 7. Typically, you will not know the bit value required by your port, and some experimentation is required.

### Characteristics
<table>
<thead>
<tr>
<th></th>
<th>Vendor</th>
<th>Parallel port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>Usage</td>
<td>DIO, Common</td>
</tr>
<tr>
<td>Access</td>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>Data type</td>
<td>Double</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>Read-only when running</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Values
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>The BIOS control register bit that determines bidirectional operation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>{5}, 6, or 7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### BitsPerSample

**Purpose**
Specify number of bits sound card uses to represent samples.

**Description**
BitsPerSample can be 8, 16, or any value between 17 and 32. The specified number of bits determines the number of unique values a sample can take on. For example, if BitsPerSample is 8, the sound card represents each sample with 8 bits. This means that each sample is represented by a number between 0 and 255. If BitsPerSample is 16, the sound card represents each sample with 16 bits. This means that each sample is represented by a number between 0 and 65,535.

For older Sound Blaster cards configured for full duplex operation, you might not be able to set BitsPerSample to 16 bits for both the analog input and analog output subsystems. Instead, you need to set one subsystem for 8 bits, and the other subsystem for 16 bits.

**Note**
To use the high-resolution (greater than 16 bit) capabilities for some sound cards, you might need to configure BitsPerSample to either 24 or 32 even if your device does not use that number of bits.

### Characteristics

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Sound cards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>AI, AO, Common</td>
</tr>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>Double</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Values

8, {16}, or 17-32
Represent data with the specified number of bits.
**Purpose**

Specify whether source constant-level output is enabled or disabled

**Description**

COLA can be Off or On. If COLA is Off, the source constant level output is disabled. If COLA is On, the source constant level output is enabled.

For the Option 1D4 single-channel source, the source COLA output is shared with the source sum input. Only one of these two sources can be enabled at any one time. For prototype Option 1D4 sources only, one of the two must be enabled at all times, and the default is for the constant-level output to be enabled.

**Characteristics**

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vendor</td>
<td>VXI Technology</td>
</tr>
<tr>
<td>Usage</td>
<td>AO, Channel</td>
</tr>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>String</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Values**

- **Off**
  - The source constant level output is disabled.
- **On**
  - The source constant level output is enabled.
**Purpose**
Specify input coupling mode

**Description**
The Coupling property is visible only if the device you are using supports coupling and the value can be changed. Coupling can be DC or AC. If Coupling is DC, the input is connected directly to the amplifier. If Coupling is AC, a series capacitor is inserted between the input connector and the amplifier. For source channels do not use Coupling because you cannot ac-couple the output of a source.

**Characteristics**

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vendor</td>
<td>VXI Technology, National Instruments</td>
</tr>
<tr>
<td>Usage</td>
<td>AI, Channel</td>
</tr>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>String</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{AC}</td>
<td>A series capacitor is inserted between the input connector and the amplifier.</td>
</tr>
<tr>
<td>DC</td>
<td>The input is connected directly to the amplifier.</td>
</tr>
</tbody>
</table>

The default is set to AC for

- National Instruments devices that use the NI-DAQmx interface and support ac-coupling
- National Instruments DSA cards using the Traditional NI-DAQ interface

In all other cards, the default is set to DC.

**Examples**
Create the analog input object ai for a National Instruments board, and add a hardware channel to it.
ai = analoginput('nidaq','Dev1');
addchannel(ai,0);

You can return the coupling modes supported by the board with the Coupling field of the daqhwinfo function.

out = daqhwinfo(ai);
out.Coupling
ans =
    'AC,DC'

Configure the channel contained by ai to use dc-coupling:

ai.Channel.Coupling = 'DC';
ai.Channel.Coupling
ans= DC
**ExternalSampleClockSource**

**Purpose**
Specify which signal provides clock for sample conversions across channels.

**Description**
`ExternalSampleClockSource` specifies the pin whose signal is used as the channel clock for conversions on each channel. This property is in effect when the `ClockSource` property is set to `ExternalSampleCtrl` or `ExternalSampleAndScanCtrl`.

Data acquisition cards with simultaneous sample and hold ignore this property.

**Note**
The toolbox cannot configure the data acquisition device to output its sample clock to the RTSI bus.

**Characteristics**
- **Vendor**: National Instruments
- **Usage**: AI
- **Access**: Read/write
- **Data type**: String
- **Read-only when running**: Yes

**Values**
- PFI0 to PFI9
- RTSIO to RTSI6

**See Also**
- **Properties**
  - ClockSource, ExternalScanClockSource

**Values**

Use specified pin from PFI0 through PFI9.

Use specified pin from RTSIO through RTSI6.
**Purpose**
Specify which signal starts series of conversions across channels

**Description**
ExternalScanClockSource specifies the pin whose signal is used as the scan clock to initiate conversions across a group of channels. This property is in effect when the ClockSource property is set to ExternalScanCtrl or ExternalSampleAndScanCtrl.

**Note** The toolbox cannot configure the data acquisition device to output its scan clock to the RTSI bus.

**Characteristics**
- **Vendor**: National Instruments
- **Usage**: AI
- **Access**: Read/write
- **Data type**: String
- **Read-only when running**: Yes

**Values**
- **PFI0 to PFI9**: Use specified pin from PFI0 through PFI9.
- **RTSI0 to RTSI6**: Use specified pin from RTSI0 through RTSI6.

**See Also**
- **Properties**
  - ClockSource, ExternalSampleClockSource
Purpose
Specify input channel grounding mode

Description
GroundingMode can be Grounded or Floating. If GroundingMode is Grounded, the low side of the channel is grounded. If GroundingMode is Floating, the low side of the channel floats thereby making the input a differential input. GroundingMode can be set only for input channels. Source channels are never floating, and are always grounded.

If a smart break-out box is attached to the channel, then the grounding mode is automatically set to the appropriate value. If a dumb break-out box (or no break-out box) is attached to the channel, the grounding mode is given by the GroundingMode value. In this case, no hardware settings are changed and no errors are generated.

Characteristics
Vendor VXI Technology
Usage AI, Channel
Access Read/write
Data type String
Read-only when running Yes

Values
{Grounded} The input channel is grounded.
Floating The input channel is floating.
HwDigitalTriggerSource

**Purpose**
Specify which signal initiates data acquisition

**Description**
HwDigitalTriggerSource defines which pin is used to initiate a data acquisition when the TriggerType property is set to HwDigital.

**Characteristics**
- **Vendor**: National Instruments
- **Usage**: AI, AO
- **Access**: Read/write
- **Data type**: String
- **Read-only when running**: Yes

**Values**
- **PFI0 to PFI9**: Use specified pin from PFI0 through PFI9.
- **RTSI0 to RTSI6**: Use specified pin from RTSI0 through RTSI6.

**See Also**
- **Properties**
  - TriggerType

16-10
**Purpose**
Specify channel input mode

**Description**
InputMode can be set to Voltage, ICP, Charge, Mic, or 200VoltMic. You can set InputMode to Charge only if a charge break-out box is attached to the specified channel. You can set InputMode to Mic or 200VoltMic only if a microphone break-out box is attached to the specified channel. For each input mode, the full-scale setting is configured with the InputRange property.

There is no ICP current source inside the E1432 device. Instead, you can attach a break-out box containing an ICP current source to the input. When this ICP break-out box is attached, setting InputMode to ICP enables the ICP current source in the break-out box. If there is no ICP break-out box attached to the input, then setting InputMode to ICP does nothing.

**Note** If a channel is not connected to a smart break-out box, then changing its input mode causes the input mode for all channels within the device to change. If there is a smart break-out box present, then you can set the input mode on a per-channel basis.

**Characteristics**

<table>
<thead>
<tr>
<th>Vendor</th>
<th>VXI Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>AI, Channel</td>
</tr>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>String</td>
</tr>
<tr>
<td>Read-only when</td>
<td>Yes</td>
</tr>
<tr>
<td>running</td>
<td></td>
</tr>
</tbody>
</table>
# InputMode

## Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Voltage}</td>
<td>The input mode is set to volts.</td>
</tr>
<tr>
<td>ICP</td>
<td>The input mode is set to ICP.</td>
</tr>
<tr>
<td>Charge</td>
<td>The input mode is set to charge-amp.</td>
</tr>
<tr>
<td>Mic</td>
<td>The input mode is set to microphone.</td>
</tr>
<tr>
<td>200VoltMic</td>
<td>The input mode is set to microphone with 200 volt supply turned on.</td>
</tr>
</tbody>
</table>

## See Also

**Properties**

Coupling, InputRange, InputType
**Purpose**
Specify input to A/D converter

**Description**
For input channels, InputSource can be SwitchBox, CALIN, Ground, and BOBCALIN. The BOBCALIN value is valid only when a smart break-out box (BoB) is connected to the input. Smart BoB’s include a charge break-out box or a microphone break-out box. When a smart break-out box is attached to an E1432 or E1433 module, additional input modes are available. These additional input modes are available through the InputMode property.

After a hardware reset, InputSource is automatically set to SwitchBox.

**Characteristics**

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vendor</td>
<td>VXI Technology</td>
</tr>
<tr>
<td>Usage</td>
<td>AI, Channel</td>
</tr>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>String</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Values**

- {SwitchBox}
  - Select the front pane connector.
- CALIN
  - Select the module’s CALIN line.
- Ground
  - Ground the input.
- BOBCALIN
  - Select the module’s CALIN line via the CAL connection in a break-out box.

**See Also**

**Properties**

- InputMode
NumMuxBoards

Purpose
Specify number of external multiplexer devices connected

Description
NumMuxBoards specifies the number of AMUX-64T multiplexer devices connected to your hardware. NumMuxBoards can be 0, 1, 2, or 4. If you are using a 1200 Series board, then NumMuxBoards can only be 0.

Characteristics

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Vendor</th>
<th>National Instruments Traditional NI-DAQ devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>AI, Common</td>
<td></td>
</tr>
<tr>
<td>Access</td>
<td>Read/write</td>
<td></td>
</tr>
<tr>
<td>Data type</td>
<td>Double</td>
<td></td>
</tr>
<tr>
<td>Read-only when running</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

Values

<table>
<thead>
<tr>
<th>Values</th>
<th>The number of AMUX-64T multiplexer devices connected.</th>
</tr>
</thead>
<tbody>
<tr>
<td>{0}, 1, 2, or 4</td>
<td></td>
</tr>
</tbody>
</table>
**Purpose**
Specify how value held by analog output subsystem is determined

**Description**
When queued data is output to the analog output (AO) subsystem, the hardware typically holds a value. For National Instruments and Measurement Computing devices, the value held is determined by `OutOfDataMode`.

`OutOfDataMode` can be `Hold` or `DefaultValue`. If `OutOfDataMode` is `Hold`, then the last value output is held by the AO subsystem. If `OutOfDataMode` is `DefaultValue`, then the value specified by the `DefaultChannelValue` property is held by the AO subsystem.

**Characteristics**
- **Vendor**: Keithley, Measurement Computing, National Instruments
- **Usage**: AO, Common
- **Access**: Read/write
- **Data type**: String
- **Read-only when running**: Yes

**Values**
- **Hold**
  - Hold the last output value.
- **DefaultValue**
  - Hold the value specified by `DefaultChannelValue`. 
**OutOfDataMode**

**Examples**

Create the analog output object `ao` and add two channels to it.

```matlab
ao = analogoutput('nidaq','Dev1');
addchannel(ao,0:1);
```

You can configure `ao` so that when queued data is finished being output, a value of 1 volt is held for both channels.

```matlab
ao.OutOfDataMode = 'DefaultValue';
ao.Channel.DefaultChannelValue = 1.0;
```

**See Also**

**Properties**

DefaultChannelValue
**Purpose**
Indicate base address of parallel port

**Description**
The PC supports up to three parallel ports that are assigned the labels LPT1, LPT2, and LPT3. You can use any of these standard ports as long as they use the usual base addresses, which are (in hex) 378, 278, and 3BC, respectively.

Additional ports, or standard ports not assigned the usual base addresses, are not accessible by the toolbox. Note that most PCs that support MATLAB will include a single parallel printer port with base address 378 (LPT1).

**Characteristics**
- **Vendor**: Parallel port
- **Usage**: DIO, Common
- **Access**: Read only
- **Data type**: String
- **Read-only when running**: Yes

**Values**
The value is automatically defined when the object is created.

**Examples**
Create a digital I/O object for parallel port LPT1 and return the PortAddress value.

```matlab
dio = digitalio('parallel','LPT1');
get(dio,'PortAddress')
```

```matlab
ans =
    0x378
```

The returned value indicates that LPT1 uses the usual base address.
**RampRate**

**Purpose** Specify source ramp-up and ramp-down rate

**Description** For input channels, RampRate is not generally used. For source channels, RampRate is usually used to ensure that the source signal starts and stops smoothly.

**Characteristics**

<table>
<thead>
<tr>
<th>Vendor</th>
<th>VXI Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>AO, Channel</td>
</tr>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>Double</td>
</tr>
<tr>
<td>Read-only when running</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Values** You can set RampRate to any value between 0 and 100 seconds.
**Purpose**
Specify source mode

**Description**
If `SourceMode` is set to Arbitrary, the host program must provide the data to use for the arbitrary source signal.

Note that there is no Off source mode. To turn a source channel off, you must make it inactive. When the source is inactive, it is normally low impedance to ground. To make the source high impedance, set the `SourceOutput` property to Open.

**Characteristics**
- **Vendor**: VXI Technology
- **Usage**: AO, Channel
- **Access**: Read/write
- **Data type**: String
- **Read-only when running**: Yes

**Values**
\{Arbitrary\}  
An arbitrary source signal.

**See Also**
**Properties**
SourceOutput
SourceOutput

**Purpose**
Specify source output

**Description**
SourceOutput can be Normal, Grounded, Open, CALOUT, or SRC&CALOUT.

If SourceOutput is Normal, the normal source output is used. This output is defined by the source mode and other source parameters.

If SourceOutput is Grounded, the source output connector remains grounded while the source D/A converter is internally connected to the CALOUT line in the module.

If SourceOutput is Open, the source remains open-circuited even when the source is started. The impedance on the output is only about 1 kilohm because the power-fail decay circuit is still connected to the output.

If SourceOutput is CALOUT, the source output is connected to the module’s internal CALOUT line. This allows the module’s CALOUT line to be driven by an external signal applied at the source output connector.

If SourceOutput is SRC&CALOUT, the source output is connected to the module’s internal CALOUT line, and the source D/A converter is also connected to the CALOUT line. This is a combination of the Grounded and CALOUT values, and is useful for multimainframe calibration.

**Characteristics**

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vendor</td>
<td>VXI Technology</td>
</tr>
<tr>
<td>Usage</td>
<td>AO, Channel</td>
</tr>
<tr>
<td>Access</td>
<td>Read/write</td>
</tr>
<tr>
<td>Data type</td>
<td>String</td>
</tr>
<tr>
<td>Read-only when</td>
<td>Yes</td>
</tr>
<tr>
<td>running</td>
<td></td>
</tr>
<tr>
<td>Values</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>{Normal}</td>
<td>Normal source output.</td>
</tr>
<tr>
<td>Grounded</td>
<td>The source output connector remains grounded while the source D/A converter is internally connected to the CALOUT line in the module.</td>
</tr>
<tr>
<td>Open</td>
<td>The source remains open-circuited even when the source is started.</td>
</tr>
<tr>
<td>CALOUT</td>
<td>The source output is connected to the module’s internal CALOUT line.</td>
</tr>
<tr>
<td>SRC&amp;CALOUT</td>
<td>The source output connector remains grounded while the source D/A converter is internally connected to the CALOUT line in the module. The source output is also connected to the module’s internal CALOUT line.</td>
</tr>
</tbody>
</table>
Span

**Purpose**

Specify measurement bandwidth in Hz

**Description**

For an input channel, span specifies the maximum frequency at which valid alias-protected data is received. Frequencies above this value are filtered out. For a source channel, Span specifies the maximum frequency at which the output signal will correctly track the signal that the source is attempting to generate.

The valid values for Span depend on the current clock frequency. You should set the clock frequency before setting Span. Normally, the maximum valid span is the clock frequency divided by 2.56. Valid spans are given by the maximum span divided by powers of two, and the maximum span divided by five and by powers of two. The ratio between the span and the maximum span is called the *decimation factor*.

For the E1432 module, the maximum number of decimate-by-two passes allowed is nine. Therefore, the maximum decimation factor is $5.2^9$, and the minimum valid span is $(\text{clock frequency})/(2.56 \times 5.2^9)$. If the clock frequency is larger than 51.2 kHz, then the module is unable to do a decimation factor of one. In this case, the minimum decimation factor is two and the maximum valid span is $(\text{clock frequency})/5.12$.

For the E1433 module, the maximum number of decimate-by-two passes allowed is 12, so the maximum decimation factor is $5.2^{12}$. Because of limits in the module’s DSP processor, when the clock frequency is set higher than 102,400 Hz, it is unable to do any decimation. In this case, the only valid span is $(\text{clock frequency})/2.56$. If you attempt to use decimation when the clock frequency is above 102,400 Hz, then an error might occur when the measurement starts.

For the Option 1D4 source board, the maximum number of decimate-by-two passes allowed is 16, and the maximum decimation factor is $5.2^{16}$.

The effective sample rate is defined as the rate at which data is received from an input or used by a source, and is normally equal to 2.56 times the span. If the data is oversampled, then the effective sample rate is 5.12 times the span.
If the digital filters in a module have a cutoff that is sharper than 1/2.56, then some of the frequencies above the maximum span might contain valid alias-protected data. This is the case with the E1432 and E1433 modules, which have a top span filter cutoff of (clock frequency)/2.226, which is 23 kHz when the clock frequency is 51.2 kHz, 88.3 kHz when the clock frequency is 196.608 kHz. However, Span ignores the extra bandwidth so that the maximum span is always 1/2.56 times the effective sample rate.

Span applies to an entire E1432 module rather than to one of its channels. After a hardware reset, each module is automatically set to the maximum legal span.

**Characteristics**

- **Vendor**: VXI Technology
- **Usage**: AI, AO, Common
- **Access**: Read/write
- **Data type**: Double
- **Read-only when running**: Yes

**Values**

Normally, the maximum valid span is given by the clock frequency divided by 2.56. Valid spans are given by the maximum span divided by powers of two, and the maximum span divided by five and by powers of two. The value set for Span automatically updates the SampleRate value.

**See Also**

- **Properties**: SampleRate
StandardSampleRates

**Purpose**
Specify whether valid sample rates snap to small set of standard values, or if you can set sample rate to any allowed value.

**Description**
StandardSampleRates can be On or Off. If StandardSampleRates is Off, then it is possible to set the sample rate to any value within the bounds supported by the hardware. For most sound cards, the lower bound is 8.000 kHz, while the upper bound is 44.1 kHz. For newer sound cards, an upper bound of 96.0 kHz might be supported. The specified sample rate is rounded up to the next integer value.

If StandardSampleRates is On, then the available sample rates snap to a small set of standard values. The standard values are 8.000 kHz, 11.025 kHz, 22.050 kHz, and 44.100 kHz. If you specify a sampling rate that is within one percent of a standard value, then the sampling rate snaps to that standard value. If you specify a sampling rate that is not within one percent of a standard value, then the sampling rate rounds up to the closest standard value.

Regardless of the StandardSampleRates value, if you specify a sampling rate that is outside the allowed limits, then an error is returned.

**Characteristics**
- **Vendor**: Sound cards
- **Usage**: AI, AO, Common
- **Access**: Read/write
- **Data type**: String
- **Read-only when running**: Yes
Values

<table>
<thead>
<tr>
<th></th>
<th>On</th>
<th>The sample rate can be set only to a small set of standard values.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{Off}</td>
<td>If supported by the hardware, the sample rate can be set to any value within the allowed bounds, up to a maximum of 96.0 kHz.</td>
</tr>
</tbody>
</table>
**StopTriggerChannel**

**Purpose**
Specify analog input channel serving as hardware stop trigger source

**Description**
StopTriggerChannel defines the channel number to be used for the HwAnalog setting of the StopTriggerType property. The channel must be a member of the analog input channel list.

To associate a particular channel with the stop trigger, assign the channel’s hardware ID number to the property. If you specify a channel object, then an error is returned.

**Characteristics**
- **Vendor**: Keithley
- **Usage**: AI, Common
- **Access**: Read/write
- **Data type**: Double
- **Read-only when running**: Yes

**Values**
Any defined analog input channel. The default value is an empty vector.

**Examples**
Create an analog input object for the Keithley KPCI-3108 board and add eight channels.

```matlab
ai = analoginput('keithley',1);
addchannel(ai,0:7);
```

Stop the acquisition when a falling voltage level of 0.1 volt is detected on the hardware channel with ID 2.

```matlab
ai.StopTriggerType='HwAnalog';
ai.StopTriggerChannel = 2;
ai.StopTriggerCondition = 'Falling';
ai.StopTriggerConditionValue = 0.1;
```
See Also

Properties

StopTriggerCondition, StopTriggerConditionValue, StopTriggerDelay, StopTriggerDelayUnits, StopTriggerType
**StopTriggerCondition**

**Purpose**
Specify condition that must be satisfied before stop trigger executes

**Description**
StopTriggerCondition can be None, Rising, or Falling. As described below, the stop trigger condition depends on the value specified for the StopTriggerType property, which can be HwDigital or HwAnalog.

If StopTriggerCondition is Rising, the trigger executes on the rising edge of TGIN line (HwDigital), or when the analog input signal rises above the value given in StopTriggerConditionValue (HwAnalog). If StopTriggerCondition is Falling, the trigger executes on the falling edge of TGIN line (HwDigital), or when the analog input signal falls below the value given in StopTriggerConditionValue (HwAnalog).

If you use stop triggers in conjunction with start triggers, and both trigger types are HwDigital, then the trigger conditions must be the same for both triggers (for example, both Rising or both Falling).

**Characteristics**

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Vendor</th>
<th>Usage</th>
<th>Access</th>
<th>Data type</th>
<th>Read-only when running</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Keithley</td>
<td>AI, Common</td>
<td>Read/write</td>
<td>String</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Values**

The following stop trigger condition is used when StopTriggerType is None.

{None} No stop trigger condition is required.

The following stop trigger conditions are used when StopTriggerType is HwDigital or HwAnalog.
StopTriggerCondition

{Rising} Trigger on the rising edge of TGIN line, or when the analog input signal rises above the value given in StopTriggerConditionValue.

Falling Trigger on the falling edge of TGIN line, or when the analog input signal falls below the value given in StopTriggerConditionValue.

See Also

Properties
StopTriggerChannel, StopTriggerConditionValue, StopTriggerDelay, StopTriggerType
**StopTriggerConditionValue**

**Purpose**
Specify value for stop trigger condition

**Description**
StopTriggerConditionValue defines the value that must be satisfied before a stop trigger executes. You use this property only when StopTriggerType is set to HwAnalog.

**Characteristics**
- **Vendor**: Keithley
- **Usage**: AI, Common
- **Access**: Read/write
- **Data type**: Double
- **Read-only when running**: Yes

**Values**
The default value is 0.

**See Also**
- **Properties**
  - StopTriggerChannel
  - StopTriggerCondition
  - StopTriggerDelay
  - StopTriggerType
### StopTriggerDelay

**Purpose**  
Specify delay value for stop trigger

**Description**  
StopTriggerDelay allows the acquisition to continue beyond a hardware stop trigger event. The property value is interpreted in StopTriggerDelayUnits, which can be either seconds or samples. StopTriggerDelay must be zero (the default) or a positive number. Negative (pretrigger) delays are not supported.

**Characteristics**
- **Vendor**: Keithley  
- **Usage**: AI, Common  
- **Access**: Read/write  
- **Data type**: Double  
- **Read-only when running**: Yes

**Values**  
The default value is 0. Only positive values are permitted. If StopTriggerDelayUnits is set to Samples, only integer values are allowed.

**See Also**  
**Properties**  
- StopTriggerDelayUnits, StopTriggerType
**StopTriggerDelayUnits**

**Purpose**
Specify units in which stop trigger delay is measured

**Description**
StopTriggerDelayUnits can be Seconds or Samples. If StopTriggerDelayUnits is Seconds, then data logging is delayed by the specified time for each channel group member. If StopTriggerDelayUnits is Samples, then data logging is delayed by the specified number of samples for each channel group member.

The stop trigger delay value is given by the StopTriggerDelay property.

**Characteristics**

- **Vendor**: Keithley
- **Usage**: AO, Channel
- **Access**: Read/write
- **Data type**: String
- **Read-only when running**: Yes

**Values**

- **{Seconds}**: The trigger is delayed by the specified number of seconds.
- **Samples**: The trigger is delayed by the specified number of samples.

**See Also**

**Properties**

StopTriggerDelay
StopTriggerType

**Purpose**
Specify type of stop trigger to execute

**Description**
StopTriggerType can be None, HwDigital, or HwAnalog. If StopTriggerType is HwDigital, the acquisition stops on the rising or falling edge of the TGIN input line as defined by the StopTriggerCondition property. If StopTriggerType is HwAnalog, the acquisition stops when the channel specified by the StopTriggerChannel property meets the conditions defined by StopTriggerCondition and StopTriggerConditionValue.

For both HwDigital and HwAnalog, SamplesPerTrigger is automatically set to Inf. Therefore, your acquisition will run until the stop trigger is received or the stop function is issued. You can continue the acquisition beyond the specified stop trigger by setting StopTriggerDelay to a positive value.

**Characteristics**

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Vendor</th>
<th>Usage</th>
<th>Access</th>
<th>Data type</th>
<th>Read-only when running</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vendor</td>
<td>Keithley</td>
<td>AI, Common</td>
<td>Read/write</td>
<td>String</td>
<td>Yes</td>
</tr>
<tr>
<td>Usage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Access</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data type</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read-only when</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>running</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


StopTriggerType

Values

{None}  The acquisition stops when the number of samples specified by SamplesPerTrigger is acquired, or the stop function is issued.

HwDigital  The acquisition stops on the rising or falling edge of the TGIN input line.

HwAnalog  The acquisition stops when the channel specified by the StopTriggerChannel property meets the specified stop trigger conditions.

See Also

Properties

SamplesPerTrigger, StopTriggerChannel, StopTriggerCondition, StopTriggerConditionValue, StopTriggerDelay
Sum

Purpose Specify whether source sum input is enabled or disabled

Description Sum can be Off or On. If Sum is Off, the sum input is disabled. If Sum is On, the sum input is enabled. The signal on the sum input is internally added to the output that the source would otherwise produce.

For the Option 1D4 single-channel source, the source sum input is shared with the source COLA output. Only one of these two sources can be enabled at any one time. For prototype Option 1D4 sources, one of the two must be enabled at all times. By default, the constant-level output is enabled and the sum input is disabled.

Characteristics

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Vendor</th>
<th>Usage</th>
<th>Access</th>
<th>Data type</th>
<th>Read-only when running</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VXI Technology</td>
<td>AO, Channel</td>
<td>Read/write</td>
<td>String</td>
<td>No</td>
</tr>
</tbody>
</table>

Values

<table>
<thead>
<tr>
<th>{Off}</th>
<th>Disable the source sum input.</th>
</tr>
</thead>
<tbody>
<tr>
<td>On</td>
<td>Enable the source sum input.</td>
</tr>
</tbody>
</table>
TransferMode

**Purpose**
Specify how data is transferred from data acquisition device to system memory

**Description**
For National Instruments NI-DAQmx hardware, this property is ignored. The device driver automatically selects the most efficient transfer mode available.

For National Instruments Traditional NI-DAQ hardware, TransferMode can be Interrupts or SingleDMA for both analog input and analog output subsystems. If TransferMode is Interrupts, then data is transferred from the hardware first-in, first-out memory buffer (FIFO) to system memory using interrupts. If TransferMode is SingleDMA, then data is transferred from the hardware FIFO to system memory using a single direct memory access (DMA) channel. Some boards also support a TransferMode of DualDMA for analog input subsystems. For example, the AT-MIO-16E-1 board supports this transfer mode. If TransferMode is DualDMA, then data is transferred from the hardware FIFO to system memory using two DMA channels. Depending on your system resources, data transfer via interrupts can significantly degrade system performance.

For Measurement Computing hardware, TransferMode can be Default, InterruptPerPoint, DMA, InterruptPerBlock, or InterruptPerScan. If TransferMode is Default, the transfer mode is automatically selected by the driver based on the board type and the sampling rate. If TransferMode is InterruptPerPoint, a single conversion is transferred for each interrupt. You should use this property value if your sampling rate is less than 5 kHz or you specify a small block size for memory buffering (as defined by the BufferingConfig property). If TransferMode is DMA, data is transferred using a single DMA channel. If TransferMode is InterruptPerBlock, a block of data is transferred for each interrupt. You should use this property value if your sampling rate is greater than 5 kHz and you are using a board that has a fast maximum sampling rate. Note that a data block is defined by the board, and usually corresponds to half the FIFO size. If TransferMode is InterruptPerScan, data is not transferred until the entire scan is complete. This can only be used when the number of points acquired is less than or equal to the FIFO size. You should use this mode if your
sampling rate is higher than the maximum continuous scan rate of the data acquisition device.

For Keithley hardware, TransferMode can be Interrupts or DMA. If TransferMode is Interrupts, then data is transferred from the hardware first-in, first-out memory buffer (FIFO) to system memory using interrupts. If TransferMode is DMA, then data is transferred from the hardware FIFO buffer to system memory using a single DMA channel. Note that if bus mastering is disabled in the DriverLINX Configuration panel for the device, then DMA is not offered as an option.

**Note** If your sampling rate is greater than ~5 kHz, you should avoid using interrupts if possible. The recommended TransferMode setting for your application will be described in your hardware documentation, and depends on the specific board you are using and your platform configuration.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vendor</strong></td>
<td>Keithley, Measurement Computing, National Instruments</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>AI, AO, Common</td>
</tr>
<tr>
<td><strong>Access</strong></td>
<td>Read/write</td>
</tr>
<tr>
<td><strong>Data type</strong></td>
<td>String</td>
</tr>
<tr>
<td><strong>Read-only when running</strong></td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Advantech**

- `{InterruptPerPoint}` Transfer single data points using interrupts.
- `InterruptPerBlock` Transfer a block of data using interrupts (AI only).
TransferMode

Keithley

DMA  Transfer data using a single DMA channel.
Interrupts  Transfer data using interrupts.

If bus mastering is disabled in the DriverLINX configuration panel for the device, then DMA is not available, and the default is set to Interrupts.

Note  Keithley and VXI Technology adaptors will be deprecated in a future version of the Toolbox. If you create a Data Acquisition Toolbox object for the 'keithley' or 'hp1432' adaptors in version R2007a, you will receive a warning stating that these adaptors will be removed in a future release.

Measurement Computing

{Default}  The transfer mode is automatically selected by the driver based on the board type and the sampling rate.
InterruptPerPoint  Transfer single data points using interrupts.
DMA  Transfer data using a single DMA channel (AI only).
InterruptPerBlock  Transfer a block of data using interrupts (AI only).
InterruptPerScan  Transfer all data when the acquisition is complete (AI only).
**National Instruments**

<table>
<thead>
<tr>
<th>TransferMode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inteupts</td>
<td>Transfer data using interrupts.</td>
</tr>
<tr>
<td>SingleDMA</td>
<td>Transfer data using a single DMA channel.</td>
</tr>
<tr>
<td>DualDMA</td>
<td>Transfer data using two DMA channels.</td>
</tr>
</tbody>
</table>

This default property value is supplied by the driver. For most devices that support data transfer via interrupts and DMA, SingleDMA is the default value.

**Examples**

Set the TransferMode property for a National Instruments board before acquiring data.

```matlab
iai = analoginput('nidaq', 'Dev1');
set(ai, 'TransferMode', 'SingleDMA');
addchannel(ai, 1:2);
softscope(ai)
```
TransferMode
Blocks — Alphabetical List

Data Acquisition Toolbox provides a Data Acquisition Block Library that enables you to interact with data acquisition hardware directly in Simulink. You can use these blocks to acquire analog or digital data in a Simulink model, or to output analog or digital data from the model to a hardware device.
Analog Input

**Purpose**
Acquire data from multiple analog channels of data acquisition device

**Library**
Data Acquisition Toolbox

**Description**
The Analog Input block opens, initializes, configures, and controls an analog data acquisition device. The opening, initialization, and configuration of the device occur once at the start of the model’s execution. During the model’s run time, the block acquires data either synchronously (deliver the current block of data the device is providing) or asynchronously (buffer incoming data).

**Note** You must have a license for both Data Acquisition Toolbox and Simulink to use this block.

The block has no input ports. It has one or more output ports, depending on the configuration you choose in the Source Block Parameters dialog box. The following diagram shows the block configured with one port for both channels and with one port for each channel, in the case of a device that has two channels.

Use the Analog Input block to incorporate live measured data into Simulink for:

- System characterization
- Algorithm verification
- System and algorithm modeling
- Model and design validation
- Controls design

You can use the Analog Input block for signal applications by using it in conjunction with basic Simulink and Signal Processing blocks.

You can use the Analog Input block either synchronously or asynchronously. Select the acquisition mode in the Source Block Parameters dialog box.

The following diagram shows the basic analog input usage scenario, in which you would:

- Acquire data at each time step or once per model execution.
- Analyze the data, or use it as input to a system in the model.
- Optionally display results.

For an example of creating a model using the Analog Input block, see Example: Bringing Analog Data into a Model.

**Other Supported Features**

The Analog Input block supports the use of Simulink Accelerator mode. This feature speeds up the execution of Simulink models.
The Analog Input block supports the use of model referencing. This feature lets your model include other Simulink models as modular components.

For more information on these features, see the Simulink documentation.
Use the Source Block Parameters dialog box to select your acquisition mode and to set other configuration options.
**Acquisition Mode**

**Asynchronous**
Initiates the acquisition when the simulation starts. The simulation runs while data is acquired into a FIFO (First in, First out) buffer. The acquisition is continuous; the block buffers data while outputting a scan/frame of data at each time step.

**Synchronous**
Initiates the acquisition at each time step. The simulation will not continue until the requested block of data is acquired. This is unbuffered input; the block will synchronously output the latest scan/frame of data at each time step.

The following diagrams show the difference between synchronous and asynchronous modes for the Analog Input block.

**Synchronous Analog Input**

At the first time step (T1), the acquisition is initiated for the required block of data (B1). The simulation does not continue until B1 is completely acquired.
Asynchronous Analog Input – Scenario 1

Scenario 1 shows the case when simulation speed outpaces data acquisition speed. At the first time step (T1), the required block of data (B1) is still being acquired. Therefore, the simulation does not continue until B1 is completely acquired.
Asynchronous Analog Input – Scenario 2

Scenario 2 shows the case when data acquisition speed outpaces simulation speed. At the first time step (T1), the required block of data (B1) has been completely acquired. Therefore the simulation runs continuously.

**Note** Several factors, including device hardware and model complexity, can affect the simulation speed, causing both scenarios 1 and 2 to occur within the same simulation.

**Options**

**Device**

The data acquisition device from which you want to acquire data. The items in the list vary, depending on which devices you have connected to your system. Devices in the list are specified by adaptor/vendor name and unique device ID, followed by the name of the device. The first available device is selected by default.
**Hardware sample rate**

The rate at which samples are acquired from the device, in samples per second. This is the sampling time for the hardware. The default is defined when a device is selected.

The sample rate must be a positive real number, and be within the range allowed for the selected hardware.

**Block size**

The desired number of data samples to output at each time step for each channel. Block size corresponds to the `SamplesPerTrigger` property for an analog input device. The default value for block size depends on the hardware selected. It must be a positive integer, and be within the range allowed for the selected hardware.

**Input type**

Specifies the hardware channel configuration, such as single-ended, differential, etc. The input type is defined by the capabilities of the selected device.

**Channels**

The channel configuration table lists your device’s hardware channels and lets you configure them. Use the check boxes and selection buttons to specify which channels to acquire data from.

The **Hardware Channel** column displays the hardware channel ID specified by the device. The **Hardware Channel** column is read-only.

The **Name** column specifies the channel name. By default the table will display any names provided by the hardware, but you can edit the names. For example, if the device is a sound card with two channels, you can name them **Left** and **Right**.

**Input Range** specifies the input ranges available for each channel supported by the hardware, and is defined when a device is selected.
Outputs

Number of Ports
Select 1 for all hardware channels (default) or 1 per hardware channel.

Using 1 for all hardware channels means that data will be output from a single port as a matrix, with a size of [Block size x Number of Channels selected].

Using 1 per hardware channel means that data will be output from N ports, where N is equal to the number of selected channels. Each output port will be a column vector with a size of [Block size x 1]. For naming, each output port will use the channel name if one was specified, or otherwise use [“HWChannel” + channel ID], for example, HWChannel12.

Signal type
Select Sample-based or Frame-based. This option determines whether the signal type is sample-based or frame-based. Sample-based is the default.

Note The Frame-based option will work only if you have the Signal Processing blockset installed.

Data type
Select your data type to output from the block. The Analog Input block supports double and native data types, as supported by the hardware. double is the default. Native data types will be dynamically populated in this list based on the hardware that is selected. For example, if int16 is a native data type of a specific hardware device, then one of the entries for Data type will be int16 (native).
Purpose
Output data to multiple analog channels of data acquisition device

Library
Data Acquisition Toolbox

Description
The Analog Output block opens, initializes, configures, and controls an analog data acquisition device. The opening, initialization, and configuration of the device occur once at the start of the model’s execution. During the model’s run-time, the block outputs data to the hardware either synchronously (outputs the block of data as it is provided) or asynchronously (buffers output data).

Note
You must have a license for both Data Acquisition Toolbox and Simulink to use this block.

The block has one or more input ports, depending on the option you choose in the Sink Block Parameters dialog box. It has no output ports. The following diagram shows the block configured with one port for both channels and with one port for each channel, in the case of a device that has two channels selected.

The Analog Output block inherits the sample time from the driving block connected to the input port. The valid data types of the signal at the input port are double or native data types supported by the hardware.
Other Supported Features

The Analog Output block supports the use of Simulink Accelerator mode. This feature speeds up the execution of Simulink models.

The Analog Output block supports the use of model referencing. This feature lets your model include other Simulink models as modular components.

For more information on these features, see the Simulink documentation.
Use the Sink Block Parameters dialog box to select your acquisition mode and to set other configuration options.
Output Mode

Asynchronous
Initiates data output to the hardware when simulation starts. The simulation runs while data is output from a FIFO (First in, First out) buffer. This mode buffers and outputs data from the block, letting you perform a frame-based or sample-based output.

Synchronous
Initiates data output to the hardware at each time step. The simulation will not continue running until the current block of data is output. In synchronous mode, the block synchronously outputs a vector or frame of samples provided at each time step.

The following diagrams show the difference between synchronous and asynchronous analog output.

Synchronous Analog Output

At the first time step (T1), data output is initiated and the corresponding block of data (B1) is output to the hardware. The simulation does not continue until B1 is output completely.
Asynchronous Analog Output – Scenario 1

Scenario 1 shows the case when data output speed outpaces simulation speed. At the first time step (T1), data output is initiated and the corresponding block of data (B1) is output to the hardware. The simulation runs continuously in this mode.
Asynchronous Analog Output – Scenario 2

Scenario 2 shows the case when simulation speed outpaces data acquisition speed. At the first time step (T1), data output is initiated and the corresponding block of data (B1) is output to the hardware. Data is queued at successive time steps and is output to the hardware once the previous block completes. The simulation runs continuously in this mode.

Note Several factors, including device hardware and model complexity, can affect the simulation speed, causing both scenarios 1 and 2 to occur within the same simulation.

Options

Device
The data acquisition device to which you want to output data. The items in the list vary, depending on which devices you have connected to your system. Devices in the list are specified by adaptor/vendor name and unique device ID, followed by the name of the device. The first available device is selected by default.
**Hardware output rate**

The rate at which samples are output to the device, in samples per second. This output rate for the hardware is defined when a device is selected. The output rate specified must be within the range supported by the selected device.

**Channels**

The channel configuration table lists your device’s hardware channels and lets you configure them. Use the check boxes and selection buttons to specify which channels to send data to.

The **Hardware Channel** column displays the channel ID specified by the device, and is read-only.

The **Name** column specifies the channel name. By default the table will display any names provided by the hardware, but you can edit the names. For example, if the device is a sound card with two channels, you can name them Left and Right.

**Output Range** specifies the output ranges available for each channel supported by the hardware, and is defined by the selected device.

The final column specifies the **Initial Value** to be output at the start of the simulation, if you are using Asynchronous mode. The default value is 0. In Synchronous mode, the **Initial Value** column does not appear in the table.

**Note** For AC-coupled devices like a sound card, this column is not used and is read-only.

**Number of ports**

Select **1 for all hardware channels** (default) or **1 per hardware channel**.
Analog Output

Using **1 for all hardware channels** means that data will be input from a single port as a matrix, with a size of \([S \times \text{Number of Channels selected}]\), where \(S\) is number of samples provided as input.

Using **1 per hardware channel** means that data will be input from \(N\) ports, where \(N\) is equal to the number of selected channels. Each input port will be a column vector with a size of \([S \times 1]\), where \(S\) is the number of samples provided as an input. For naming, each output port will use the channel name if one was specified, or otherwise use \[“HWChannel” + \text{channel ID}\], for example, HWChannel12.
### Purpose
Acquire latest set of values from multiple digital lines of data acquisition device

### Library
Data Acquisition Toolbox

### Description
The Digital Input block synchronously outputs the latest scan of data available from the digital lines selected at each simulation time step. It acquires unbuffered digital data, and the data delivered is a binary vector.

**Note** You must have a license for both Data Acquisition Toolbox and Simulink to use this block.

The block has no input ports. It has one or more output ports, depending on the option you choose in the Source Block Parameters dialog box. The following diagram shows the block configured with one port for all lines and with one port for each line, in the case of a device that has 17 lines selected.
The block inherits the sample time of the model.

The output data is always a binary vector (binvec), i.e., a vector of logical values.
Digital input acquisition is done synchronously. The following diagram shows synchronous digital input.

At the first time step (T1), data is acquired from the selected hardware lines. The simulation does not continue until data is read from all lines.

**Other Supported Features**

The Digital Input block supports the use of Simulink Accelerator mode. This feature speeds up the execution of Simulink models.

The Digital Input block supports the use of model referencing. This feature lets your model include other Simulink models as modular components.

For more information on these features, see the Simulink documentation.
Use the Source Block Parameters dialog box to set configuration options.

**Device**

The data acquisition device from which you want to acquire data. The items in the list vary, depending on which devices you have connected to your system. Devices in the list are specified by adaptor/vendor name and unique device ID, followed by the name of the device. The first available device is selected by default.

**Lines**

The line configuration table lists your device’s lines and lets you configure them. The table lists all the lines that can be configured for input. Use the check boxes and selection buttons to specify which lines to acquire data from.
**Hardware Port ID**
Specifies the ID for each hardware port. This is automatically detected and filled in by the selected device, and is read-only.

**Hardware Line ID**
Specifies the ID of the hardware line. This is automatically detected and filled in by the selected device, and is read-only.

**Name**
Specifies the hardware line name. This is automatically detected and filled in from the hardware, but you can edit the name.

**Number of ports**
Select **1 for all hardware lines** (default) or **1 per hardware line**.

Using **1 for all hardware lines** means that the block will have only one output port for all of the lines that are selected in the table. Data must be \([S \times \text{number of lines}]\), where \(S\) is the number of samples. Data will be a binary vector (binvec).

Using **1 per hardware line** means the block will have one output port per selected line. The name of each output port is the name specified in the table for each line. If no name is provided, the name is “Port” + HwPort ID + “Line” + Line ID. For example, if line 2 of hardware port 3 is selected, and you did not specify a name in the line table, Port3Line2 appears in the block. Data will be \([1 \times 1]\).
**Digital Output**

**Purpose**
Output data to multiple digital lines of data acquisition device

**Library**
Data Acquisition Toolbox

**Description**
The Digital Output block synchronously outputs the latest set of data to the hardware at each simulation time step. It outputs unbuffered digital data. The output data is always a binary vector (binvec).

**Note**
You must have a license for both Data Acquisition Toolbox and Simulink to use this block.

The block has no output ports. It can have one or more input ports, depending on the option you choose in the Sink Block Parameters dialog box. The following diagram shows the block configured with one port for all lines and with one port for each line, in the case of a device that has 12 lines.
The Digital Output block inherits the sample time from the driving block connected to the input port. The data type of the signal at the input port must be a logical data type.

Digital output is done synchronously. The following diagram shows synchronous digital output.

At the first time step (T1), data is output to the selected hardware lines. The simulation does not continue until data is output to all lines.

**Other Supported Features**

The Digital Output block supports the use of Simulink Accelerator mode. This feature speeds up the execution of Simulink models.

The Digital Output block supports the use of model referencing. This feature lets your model include other Simulink models as modular components.

For more information on these features, see the Simulink documentation.
Use the Sink Block Parameters dialog box to set configuration options.

**Device**

The data acquisition device to which you want to output data. The items in the list vary, depending on which devices you have connected to your system. Devices in the list are specified by adaptor/vendor name and unique device ID, followed by the name of the device. The first available device is selected by default.
Digital Output

**Lines**

The line configuration table lists your device’s lines and lets you configure them. Use the check boxes and selection buttons to specify which lines to send data to.

**Hardware Port ID**

Specifies the ID for each hardware port. This is automatically detected and filled in by the selected device, and is read-only.

**Hardware Line ID**

Specifies the ID of the hardware line. This is automatically detected and filled in by the selected device, and is read-only.

**Name**

Specifies the hardware line name. This is automatically detected and filled in by the selected device, but you can edit the name.

**Number of ports**

Select **1 for all hardware lines** (default) or **1 per hardware line**.

Using **1 for all hardware lines** means that the block will have only one input port for all lines selected in the table. Data needs to be \([S \times \text{number of lines}]\), where \(S\) is the number of samples. Data at the input port needs to be a binary vector (binvec).

Using **1 per hardware line** means the block will have one input port per selected line. The name of each input port is the name specified in the table for each line. If no name is provided, the name is “Port” + HwPort ID + Line + Line ID. For example, if line 2 of port 3 is selected, and you did not specify a name in the line table, `Port3Line2` appears in the block. Data needs to be \([1 \times 1]\).
Troubleshooting Your Hardware

This appendix describes simple tests you can perform to troubleshoot your data acquisition hardware. The tests involve using software provided by the vendor or the operating system (sound cards), and do not involve using Data Acquisition Toolbox. The sections are as follows.

- Advantech Hardware (p. A-3) How to use the Advantech Device Manager
- Agilent Technologies Hardware (p. A-5) How to use the Soft Front Panel
- Measurement Computing Hardware (p. A-7) How to use InstaCal
- National Instruments Hardware (p. A-9) How to use the Measurement & Automation Explorer
- Sound Cards (p. A-12) How to use Windows resources
- Other Things to Try (p. A-20) How to register the hardware driver adaptor or contact The MathWorks

To accurately test your hardware, you should use these vendor tools to match the requirements of your data acquisition session. For example, you should select the appropriate sampling rate, number of channels, acquisition mode (continuous or single-point), and input range. If these tests do not help you, then you might need to register the hardware driver adaptor or contact The MathWorks for support. Contact information is provided in “Contacting The MathWorks” on page A-21 as well as in the beginning of this guide. If the problem is with your hardware, then you should contact the hardware vendor.
Note that if you cannot access your board using the vendor’s software, then you will not be able to do so with Data Acquisition Toolbox.

**Note**  To see the full list of hardware that the toolbox supports, visit the Data Acquisition Toolbox product page at the MathWorks Web site [www.mathworks.com/products/daq](http://www.mathworks.com/products/daq).
Advantech Hardware

In this section...

“What Driver Are You Using?” on page A-3
“Is Your Hardware Functioning Properly?” on page A-3

What Driver Are You Using?
Data Acquisition Toolbox is compatible only with specific versions of Advantech drivers and is not guaranteed to work with any other versions. For a list of the Advantech driver versions that are compatible with Data Acquisition Toolbox, refer to the product page on the MathWorks Web site at http://www.mathworks.com/products/daq/.

If you think your driver is incompatible with Data Acquisition Toolbox, you should verify that your hardware is functioning properly before updating drivers. If your hardware is functioning properly, then you are probably using unsupported drivers. For the latest drivers, visit the Advantech Web site at http://www.advantech.com/.

With the Advantech Device Manager, you can find out which version of Advantech drivers you are using. You should be able to access this program through the Windows desktop.

To see if a specific version of a driver is installed on your system, select the type of device in the Supported Devices list, and click About.

Is Your Hardware Functioning Properly?
To troubleshoot your Advantech hardware, you use the Advantech Device Test dialog box. This dialog allows you to test each subsystem supported by your board, and is installed as part of the Advantech Device Manager. To access the Advantech Device Test dialog box from the Advantech Device Manager, select the appropriate device in the Installed Devices list, and click Test.

For example, suppose you want to verify that the analog input subsystem on your PCI-1710 board is operating correctly. To do this, connect a known
signal, such as that produced by a function generator, to one or more channels using a screw terminal panel.

If the Advantech Device Test dialog box does not provide you with the expected results for the subsystem under test, and you are sure that your test setup is configured correctly, then the problem is probably in the hardware.

To get support for your Advantech hardware, visit their Web site at http://www.advantech.com/.
What Driver Are You Using?

Data Acquisition Toolbox is compatible only with specific versions of the HP E1432 driver and is not guaranteed to work with any other versions. You can find out which driver version you are using with the Soft Front Panel, which is described in the next section.

If you think your driver is incompatible with Data Acquisition Toolbox, then you should verify that your hardware is functioning properly before updating drivers. If your hardware is functioning properly, then you are probably using unsupported drivers. Visit the Agilent Web site at http://agilent.com/ for the latest drivers.

For a list of the HP E1432 driver versions that are compatible with Data Acquisition Toolbox, refer to the product page on the MathWorks Web site at http://www.mathworks.com/products/daq/.

Is Your Hardware Functioning Properly?

To troubleshoot your Agilent hardware, you should use the HP E1432 Soft Front Panel. The Soft Front Panel allows you to test each module supported by the HP E1432 driver software, and is installed as part of this software. You can access the Soft Front Panel through the Windows Start button.

Select Start > Programs > hpe1432 > HP E1432 Soft Front Panel.

For example, suppose you want to verify that the HP E1432 module is operating correctly. To do this, you should connect a known signal – such as that produced by a function generator — to the module. You then configure the input parameters.
If the Soft Front Panel does not provide you with the expected results for the module under test, and you are sure that your test setup is configured correctly, then the problem is probably with the hardware.

To get support for your Agilent Technologies hardware, visit their Web site at http://www.agilent.com/.
Measurement Computing Hardware

What Driver Are You Using?
Data Acquisition Toolbox is compatible only with specific versions of the Universal Library drivers or the associated release of the InstaCal software, and is not guaranteed to work with any other versions. For a list of the driver versions that are compatible with Data Acquisition Toolbox, refer to the product page on the MathWorks Web site at http://www.mathworks.com/products/daq/.

If you think your driver is incompatible with Data Acquisition Toolbox, then you should verify that your hardware is functioning properly before updating drivers. If your hardware is functioning properly, then you are probably using unsupported drivers. Visit the Measurement Computing Web site at http://www.measurementcomputing.com/ for the latest drivers.

Suppose you are using InstaCal software with your hardware. You can access this software through the Windows Start button.

Select Start > Programs > Measurement > Computing > InstaCal.

The driver version is available through the Help menu.

Select Help > About > InstaCal.

Is Your Hardware Functioning Properly?
To troubleshoot your Measurement Computing hardware, you should use the test feature provided by InstaCal. To access this feature, select the board you want to test from the PC Board List, and select Analog from the Test menu.

For example, suppose you want to verify that the analog input subsystem on your PCI-DAS4020/12 board is operating correctly. To do this, you should
connect a known signal — such as that produced by a function generator — to one of the channels, using a BNC cable.

If InstaCal does not provide you with the expected results for the subsystem under test, and you are sure that your test setup is configured correctly, then the problem is probably with the hardware.

To get support for your Measurement Computing hardware, visit their Web site at http://www.measurementcomputing.com/.
National Instruments Hardware

In this section...

“NI-DAQmx Versus Traditional NI-DAQ Drivers” on page A-9
“What Driver Are You Using?” on page A-10
“Is Your Hardware Functioning Properly?” on page A-10

NI-DAQmx Versus Traditional NI-DAQ Drivers

National Instruments provides two drivers for accessing their hardware. Data Acquisition Toolbox supports both:

- Traditional NI-DAQ
- NI-DAQmx

Some older National Instruments devices require the Traditional NI-DAQ driver. Many of the newest National Instruments devices, such as the M-series, require the NI-DAQmx driver. However, many of the more popular National Instruments devices, such as E-series and S-series, can be used with either driver. To find out which driver your hardware requires, see the NI-DAQmx release notes at the National Instruments Web site, http://www.ni.com.

If your hardware can use either driver, you should choose the NI-DAQmx interface. If you have a mix of hardware that cannot all use the same driver, you can install both drivers to access your hardware. Any device that supports both drivers will appear twice in the results of daqhwinfo('nidaq'); you should access these devices from the Traditional NI-DAQ interface.

```matlab
daqhwinfo('nidaq')
AdaptorDllName: [1x63 char]
AdaptorDllVersion: '2.10 (R2007a)'
AdaptorName: 'nidaq'
BoardNames: {'PCI-4472' 'PCI-4472'}
InstalledBoardIds: {'Dev4' '1'}
ObjectConstructorName: {2x3 cell}
```
Notice that the 'PCI-4472' board appears in the list twice. This device is available through both the new NI-DAQmx interface and the Traditional NI-DAQ interface.

Devices accessed by NI-DAQmx use a string device ID such as 'Dev1' to identify the board. Traditional NI-DAQ devices use a numeric device ID. For example:

```plaintext
ai_mx = analoginput('nidaq','Dev4')
ai_trad = analoginput('nidaq','1')
```

**What Driver Are You Using?**

Data Acquisition Toolbox is compatible only with specific versions of the NI-DAQ driver and is not guaranteed to work with any other versions. For a list of the NI-DAQ driver versions that are compatible with Data Acquisition Toolbox, refer to the product page on the MathWorks Web site at [http://www.mathworks.com/products/daq/](http://www.mathworks.com/products/daq/).

If you think your driver is incompatible with Data Acquisition Toolbox, then you should verify that your hardware is functioning properly before updating drivers. If your hardware is functioning properly, then you are probably using unsupported drivers. Visit the National Instruments Web site at [http://www.ni.com/](http://www.ni.com/) for the latest NI-DAQ drivers.

You can find out which version of NI-DAQ you are using with National Instruments’ **Measurement & Automation Explorer**. You should be able to access this program through the Windows Desktop. The driver version is available through the **Help** menu.

Select **Help > System Information** to open the **Measurement & Automation Explorer**.

**Is Your Hardware Functioning Properly?**

To troubleshoot your National Instruments hardware, you should use the **Test Panel**. The **Test Panel** allows you to test each subsystem supported by your board, and is installed as part of the NI-DAQ driver software. You can access the **Test Panel** by right-clicking the appropriate device in the Measurement & Automation Explorer and choosing **Test Panel**.
For example, suppose you want to verify that the analog input subsystem on your PCI-6024E board is operating correctly. To do this, you should connect a known signal — such as that produced by a function generator — to one or more channels, using a screw terminal panel.

If the Test Panel does not provide you with the expected results for the subsystem under test, and you are sure that your test setup is configured correctly, then the problem is probably with the hardware.

To get support for your National Instruments hardware, visit their Web site at http://www.ni.com/.
Sound Cards

<table>
<thead>
<tr>
<th>In this section...</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Verify if your Sound Card is Functioning” on page A-12</td>
</tr>
<tr>
<td>“Microphone and Sound Card Types” on page A-16</td>
</tr>
<tr>
<td>“Testing with a Microphone” on page A-17</td>
</tr>
<tr>
<td>“Testing with a CD Player” on page A-17</td>
</tr>
<tr>
<td>“Running in Full-Duplex Mode” on page A-18</td>
</tr>
</tbody>
</table>

Verify if your Sound Card is Functioning

You can verify that your sound card is functioning properly by recording data and then playing back the recorded data. Recording data uses the sound card’s analog input subsystem, while playing back data uses the sound card’s analog output subsystem. Successful completion of these two tasks indicates your sound card works properly. The data to be recorded can come from two sources:

- A microphone
- A CD player

The first thing you should do is enable your sound card’s ability to record and play data. This is done using the Microsoft Windows Sounds and Audio Devices Properties dialog box. You can access this dialog box using the Windows Start button.

Select Start > Settings > Control Panel, then double-click Sounds and Audio Devices.
The Sounds and Audio Devices Properties dialog box is shown below, and is configured for both playback and recording.

You can record data and then play it back using the Windows **Sound Recorder** panel. To access this application, select the following:

**Start > Programs > Accessories > Entertainment > Sound Recorder**

The figure below shows how to record and play data.
You must also make sure that your microphone or CD player is enabled for recording and playback using the Windows **Volume Control** panel. To access this application:

**Start > Programs > Accessories > Entertainment > Volume Control**

The **Volume Control** panel is shown below. The CD, microphone, and line devices are enabled for playback when the **Mute** check box is cleared for the **CD, Microphone**, and **Line** volume controls, respectively. You can play .WAV files by leaving the **Mute** check box cleared for the **Wave** volume control.

If the CD, microphone, or Wave Output controls do not appear in the **Volume Control** panel, you must modify the playback properties by selecting **Properties** from the **Options** menu.
The Properties dialog box is shown below for playback devices. Select the appropriate device check box to enable playback.

![Properties dialog box for playback devices](image)

To verify if the CD and microphone are enabled for recording, click the **Recording** option in the Properties dialog box, and then select the appropriate device check box to enable recording. The Properties dialog box is shown below for recording devices.

![Properties dialog box for recording devices](image)
The **Recording Control** panel is shown below. You enable the CD or microphone for recording when the **Select** check box is selected for the **CD** or **Microphone** controls, respectively.

![Recording Control Panel](image)

**Microphone and Sound Card Types**

Your microphone will be one of two possible types: powered or unpowered. You can use powered microphones only with Sound Blaster or Sound Blaster-compatible microphone inputs. You can use unpowered microphones with any sound card microphone input. Some laptops must use unpowered microphones because they do not have Sound Blaster compatible sound cards.

As shown below, you can easily identify these two microphone types by their jacks.

- ![Unpowered Microphone Jack](image)
  - *Unpowered microphone jack*

- ![Powered Microphone Jack](image)
  - *Powered microphone jack*

You can find out which sound card brand you have installed by clicking the **Devices** tab on the Sounds and Audio Devices Properties dialog box. Refer to “Sound Cards” on page A-12 for a picture of this dialog box.
Testing with a Microphone
To test your sound card with a microphone, follow these steps:

1. Plug the microphone into the appropriate sound card jack. For a Sound Blaster sound card, this jack is labeled MIC IN.

2. Record audio data by selecting the Record button on the Sound Recorder and then speak into the microphone. While recording, the green line in the Sound Recorder should indicate that data is being captured. If this is the case, then the analog input subsystem on your sound card is functioning properly.

3. After recording the audio data, save it to disk. The data is automatically saved as a .WAV file.

4. Play the saved .WAV file. While playing, the green line in the Sound Recorder should indicate that data is being captured. If this is the case, then the analog output subsystem on your sound card is functioning properly.

If you are not able to record or play data, make sure that the sound card and input devices are enabled for recording and playback as described in the beginning of this section.

Testing with a CD Player
To test your sound card with a CD player, follow these steps:

1. Check that your CD is physically connected to your sound card.
   - Open your computer and locate the back of the CD player.
   - If there is a wire connecting the Audio Out CD port with the sound card, you can record audio data from your CD. If there is no wire connecting your CD and sound card, you must either make this connection or use the microphone to record data.

2. Put an audio CD into your CD player. A Windows CD player application should automatically start and begin playing the CD.
3 While the CD is playing, record audio data by clicking the Record button on the Sound Recorder. While recording, the green line in the Sound Recorder should indicate that data is being captured. If this is the case, the analog input subsystem on your sound card is functioning properly. Note that the CD player converts digital audio data to analog audio data. Therefore, the CD sends analog data to the sound card.

4 After recording the audio data, save it to disk. The data is automatically saved as a .WAV file.

5 Play the saved .WAV file. While playing, the green line in the Sound Recorder should indicate that data is being captured. If this is the case, then the analog output subsystem on your sound card is functioning properly.

If you are not able to record or play data, make sure that the sound card and input devices are enabled for recording and playback as described in the beginning of this section.

**Running in Full-Duplex Mode**

The term full duplex refers to a system that can send and receive information simultaneously. For sound cards, full duplex means that the device can acquire input data via an analog input subsystem while outputting data via an analog output subsystem at the same time.

Note that full tells you nothing about the bit resolution or the number of channels used in each direction. Therefore, sound cards can simultaneously receive and send data using 8 or 16 bits while in mono or stereo mode. A common restriction of full-duplex mode is that both subsystems must be configured for the same sampling rate.

If you try to run your card in full duplex mode and the following error is returned,

```
 Error using ==> daqdevice/start
 Device 'Winsound' already in use.
```

then your sound card is not configured properly, it does not support this mode, or you don’t have the correct driver installed.
If your card supports full-duplex mode, then you might need to enable this feature through the Sounds and Audio Devices Properties dialog box. Refer to “Sound Cards” on page A-12 for a picture of this dialog box. If you are unsure about the full-duplex capabilities of your sound card, refer to its specification sheet or user manual. It is usually very easy to update your hardware drivers to the latest version by visiting the vendor’s Web site.
Other Things to Try

<table>
<thead>
<tr>
<th>In this section...</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Registering the Hardware Driver Adaptor” on page A-20</td>
</tr>
<tr>
<td>“Contacting The MathWorks” on page A-21</td>
</tr>
</tbody>
</table>

Registering the Hardware Driver Adaptor
When you first create a device object, the associated hardware driver adaptor is automatically registered so that the data acquisition engine can make use of its services.

The hardware driver adaptors included with the toolbox are all located in the `daq/private` directory. The full name for each adaptor is shown below.

Supported Vendors/Device Types and Full Adaptor Names

<table>
<thead>
<tr>
<th>Vendor/Device Type</th>
<th>Full Adaptor Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advantech</td>
<td>mwadvantech.dll</td>
</tr>
<tr>
<td>Agilent Technologies</td>
<td>mwhpe1432.dll</td>
</tr>
<tr>
<td>Keithley</td>
<td>mwkeithley.dll</td>
</tr>
<tr>
<td>Measurement Computing</td>
<td>mwmcc.dll</td>
</tr>
<tr>
<td>National Instruments</td>
<td>mwnidaq.dll</td>
</tr>
<tr>
<td>Parallel ports</td>
<td>mwparallel.dll</td>
</tr>
<tr>
<td>Windows sound cards</td>
<td>mwwinsound.dll</td>
</tr>
</tbody>
</table>

If for some reason a toolbox adaptor is not automatically registered, then you need to register it manually using the `daqregister` function. For example, to manually register the sound card adaptor:

```
daqregister('winsound');
```

If you are using a third-party adaptor, then you might need to register it manually. If so, you must supply the full pathname to `daqregister`. For example, to register the third-party adaptor `myadaptor.dll`:
daqregister('D:/MATLABR12/toolbox/daq/myadaptors/myadaptor.dll')

**Note** You must install the associated hardware driver before adaptor registration can occur.

**Contacting The MathWorks**

If you need support from The MathWorks, visit our Web site at [http://www.mathworks.com/support/](http://www.mathworks.com/support/).

Before contacting The MathWorks, you should run the `daqsupport` function. This function returns diagnostic information such as

- The versions of the MathWorks products you are using
- Your MATLAB path
- The characteristics of your hardware

The output from `daqsupport` is automatically saved to a text file, which you can use to help troubleshoot your problem. For example, to have MATLAB generate this file for you, type

    daqsupport
# Vendor Limitations

This appendix describes specific limitations of Data Acquisition Toolbox particular to each vendor:

<table>
<thead>
<tr>
<th>Vendor Hardware</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keithley Hardware (p. B-2)</td>
<td>Describes reported limitations with Keithley hardware</td>
</tr>
<tr>
<td>National Instruments Hardware (p. B-5)</td>
<td>Describes reported limitations with NI hardware</td>
</tr>
<tr>
<td>Measurement Computing Hardware (p. B-7)</td>
<td>Describes reported limitations with MCC hardware</td>
</tr>
<tr>
<td>VXI Technology Hardware (p. B-8)</td>
<td>Describes reported limitations with VXI Technology hardware</td>
</tr>
<tr>
<td>Windows Sound Cards (p. B-9)</td>
<td>Describes reported limitations with Windows sound cards</td>
</tr>
</tbody>
</table>
## Keithley Hardware

Known problems associated with Keithley hardware are described below. If there is a known resolution, then it is described as well.

**Note** Keithley and VXI Technology adaptors will be deprecated in a future version of the Toolbox. If you create a Data Acquisition Toolbox object for the 'keithley' or 'hp1432' adaptors in version R2007a, you will receive a warning stating that these adaptors will be removed in a future release.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Boards</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data missed event on output</td>
<td>KCPI-1801/02</td>
<td>For high sample rates and large numbers of samples, these boards may drop samples on output. To avoid this problem, try increasing the default BufferingConfig property value.</td>
</tr>
<tr>
<td></td>
<td>KPCI-3110</td>
<td></td>
</tr>
<tr>
<td>No output at maximum sampling rate</td>
<td>KPCI-3110</td>
<td>When the sampling rate is set above 600 kHz, the output is corrupted.</td>
</tr>
<tr>
<td><strong>Problem</strong></td>
<td><strong>Boards</strong></td>
<td><strong>Comments</strong></td>
</tr>
<tr>
<td>-------------</td>
<td>------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Cannot sample properly below 10 Hz</td>
<td>KPCI-3110</td>
<td>For sample rates at or below approximately 10 Hz, the acquisition takes longer than expected. For example, a 1-second acquisition at 10 Hz takes approximately 3 seconds. This is due to a limitation in the number of blocks that the hardware device can transfer at a time. To avoid this problem, use software clocking.</td>
</tr>
<tr>
<td>Time out on output</td>
<td>Software clocked devices</td>
<td>Some software clocked devices may time out when outputting data. This is due to a limitation in the number of blocks that the hardware devices can transfer at a time.</td>
</tr>
<tr>
<td>Queuing data while running</td>
<td>KPCI-1801/02HC KPCI-3110</td>
<td>If an analog output object is running and more data is queued, then no data is output. To avoid this problem, try increasing the default BufferingConfig property value.</td>
</tr>
<tr>
<td>Problem</td>
<td>Boards</td>
<td>Comments</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Digital triggers are not available.</td>
<td>All</td>
<td>Digital triggers are not available if supported in interrupt mode. This applies to both start and stop triggers. Digital triggers are available only if supported in direct memory access (DMA) mode.</td>
</tr>
<tr>
<td>Limited channel skew values</td>
<td>PCMCIA boards</td>
<td>These boards support only these specific channel skew values: 10 µs, 20 µs, 40 µs.</td>
</tr>
<tr>
<td>SS/H is not supported for analog output.</td>
<td>PCMCIA boards, DDA-08, DDA-16</td>
<td>The analog output subsystems for these boards have simultaneous sample and hold (SS/H) capabilities. However, the toolbox does not support this feature.</td>
</tr>
<tr>
<td>Repeating triggers may result in dropped samples.</td>
<td>All</td>
<td>If the TriggerRepeat property is nonzero, samples may be dropped during acquisition. To avoid this problem, configure the BufferingConfig property to an exact multiple of the SamplesPerTrigger property.</td>
</tr>
<tr>
<td>Analog input subsystem is not available.</td>
<td>KPCMCIA-16AIO</td>
<td>N/A</td>
</tr>
</tbody>
</table>
National Instruments Hardware

- Data Acquisition Toolbox requires Version 7.5 of the NI-DAQmx drivers.
- If you use Data Acquisition Toolbox and National Instruments’ Measurement and Automation (M&A) Explorer at the same time, a conflict will occur and you will not be able to access your board. To avoid a conflict, you should access your board using either the toolbox or the M&A Explorer, and close the other software application.
- If you install NI-DAQ on your computer, and then install LabVIEW 6i on the same computer, you will need to reinstall NI-DAQ.
- When running a device with a Traditional NI-DAQ driver at a sampling rate of 5000 Hz or higher and with a TransferMode property value of Interrupt, system performance might decline.
- You should configure the SampleRate property with the setverify function just before starting the hardware. Note that the SampleRate value depends on the number of channels added to the device object, and the ChannelSkew property value depends on the SampleRate value.
- When using the 1200 series hardware, you must add channels in reverse order. If you specify invalid channels, the data acquisition engine will create the number of requested channels with valid hardware IDs. You can determine the hardware IDs with the object’s display or with the HwChannel property.
- Only one digital I/O (DIO) object should be associated with a given DIO subsystem. To perform separate tasks with the hardware lines, you should add all the necessary lines to the DIO object, but partition them into separate line groups based on the task.
- When using a Traditional NI-DAQ driver, all channels contained within an analog input object must have the same polarity. In other words, the InputRange property for these channels must have all unipolar values or all bipolar values.
- When using mux boards, you must add channels in a specific order using the addmuxchannel function.
- If you have trouble acquiring data with the DAQPad-MIO-16XE-50, you should increase the size of the engine buffer with the BufferingConfig property.
• The ability to use PXI signals in Data Acquisition Toolbox is not available. These modes are supported by NI 6281 PXI boards and by the Ni_DAQmx library, but are not available in the toolbox. In particular, the ability to use the PXI_STAR signal for the HwDigitalTriggerSource property of the AnalogInput object and the PXI_CLK10 backplane clock for the ExternalSampleClock property are unavailable.

• Objects created for National Instruments devices, and used with the NI-DAQmx adaptor have the following behavior when you use the GETSAMPLE, PUTSAMPLE, GETVALUE, and PUTVALUE functions:
  - The first time the command is used with the object, the corresponding subsystem of the device is reserved by the MATLAB session.
  - If you then try to access that subsystem in different session of MATLAB, or any other application from the same computer, you may receive an error message informing you that the subsystem is reserved.
  - You must delete the object in the first session before you can use it in the next one.
Measurement Computing Hardware

- For boards that do not have a channel gain list, an error occurs at start if all the channel input ranges are not the same or the channel scan order is not contiguous. However, if the ClockSource property value is set to software, this rule does not apply.

- You should configure the SampleRate property with the setverify function just before starting the hardware. Note that the SampleRate value is dependent upon the number of channels added to the device object.

- For boards that do not support continuous background transfer mode (i.e., the board does not have hardware clocking), the only available ClockSource property value is software.

- When running at a sampling rate of 5000 Hz or higher and with a TransferMode property value of InterruptPerPoint, there may be a considerable decline in system performance.

- Most boards do not support simultaneous input and output. However, if software clocking is used, then this limitation does not apply.

- To use hardware digital triggers with the PCI-DAS4020/12 board, you must first configure the appropriate trigger mode with InstaCal.

- Expansion boards are not supported. This includes the CIO-EXP family of products.

- MEGA-FIFO hardware is not supported.
VXI Technology Hardware

- When you start an analog input object associated with an E1432A or E1433A board that has the Arbitrary Source Output option, the source is automatically started. Therefore, you should not use a TriggerType property value of Manual with hardware having this option.

- For analog output objects, you should configure the SampleRate and Span properties with the setverify function just before starting the hardware, since these property values depend on the number of channels contained by the analog output object.

- You must add channels in increasing order and a channel array cannot contain repeated channels.

- The first time you connect a device object to a VXI board, a list of available hardware is determined and all the hardware is initialized. However, this list is not updated during a MATLAB session. Therefore, if you install a new board or remove an existing board while MATLAB is running, you will not see the new configuration. To see the new configuration, you must restart MATLAB. When all device objects are deleted from the data acquisition engine, all the hardware is closed.

- For the E1433A, the minimum sampling rate is 20 Hz and the minimum span is 7.8125.

- The first channel in the TriggerChannel property list is used to trigger the object.

- For the E1434A, channels 1 and 2 and channels 3 and 4 share a 56000 DSP. Therefore, certain operational aspects are coupled between the channels in each pair. For example, both channels in each pair will have the same RampRate property value.

- If you create a device object that spans multiple boards, the device object should list the logical addresses using the same order as returned by the daqhwnfo function. To determine the logical address order, use the daqhwnfo function with no input arguments and examine the InstalledBoardIds field.
Windows Sound Cards

- The maximum sampling rate depends on the `StandardSampleRates` property value. If `StandardSampleRates` is `On`, the maximum `SampleRate` property value is 44100. If `StandardSampleRates` is `Off`, the maximum `SampleRate` property value is 96000 if supported by the sound card.

For some sound cards that allow nonstandard sampling rates, certain values above 67,000 Hz will cause your computer to hang.

- If you are acquiring data when `StandardSampleRates` is `Off`, one of these messages may be returned to the command line depending on the specific sound card you are using:
  - "Invalid format for device winsound" occurs when the sound card does not allow for any nonstandard value.
  - "Device Winsound already in use" occurs when a nonstandard sampling rate is specified and the device takes longer than expected to acquire data.
Managing Your Memory Resources

This appendix describes how to manage memory resources. The sections are as follows.

- **Memory Allocation (p. C-2)** How the toolbox automatically allocates memory resources and how you can override this allocation
- **How Much Memory Do You Need? (p. C-4)** How to determine the memory required for your acquisition needs
- **Example: Managing Memory Resources (p. C-5)** An example using a sound card that illustrates how the toolbox allocates memory
Memory Allocation

When data is acquired from an analog input subsystem or output to an analog output subsystem, it must be temporarily stored in computer memory.

Data Acquisition Toolbox allocates memory in terms of data blocks. A data block is defined as the smallest “slice” of memory that the data acquisition engine can usefully manipulate. For example, acquired data is logged to a disk file using an integral number of data blocks. A representation of allocated memory using n data blocks is shown below.

```
block 1    block 2    block 3    ...    block n
```

Data Acquisition Toolbox strives to make memory allocation as simple as possible. For this reason, the data block size and number of blocks are automatically calculated by the engine. This calculation is based on the parameters of your acquisition such as the sampling rate, and is meant to apply to most common data acquisition applications. Additionally, as data is acquired, the number of blocks dynamically increases up to a predetermined limit. However, the engine cannot guarantee that the appropriate block size, number of blocks, or total memory is allocated under these conditions:

- You select certain property values. For example, if the samples to acquire per trigger are significantly less than the FIFO buffer of your hardware.
- You acquire data at the limits of your hardware, your computer, or the toolbox. In particular, if you are acquiring data at very high sampling rates, then the allocated memory must be carefully evaluated to guarantee that samples are not lost.

You are free to override the memory allocation rules used by the engine and manually change the block size and number of blocks, provided the device object is not running. However, you should do so only after careful consideration, as system performance might be adversely affected, which can result in lost data.
You can manage memory resources using the BufferingConfig property and the daqmem function. With BufferingConfig, you can configure and return the block size and number of blocks used by a device object. With daqmem, you can return the current state of the memory resources used by a device object, and configure the maximum memory that one or more device objects can use.
How Much Memory Do You Need?

The memory (in bytes) required for data storage depends on these factors:

- The number of hardware channels you use
- The number of samples you need to store in the engine
- The data type size of each sample

The memory required for data storage is given by the formula

\[
\text{memory required} = \text{samples stored} \times \text{channel number} \times \text{data type}
\]

Of course, the number of samples you need to store in the engine at any time depends on your particular needs. The memory used by a device object is given by the formula

\[
\text{memory used} = \text{block size} \times \text{block number} \times \text{channel number} \times \text{data type}
\]

The block size and block number are given by the BufferingConfig property. The data type is given by the NativeDataType field of the daqhwinfo function.

You can display the memory resources used by (and available to) a device object with the daqmem function. For analog input objects, memory is used when channels are added. For analog output objects, memory is used when data is queued in the engine. For both device objects, the memory used can dynamically change based on the number of samples acquired or queued.
Example: Managing Memory Resources

Suppose you create the analog input object `ai` for a sound card, add two channels to it, and configure a four second acquisition using a sampling rate of 11.025 kHz.

```matlab
ai = analoginput('winsound');
addchannel(ai,1:2);
set(ai,'SampleRate',11025);
set(ai,'SamplesPerTrigger',44100);
```

You return the default block size and number of blocks with the `BufferingConfig` property.

```matlab
get(ai,'BufferingConfig')
ans =
    1024    30
```

You return the memory resources with the `daqmem` function.

```matlab
daqmem(ai)
ans =
    UsedBytes: 122880
    MaxBytes: 18011136
```

The `UsedBytes` field tells you how much memory is currently used by `ai`, while the `MaxBytes` field tells you the maximum memory that `ai` can use to store acquired data. Note that the value returned for `MaxBytes` depends on the total available computer memory, and might be different for your platform.

You can verify the `UsedBytes` value with the formula given in the previous section. However, you must first find the size (in bytes) of each sample using the `daqhwinfo` function.

```matlab
hwinfo = daqhwinfo(ai);
hwinfo.NativeDataType
ans =
    int16
```

The value of the `NativeDataType` field tells you that each sample requires two bytes. Therefore, the initial allocated memory is 122,880 bytes. However,
if you want to keep all the acquired data in memory, then 176,400 bytes are required. Data Acquisition Toolbox will accommodate this memory requirement by dynamically increasing the number of data blocks after you start ai.

    start(ai)

After all the data is acquired, you can examine the final number of data blocks used by ai.

    ai.BufferingConfig
    ans =
          1024  44

The final total memory used is

    daqmem(ai)
    ans =
          UsedBytes:  180224
               MaxBytes:  18011136

Note that this was more than enough memory to store all the acquired data.
**accuracy**
A determination of how close a measurement comes to the true value.

**acquiring data**
The process of inputting an analog signal from a sensor into an analog input subsystem, and then converting the signal into bits that the computer can read.

**actuator**
A device that converts data output from your computer into a physical variable.

**adaptor**
The interface between the data acquisition engine and the hardware driver. The adaptor’s main purpose is to update the engine with properties that are unique to the hardware device.

**A/D converter**
An analog input subsystem.

**analog input subsystem**
Hardware that converts real-world analog input signals into bits that a computer can read. This is also referred to as an AI subsystem, an A/D converter, or an ADC.

**analog output subsystem**
Hardware that converts digital data to a real-world analog signal. This is also referred to as an AO subsystem, a D/A converter, or a DAC.

**bandwidth**
The range of frequencies present in the signal being measured. You can also think of bandwidth as being related to the rate of change of the signal. A slowly varying signal has a low bandwidth, while a rapidly varying signal has a high bandwidth.

**base property**
A property that applies to all supported hardware subsystems of a given type (analog input, analog output, etc.). For example, the SampleRate
property is supported for all analog input subsystems regardless of the vendor.

**callback function**
An M-file function that you construct to suit your specific data acquisition needs. If you supply the callback function as the value for a callback property, then the function is executed when the event associated with the callback property occurs.

**callback property**
A property associated with a specific event type. When an event occurs, the engine examines the associated callback property. If a callback function is given as the value for the callback property, then that function is executed. All event types have a callback property.

**channel**
A component of an analog input subsystem or an analog output subsystem that you read data from, or write data to.

**channel group**
The collection of channels contained by an analog input object or an analog output object. For scanning hardware, the channel group defines the scan order.

**channel property**
A property that applies to individual channels.

**channel skew**
The time gap between consecutively sampled channels. Channel skew exists only for scanning hardware.

**common property**
A property that applies to every channel or line contained by a device object.

**configuration**
The process of supplying the device object with the resources and information necessary to carry out the desired tasks. Configuration consists of two steps: adding channels or lines, and setting property values to establish the desired behavior.
counter/timer subsystem
Hardware that is used for event counting, frequency and period measurement, and pulse train generation. This subsystem is not supported by Data Acquisition Toolbox.

D/A converter
A digital to analog subsystem.

data acquisition session
A process that encompasses all the steps you must take to acquire data using an analog input object, output data using an analog output object, or read values from or write values to digital I/O lines. These steps are broken down into initialization, configuration, execution, and termination.

data block
The smallest “slice” of memory that the data acquisition engine can usefully manipulate.

device object
A MATLAB object that allows you to access a hardware device.

device-specific property
A property that applies only for specific hardware devices. For example, the BitsPerSample property is supported only for sound cards.

differential input
Input channel configuration where there are two signal wires associated with each input signal — one for the input signal and one for the reference (return) signal. The measurement is the difference in voltage between the two wires, which helps reduce noise and any voltage common to both wires.

digital I/O subsystem
Hardware that sends or receives digital values (logic levels). This is also referred to as a DIO subsystem.
**DMA**
Direct memory access (DMA) is a system of transferring data whereby samples are automatically stored in system memory while the processor does something else.

**engine**
A MEX-file (shared library) that stores the device objects and associated property values that control your data acquisition application, controls the synchronization of events, and controls the storage of acquired or queued data.

**engineering units properties**
Channel properties that allow you to linearly scale input or output data.

**event**
An event occurs at a particular time after a condition is met. Many event types are automatically generated by the toolbox, while others are generated only after you configure specific properties.

**execution**
The process of starting the device object and hardware device. While an analog input object is executing, you can acquire data. While an analog output object is executing, you can output data.

**FIFO buffer**
The first-in first-out (FIFO) memory buffer, which is used by data acquisition hardware to temporarily store data.

**full duplex**
A system that can send and receive information simultaneously. For sound cards, full duplex means that the device can acquire input data via an analog input subsystem while outputting data via an analog output subsystem at the same time.

**input range**
The span of input values for which an A/D conversion is valid.
**interrupts**
The slowest but most common method to move acquired data from the hardware to system memory. Interrupt signals can be generated when one sample is acquired or when multiple samples are acquired.

**line**
A component of a digital I/O subsystem that you can read digital values from, or write digital values to.

**line group**
The collection of lines contained by a digital I/O object.

**line properties**
Properties that are configured for individual lines.

**logging**
A state of Data Acquisition Toolbox where an analog input object stores acquired data to memory or a log file.

**noise**
Any measurement that is not part of the phenomena of interest.

**onboard clock**
A timer chip on the hardware board which is programmed to generate a pulse train at the desired rate. In most cases, the onboard clock controls the sampling rate of the board.

**output range**
The span of output values for which a D/A conversion is valid.

**posttrigger data**
Data that is acquired and stored in the engine after the trigger event occurs.

**precision**
A determination of how exactly a result is determined without reference to what the result means.
**pretrigger data**
Data that is acquired and stored in the engine before the trigger event occurs.

**properties**
A characteristic of the toolbox or the hardware driver that you can configure to suit your needs. The property types supported by the toolbox include base properties, device-specific properties, common properties, and channel or line properties.

**quantization**
The process of converting an infinitely precise analog signal to a binary number. This process is performed by an A/D converter.

**queuing data**
The process of storing data in the engine for eventual output to an analog output subsystem.

**running**
A state of Data Acquisition Toolbox where a device object is executing.

**sample rate**
The per-channel rate (in samples/second) that an analog input or analog output subsystem converts data.

**sampling**
The process whereby an A/D converter or a D/A converter takes a "snapshot" of the data at discrete times. For most applications, the time interval between samples is kept constant (e.g., sample every millisecond) unless externally clocked.

**scanning hardware**
Data acquisition hardware that samples a single input signal, converts that signal to a digital value, and then repeats the process for every input channel used.

**sending**
A state of Data Acquisition Toolbox where an analog output object is outputting (sending) data from the engine to the hardware.
sensor
A device that converts a physical variable into a signal that you can input into your data acquisition hardware.

signal conditioning
The process of making a sensor signal compatible with the data acquisition hardware. Signal conditioning includes amplification, filtering, electrical isolation, and multiplexing.

single-ended input
Input channel configuration where there is one signal wire associated with each input signal, and all input signals are connected to the same ground. Single-ended measurements are more susceptible to noise than differential measurements due to differences in the signal paths.

SS/H hardware
Data acquisition hardware that simultaneously samples all input signals, and then holds the values until the A/D converter digitizes all the signals.

subsystem
A data acquisition hardware component that performs a specific task. Data Acquisition Toolbox supports analog input, analog output, and digital I/O subsystems.

trigger event
An analog input trigger event initiates data logging to memory or a disk file. An analog output trigger event initiates the output of data from the engine to the hardware.
Examples

Use this list to find examples in the documentation.
Getting Started with the Data Acquisition Toolbox

“Acquiring Data” on page 2-12
“Outputting Data” on page 2-13
“Reading and Writing Digital Values” on page 2-14

Getting Started with Analog Input

“Example: Adding Channels for a Sound Card” on page 4-8
“Acquiring Data with a Sound Card” on page 4-17
“Acquiring Data with a National Instruments Board” on page 4-21

Doing More with Analog Input

“Example: Polling the Data Block” on page 5-10
“Example: Previewing and Extracting Data” on page 5-13
“Example: Voice Activation Using a Software Trigger” on page 5-23
“Example: Voice Activation and Pretriggers” on page 5-28
“Example: Voice Activation and Repeating Triggers” on page 5-30
“Example: Retrieving Event Information” on page 5-51
“Displaying Event Information with a Callback Function” on page 5-54
“Passing Additional Parameters to a Callback Function” on page 5-55
“Example: Performing a Linear Conversion” on page 5-59

Analog Output

“Outputting Data with a Sound Card” on page 6-9
“Outputting Data with a National Instruments Board” on page 6-11
“Example: Queuing Data with putdata” on page 6-18
“Example: Retrieving Event Information” on page 6-31
“Displaying the Number of Samples Output” on page 6-32
“Displaying EventLog Information” on page 6-33
“Example: Performing a Linear Conversion” on page 6-36
Digital I/O

“Example: Adding Lines for National Instruments Hardware” on page 7-13
“Example: Writing and Reading Digital Values” on page 7-18
“Example: Generating Timer Events” on page 7-22

Saving and Loading the Session

“Example: Logging and Retrieving Information” on page 8-9

Bringing Analog Data into a Model

“Example: Bringing Analog Data into a Model” on page 10-6
A

A/D converter 1-10
  input range 5-57
  sampling rate 4-11
absolute time 5-17
ac
coupling 16-5
accuracy 1-36
acquiring data 3-23
  continuous
    samples per trigger 5-22
    simultaneous input and output 6-38
    trigger repeats 5-30
  single point 12-59
actuator 1-7
adaptor kit 2-10
adaptors
  registering A-20
  supported hardware 2-9
  third-party A-20
addchannel function 12-2
  AI object 4-5
  AO object 6-4
addline function 12-8
addmuxchannel function 12-12
Advantech hardware
  properties 15-2
  troubleshooting A-3
Agilent Technologies hardware
  driver A-5
  properties 15-2
  trigger types
    AO object 6-25
    troubleshooting A-5
alias 1-43
AMUX-64T
  adding channels 12-12
  channel indices 12-83
Analog Input block 17-2
  creating a model 10-7
  running a model 10-14
  specifying parameters 10-11
analog input object
  acquisition
    continuous 5-22
    single point 12-59
  adding channels 4-5
  creating 4-3
  display summary 4-26
  engineering units 5-57
  events and callbacks 5-45
  extracting data 5-11
  logging
    data 4-15
    information to disk 8-5
  previewing data 5-8
  properties
    basic setup 4-10
    channel 13-6
    common 13-3
    configuring 3-19
  sampling rate 4-11
  starting 4-15
  status evaluation 4-25
  stopping 4-16
  triggers
    configuring 5-19
    types 4-12
Analog Output block 17-11
analog output object
  adding channels 6-3
  creating 6-2
  display summary 6-13
  engineering units 6-35
  events and callbacks 6-26
  output
    continuous 6-22
    single point 12-95
Index

properties
  basic setup 6-5
  channel 13-10
  common 13-7
  configuring 3-19
queueing data for output 6-8
sampling rate 6-5
starting 6-8
status evaluation 6-12
stopping 6-8
triggers
  configuring 6-20
  types 6-7
analog triggers
  MCC hardware 5-41
  NI hardware 5-43
  VXI hardware 5-39
analoginput function 12-14
analogoutput function 12-17
antialiasing filter 1-39
array
  data returned by getdata 5-12
device object 3-6

B
bandwidth 1-14
base properties 3-14
BiDirectionalBit property 16-2
binary vector 7-15
binvec2dec function 12-20
BitsPerSample property 16-3
block
  Analog Input 17-2
  Analog Output 17-11
  Digital Input 17-19
  Digital Output 17-24
Block Library 10-3
blocking function
  getdata 5-12
  putdata 6-8
blocks
  overview 10-2
board ID 4-3
buffer
  configuration 14-2
  extracting data 5-12
  previewing data 5-9
  queuing data 6-16
BufferingConfig property 14-2
BufferingMode property 14-5

calibration 1-5
callback function 5-52
callback properties
  AI object 5-45
  AO object 6-26
  saving property values to a MAT-file 8-2
Channel Editor GUI
  Channel Display pane 9-8
  Channel pane 9-10
  Channel Properties pane 9-17
Channel Exporter GUI 9-29
channel gain list 4-7
channel group
  AI object 4-5
  AO object 6-3
channel names 4-8
channel properties 3-13
channel properties 3-13
  AI object 13-6
  AO object 13-10
Channel property 14-7
channel skew 5-6
ChannelName property 14-9
channels 3-9
adding
  AI object 4-5
  AO object 6-3
descriptive names 4-8
input configuration 5-2
mapping to hardware IDs 3-10
Oscilloscope
  hardware 9-3
  math 9-10
  reference 9-10
  referencing 4-7
  scan order 4-6
ChannelSkew property 14-11
ChannelSkewMode property 14-12
cleaning up the MATLAB environment
  clear function 3-27
  daqfind function 12-44
  delete function 3-27
  clear function 12-21
clipping 6-36
clock function 5-36
clocked acquisition 1-27
ClockSource property 14-15
COLA property 16-4
common properties 3-13
  AI object 13-3
  AO object 13-7
  DIO object 13-11
configuring property values
  dot notation 3-20
  set function 3-20
constructor 3-5
Contents 2-17
continuous acquisition
  example using AI and AO 6-38
  samples per trigger 5-22
  trigger repeats 5-30
continuous output 6-22
Coupling property 16-5
creation function 3-5
custom adaptors 2-10

D
D/A converter 1-11
  output range 6-35
  sampling rate 6-5
daqcallback
  AI example 5-54
  default property value
    data missed event (AI) 5-46
    run-time error event:AI object 5-47
    run-time error event:AO object 6-27
daqcallback function 12-23
daqfind function 12-25
daqhelp function 12-28
daqhwinfo function 12-31
daqmem function 12-34
daqread function 12-37
daqregister function 12-38
daqreset function 12-40
daqsupport function A-21
data
  extracting from engine 5-11
  previewing 5-8
  queuing for output 6-16
data acquisition session 3-2
  acquiring data (AI) 4-15
  adding channels
    AI object 4-5
    AO object 6-3
  adding lines 7-6
  cleaning up 3-27
  configuring properties
    AI object 4-10
    AO object 6-5
  creating a device object
    AI object 4-3
    AO object 6-2
    DIO object 7-3
loading 8-2
outputting data (AO) 6-7
saving 8-2
Data Acquisition Toolbox Block Library
opening 10-3
Data Acquisition Toolbox blocks 10-2
data block C-2
polling 5-10
data flow
  acquired data 2-7
  output data 2-8
data missed event 5-46
data tips (Oscilloscope) 9-7
DataMissedFcn property 14-18
dc
coupling 16-5
debugging your hardware A-21
daqsupport A-21
dec2binvec function 12-41
DefaultChannelValue property 14-20
delete function 12-43
demos 2-18
descriptive names
  channels 4-8
  lines 7-13
device ID 4-3
device object 3-5
  array 3-6
    simultaneous input and output 6-38
  configuring property values 3-19
  copying 3-7
  creating
    AI object 4-3
    AO object 6-2
    DIO object 7-3
invalid 3-8
loading 8-2
saving 8-2
specifying property names 3-21
starting 3-24
stopping 3-25
device-specific properties 3-14
  Advantech hardware 15-2
  Agilent hardware 15-2
  Keithley hardware 15-3
  MCC hardware 15-4
  NI hardware 15-5
  parallel port 15-5
  sound cards 15-6
differential inputs 1-29
digital I/O object
  adding lines 7-6
  creating 7-3
  display summary 7-24
  parallel port adaptor 7-4
  port types 7-8
  properties
    common 13-11
    line 13-12
    reading values 7-17
    starting 7-21
    status evaluation 7-24
    stopping 7-21
    writing values 7-15
Digital Input block 17-19
Digital Output block 17-24
digital triggers
  Agilent hardware
    AO object 6-25
  MCC hardware (AI) 5-40
  NI hardware
    AI object 5-42
    AO object 6-24
  VXI hardware
    AI object 5-38
digital values
  reading 7-17
  writing 7-15
digitalio function 12-46
Direction property 14-21
disk logging 14-49
disp function 12-48
display summary
  AI object 4-26
  AO object 6-13
  DIO object 7-24
DMA 1-32
  NI hardware 16-36
documentation examples 2-17
dot notation
  configuring property values 3-20
  returning property values 3-18
  saving property values to an M-file 8-2
driver
  Agilent hardware A-5
  MCC hardware A-7
  NI hardware A-9

e
E1432 driver A-5
eengine 2-6
  extracting data from 5-11
  queuing data to 6-16
engineering units
  AI object 5-57
  AO object 6-35
event log
  AI object 5-48
  AO object 6-29
event types
  data missed (AI) 5-46
  input overrange (AI) 5-46
  run-time error
    AI object 5-47
    AO object 6-27
  samples acquired (AI) 5-47
  samples output (AO) 6-27
start
  AI object 5-47
  AO object 6-28
stop
  AI object 5-48
  AO object 6-28
timer
  AI object 5-48
  AO object 6-28
trigger
  AI object 5-48
  AO object 6-28
EventLog property 14-23
events 2-6
  AI object 5-45
  AO object 6-26
displaying with showdaqevents
  AO object 6-23
displaying with showdaqevents function
  AI object 5-35
eexample index 2-17
eexamples
  acquiring data
    NI hardware 4-21
    sound card 4-17
  adding lines 7-13
  bringing analog data into a Simulink model 10-6
  generating timer events (DIO) 7-22
  logging and retrieving information (AI) 8-9
  outputting data with a National Instruments board 6-11
  outputting data with a sound card 6-9
  performing a linear conversion
    AI object 5-59
    AO object 6-36
  polling the data block (AI) 5-10
  previewing and extracting data 5-13
  reading and writing DIO values 7-18
retrieving event information
  AI object 5-51
  AO object 6-31
using blocks 10-6
using callback properties
  AI object 5-54
  AO object 6-32
using putdata 6-18
voice activation (AI)
  pretriggers 5-28
  repeating triggers 5-30
  software trigger 5-23
execution
  AI object 4-15
  AO object 6-7
  DIO object 7-20
exporting (Oscilloscope)
  channel data 9-29
  measurements 9-30
external clock 1-27
  clock sources 14-15
ExternalSampleClockSource property 16-7
ExternalScanClockSource property 16-8
extracting data 5-11
  event information 12-57
  native data 14-57
  time information 5-16

F
fft 4-19
FIFO 1-31
  TransferMode 16-36
filtering 1-39
finding device objects 12-25
floating signal 1-28
flow of data
  acquired 2-7
  output 2-8
flushdata 12-50
full duplex A-18
  BitsPerSample property 16-3
function handle 5-52
functions
  addchannel 12-2
  addline 12-8
  addmuxchannel 12-12
  analoginput 12-14
  analogoutput 12-17
  binvec2dec 12-20
  clear 12-21
  daqcallback 12-23
  daqfind 12-25
  daqhelp 12-28
  daqhwinfo 12-31
  daqmem 12-34
  daqread 12-37
  daqregister 12-38
  daqreset 12-40
  dec2binvec 12-41
  delete 12-43
  digitalio 12-46
  disp 12-48
  flushdata 12-50
  get 12-52
  getdata 12-54
  getsample 12-59
  getvalue 12-61
  inspect 12-63
  ischannel 12-65
  isdioline 12-66
  islogging 12-67
  isrunning 12-69
  issending 12-71
  isvalid 12-73
  length 12-76
  load 12-78
  makenames 12-81
  muxchanidx 12-83
Index

obj2mfile 12-85
peekdata 12-88
propinfo 12-90
putdata 12-92
putsample 12-95
putvalue 12-97
save 12-99
set 12-101
setverify 12-104
showdaqevents 12-107
size 12-110
softscope 12-113
start 12-122
stop 12-124
trigger 12-126
wait 12-127

G

gain 1-26
   engineering units (AI) 5-57
gain list 4-7
get function 12-52
getdata function 12-54
getsample function 12-59
getvalue function 12-61
grounded signal 1-28
GroundingMode property 16-9
GUI
   Channel Editor
      Channel Display pane 9-8
      Channel pane 9-10
      Channel Properties pane 9-17
   Channel Exporter 9-29
   Hardware Configuration 9-4
   Measurement Editor
      Measurement pane 9-23
      Measurement Properties pane 9-28
   Measurement Exporter 9-30
   Oscilloscope 9-3
   Scope Editor
      Scope pane 9-7
      Scope Properties pane 9-16

H

hardware
   initializing 12-40
   resources 2-19
   scanning 1-21
   setting up 1-7
   simultaneous sample and hold 1-23
   supported vendors 2-9
   hardware channels (Oscilloscope) 9-3
   Hardware Configuration GUI 9-4
   hardware ID
      channel 4-5
      device (board) 4-3
      line 7-6
      mapping to channels 3-10
      port 7-6
   hardware triggers
      AI object 5-37
      AO object 6-24
   help 2-23
   holding the last output value 16-15
   HP E1432 driver A-5
   HwChannel property 14-26
   HwDigitalTriggerSource property 16-10
   HwLine property 14-28

I

ID

   channel 4-5
      HwChannel 4-7
   device (board) 4-3
   line 7-6
      HwLine 7-12
   mapping to channels 3-10
Index

port 7-6
immediate trigger
  AI object 5-22
  AO object 6-22
Index property 14-30
indexing
  channel array 4-7
  line array 7-12
initializing the hardware 12-40
InitialTriggerTime property 14-32
input overrange event 5-46
input range 1-26
  engineering units 5-58
InputMode property 16-11
InputOverRangeFcn property 14-34
InputRange property 14-36
InputSource property 16-13
InputType property 14-39
inspect function 12-63
Inspector, property 3-22
InstaCal A-7
  hardware configuration 2-19
internal clock 1-28
interrupts 1-32
  NI hardware 16-36
invalid device object 3-8
ischannel function 12-65
isdioline function 12-66
islogging function 12-67
isnan function 5-35
isrunning function 12-69
issending function 12-71
isvalid function 12-73

K
Keithley hardware
  properties 15-3

L
least significant bit (DIO) 7-12
length function 12-76
line group 7-6
line names 7-13
line object 7-6
line properties 3-13
Line property 14-42
line-configurable device 7-8
Linear conversion
  AI object with asymmetric data 5-60
LineName 7-13
LineName property 14-44
lines 3-9
  adding 7-6
  descriptive names 7-13
  referencing 7-12
load function 12-78
loading
  device objects
    M-file 8-3
    MAT-file 8-4
  Oscilloscope configuration 9-32
LogFile property 14-46
logging
  data to memory 3-23
  information to disk (AI) 8-5
    file name specification 8-6
    multiple files 8-6
    retrieving data with daqread 8-7
Logging property 14-47
LoggingMode property 14-49
LogToDiskMode property 14-51

J
jitter 1-27
| M | makenames function 12-81  
managing data  
 acquired 5-8  
 output 6-16  
 manual trigger  
 AI object 5-22  
 AO object 6-22  
 ManualTriggerHWOn property 14-53  
 mapping channels to hardware IDs 3-10  
 MAT-file  
 device objects, saving to 8-4  
 properties, saving to 8-2  
 math channels (Oscilloscope) 9-10  
 maximum samples queued 14-55  
 MaxSamplesQueued property 14-55  
 Measurement and Automation Explorer A-10  
 hardware configuration 2-19  
 Measurement Computing hardware  
 channel configuration 5-3  
 driver A-7  
 properties 15-4  
 trigger types (AI) 5-39  
 troubleshooting A-7  
 Measurement Editor GUI  
 Measurement pane 9-23  
 Measurement Properties pane 9-28  
 Measurement Exporter GUI 9-30  
 memory resources C-2  
 mono mode 4-8  
 most significant bit (DIO) 7-12  
 multifunction boards 1-10  
 multiple device objects  
 array 3-6  
 starting 6-38  
 stopping 6-39  
 multiplexing 1-16  
 mux board  
 adding channels 12-12  
 channel indices 12-83  
 muxchanidx function 12-83  

| N | Name property 14-56  
National Instruments hardware  
 channel configuration 5-4  
 data transfer mechanisms 16-36  
 driver A-9  
 properties 15-5  
 trigger types  
 AI object 5-41  
 AO object 6-24  
 troubleshooting A-9  
 native data  
 getdata 12-54  
 offset 14-57  
 putdata 12-92  
 scaling 14-59  
 NativeOffset property 14-57  
 NativeScaling property 14-59  
 NI-DAQ driver A-9  
 noise 1-38  
 NumMuxBoards property 16-14  
 Nyquist frequency 4-18  
 Nyquist theorem 1-40  

| O | obj2mfile function 12-85  
 object constructor 3-5  
 onboard clock 1-27  
 one-shot acquisition 5-30  
 online help 2-23  
 Oscilloscope  
 displaying channels 9-6  
 exporting data 9-29  
 making measurements 9-22  
 opening 9-3  
 saving and loading the configuration 9-32  

Index-9
Index

scaling channel data 9-15
triggering 9-19
OutOfDataMode property 16-15
output range 6-35
OutputRange property 14-61
outputting data 3-23
continuous 6-22
holding the last value 16-15
single point 12-95
overloaded functions 1-45
overrange condition 1-26

P
parallel port
adaptor 7-4
device-specific properties 15-5
Parent property 14-63
PC clock 1-27
peekdata function 12-88
polarity 1-26
engineering units (AI) 5-57
polling the data block 5-10
port characteristics 7-8
Port property 14-64
port-configurable device 7-8
PortAddress property 16-17
postriggers 5-28
precision 1-36
pretriggers 5-27
previewing data 5-8
properties
BiDirectionalBit 16-2
BitsPerSample 16-3
BufferingConfig 14-2
BufferingMode 14-5
Channel 14-7
ChannelName 14-9
ChannelSkew 14-11
ChannelSkewMode 14-12
ClockSource 14-15
COLA 16-4
Coupling 16-5
DataMissedFcn 14-18
DefaultChannelValue 14-20
Direction 14-21
EventLog 14-23
ExternalSampleClockSource 16-7
ExternalScanClockSource 16-8
GroundingMode 16-9
HwChannel 14-26
HwDigitalTriggerSource 16-10
HwLine 14-28
Index 14-30
InitialTriggerTime 14-32
InputMode 16-11
InputOverRangeFcn 14-34
InputRange 14-36
InputSource 16-13
InputType 14-39
Line 14-42
FieldName 14-44
LogFileName 14-46
Logging 14-47
LoggingMode 14-49
LogToDiskMode 14-51
ManualTriggerHwOn 14-53
MaxSamplesQueued 14-55
Name 14-56
NativeOffset 14-57
NativeScaling 14-59
NumMuxBoards 16-14
OutOfDataMode 16-15
OutputRange 14-61
Parent 14-63
Port 14-64
PortAddress 16-17
RampRate 16-18
RepeatOutput 14-65
Running 14-67

Index-10
RuntimeErrorFcn 14-68
SampleRate 14-70
SamplesAcquired 14-72
SamplesAcquiredFcn 14-73
SamplesAcquiredFcnCount 14-75
SamplesAvailable 14-76
SamplesOutput 14-77
SamplesOutputFcn 14-78
SamplesOutputFcnCount 14-79
SamplesPerTrigger 14-80
Sending 14-82
SensorRange 14-83
SourceMode 16-19
SourceOutput 16-20
Span 16-22
StandardSampleRates 16-24
StartFcn 14-84
StopFcn 14-86
StopTriggerChannel 16-26
StopTriggerCondition 16-28
StopTriggerConditionValue 16-30
StopTriggerDelay 16-31
StopTriggerDelayUnits 16-32
StopTriggerType 16-33
Sum 16-35
Tag 14-88
Timeout 14-89
TimerFcn 14-91
TimerPeriod 14-93
TransferMode 16-36
TriggerChannel 14-94
TriggerCondition 14-95
TriggerConditionValue 14-101
TriggerDelay 14-103
TriggerDelayUnits 14-105
TriggerFcn 14-106
TriggerRepeat 14-108
TriggersExecuted 14-109
TriggerType 14-110
Type 14-113
Units 14-114
UnitsRange 14-115
UserData 14-117
property characteristics 2-23
Property Inspector 3-22
property types
  base 3-14
  channel 3-13
    AI object 13-6
    AO object 13-10
  common 3-13
    AI object 13-3
    AO object 13-7
    DIO object 13-11
  device-specific 3-14
    Advantech hardware 15-2
    Agilent hardware 15-2
    Keithley hardware 15-3
    MCC hardware 15-4
    NI hardware 15-5
    parallel port 15-5
    sound cards 15-6
  line 3-13
    DIO object 13-12
Oscilloscope
  channel 9-16
  display 9-9
  measurement 9-26
  trigger 9-21
property values
  configuring 3-19
  default 3-21
  saving 8-2
  specifying names 3-21
propinfo function 12-90
putdata function 12-92
putsample function 12-95
putvalue function 12-97
quantization 1-24
queuing data for output 6-16
   maximum number of samples 14-55
Quick Reference Guide 2-18

RampRate property 16-18
read-only properties 2-23
reading digital values 7-17
reference channels (Oscilloscope) 9-10
registering your adaptor A-20
relative time 5-16
repeating triggers 5-30
RepeatOutput property 14-65
resetting the hardware 12-40
retrieving data from a log file 8-7
returning property values
   dot notation 3-18
   get 3-17
   set function 3-16
run-time error event
   AI object 5-47
   AO object 6-27
running device objects 3-23
Running property 14-67
RuntimeErrorFcn property 14-68

SampleRate property 14-70
samples acquired event 5-47
samples output event 6-27
samples per trigger
   postrigger data 5-28
   pretrigger data 5-27
SamplesAcquired property 14-72
SamplesAcquiredFcn property 14-73
SamplesAcquiredFcnCount property 14-75
samplesAvailable property 14-76
SamplesOutput property 14-77
SamplesOutputFcn property 14-78
SamplesOutputFcnCount property 14-79
SamplesPerTrigger property 14-80
sampling 1-20
sampling rate
   AI subsystem 5-5
   AO subsystem 6-5
saturation 6-36
save function 12-99
saving
   device objects
      M-file 8-2
      MAT-file 8-4
   information to disk (AI) 8-5
   Oscilloscope configuration 9-32
   property values to a MAT-file 8-2
scaling the data
   AI object 5-57
   AO object 6-35
scanning hardware 1-21
   channel order 4-6
Scope Editor GUI
   Scope pane 9-7
   Scope Properties pane 9-16
sending data 3-23
Sending property 14-82
SensorRange property 14-83
sensors 1-11
   range 14-83
session 3-2
   loading 8-2
   saving 8-2
set function 12-101
settling time 1-36
setverify function 12-104
showdaqevents function 12-107
signal conditioning 1-15
Simulink block
Analog Input 17-2
Analog Output 17-11
Digital Input 17-19
Digital Output 17-24
Simulink blocks 10-2
  overview 10-2
Simulink Library Browser 10-3
Simulink model 10-6
simultaneous input and output 6-38
simultaneous sample and hold hardware 1-23
single-ended inputs 1-30
single-point
  acquisition 12-59
  output 12-95
size function 12-110
skew 5-6
software scope function 12-113
software clock 1-28
  MCC hardware 14-15
software trigger 5-22
sound cards
  channel configuration 5-5
  device-specific properties 15-6
  mono mode 4-8
  standard sample rates 16-24
  stereo mode 4-9
  troubleshooting A-12
SourceMode property 16-19
SourceOutput property 16-20
Span property 16-22
StandardSampleRates property 16-24
start event
  AI object 5-47
  AO object 6-28
start function 12-122
StartFcn
  AI object 5-47
  AO object 6-28
StartFcn property 14-84
starting multiple device objects 6-38
state
  logging 3-23
  running 3-23
  sending 3-23
status evaluation
  AI object 4-25
  AO object 6-12
  DIO object 7-24
stereo mode 4-9
stop event
  AI object 5-48
  AO object 6-28
stop function 12-124
StopFcn property 14-86
StopTriggerChannel property 16-26
StopTriggerCondition property 16-28
StopTriggerConditionValue property 16-30
StopTriggerDelay property 16-31
StopTriggerDelayUnits property 16-32
StopTriggerType property 16-33
Sum property 16-35
synchronizing triggers 14-53
Tag property 14-88
third-party adaptors A-20
time
  absolute 5-17
  initial trigger 5-17
  relative 5-16
Timeout property 14-89
timer event
  AI object 5-48
  AO object 6-28
  DIO object 7-20
TimerFcn property 14-91
TimerPeriod property 14-93
toolbox components
  data acquisition engine 2-6
hardware driver adaptor 2-9
M-files 2-6
transducer 1-7
TransferMode property 16-36
trigger event
  AI object 5-48
  AO object 6-28
trigger function 12-126
TriggerChannel property 14-94
TriggerCondition property 14-95
TriggerConditionValue property 14-101
TriggerDelay property 14-103
TriggerDelayUnits property 14-105
triggered
  acquisition 5-19
  output 6-20
TriggerFcn property 14-106
TriggerRepeat property 14-108
triggers
  delays 5-26
  Oscilloscope 9-19
  postriggers 5-28
  pretriggers 5-27
  repeating 5-30
samples acquired for each trigger 4-13
synchronizing for AI and AO 14-53
times
  AI object 5-36
  AO object 6-23
  initial trigger 5-17
trigger conditions (AI) 5-20
trigger types
  AI object 5-20
  AO object 6-21
  Oscilloscope 9-19
TriggersExecuted property 14-109
TriggerType property 14-110
Type property 14-113
U
undersampling 1-40
Units property 14-114
UnitsRange property 14-115
Universal Library driver A-7
UserData property 14-117
  saving values to a MAT-file 8-2
  using Simulink blocks 10-2
  using the block library 10-6
V
verifying property values 4-12
voice activation example 5-23
VXI Technology hardware
  channel configuration 5-4
  decimation factor 16-22
  trigger types
    AI object 5-37
W
wait function 12-127
Workspace browser
  DAQ Help 2-23
  Display Hardware Info 2-22
  Display Summary 4-26
  Property Editor 3-22
  Show DAQ Events
    AI object 5-52
    AO object 6-31
writing digital values 7-15