Data Acquisition Toolbox

For Use with MATLAB®

Computation

Visualization

Programming

User's Guide

Version 1.0 (Beta 2)

How to Contact The MathWorks:

508-647-7000 Phone

508-647-7001 Fax

The MathWorks, Inc.
24 Prime Park Way
Natick, MA 01760-1500 Mail

`http://www.mathworks.com`	Web
`ftp.mathworks.com`	Anonymous FTP server
`comp.soft-sys.matlab`	Newsgroup

`support@mathworks.com`	Technical support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`subscribe@mathworks.com`	Subscribing user registration
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

Data Acquisition Toolbox User's Guide

Printing History: November 1998 First printing for Beta 1
January 1999 Second printing for Beta 2 (online only)

Contents

Before You Begin

1 Getting Started with the Data Acquisition Toolbox

2 Data Acquisition Toolbox Concepts

3 Getting Started With Analog Input

4 Doing More With Analog Input

5 Analog Output

6 Function Reference

7 Property Reference

A Device-Specific Properties

B Testing Your Hardware

Before You Begin

What Is the Data Acquisition Toolbox?

The Data Acquisition Toolbox, includes a collection of M-files and DLLs that provide:

- A framework for bringing live, measured data into MATLAB®
- Access to a wide range of data acquisition devices and vendors
- A consistent and flexible MATLAB command line interface to all supported vendors.

Intended Audience

This document assumes that you are familiar with the use of MATLAB and that you have a basic understanding of data acquisition concepts.

Related Products

Requirements

The Data Acquisition Toolbox runs on Windows95, Windows98, and Windows NT 4.0.

The Data Acquisition Toolbox requires:

- MATLAB 5.3 (R11)
- A supported data acquisition device

Note: The supported devices for the Beta 2 version of the Data acquisition Toolbox include sound cards, and National Instruments E-series boards (analog input and analog output subsystems only).

What Is MATLAB?

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm development
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or Fortran.

The name MATLAB stands for *matrix laboratory*. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK

and EISPACK projects, which together represent the state-of-the-art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of application-specific solutions called *toolboxes*. Very important to most users of MATLAB, toolboxes allow you to *learn* and *apply* specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

See the MATLAB documentation set for more information.

Documentation Conventions

The documentation conventions used in this book are shown below.

To Indicate	This Guide Uses	Example
Example code	`Monospace type`	To assign the value 5 to `A`, enter `A = 5`
Function names/syntax	`Monospace type`	The `cos` function finds the cosine of each array element.
Keys	**Boldface** with an initial capital letter	Press the **Return** key.
Mathematical expressions	Variables in *italics*. Functions, operators, and constants in standard type.	This vector represents the polynomial $p = x^2 + 2x + 3$

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

Menu names, menu items, and controls	**Boldface** with an initial capital letter	Choose the **File** menu.
New terms	NCS *italics*	An *array* is an ordered collection of information.

Installation

To acquire live, measured data in the MATLAB environment, you must install:

- The Data Acquisition Toolbox
- Data acquisition hardware
- Software associated with your hardware (drivers, support libraries, etc.).

Installation of these items is discussed below.

Toolbox Installation

Your platform-specific MATLAB *Installation Guide* provides essentially all of the information you need to install the Data Acquisition Toolbox. Vendor-specific drivers or support libraries are not installed with the Data Acquisition Toolbox.

Prior to installation, you must obtain a License File or Personal License Password from The MathWorks. The License File or Personal License Password identifies the products you are permitted to install and use.

If you experience installation difficulties and have Web access, connect to The MathWorks home page (`http://www.mathworks.com`). Look for the license manager and installation information under the `Tech Notes/FAQ` link under `Tech Support Info`.

Hardware and Driver Installation

Installation of your hardware is described by its documentation. Installation of drivers and any other associated software should be described by this documentation as well.

Note: Vendor-specific drivers or support libraries are not installed with the Data Acquisition Toolbox.

1

Getting Started with the Data Acquisition Toolbox

Collecting Live Data

Perhaps the most effective way to get started with the Data Acquisition Toolbox is to collect some live data. If you have a sound card installed, you can run the following code which collects one second of data.

```
ai = analoginput('winsound')
addchannel(ai, 1)
set(ai, 'SampleRate', 8192);
start(ai)
data = getdata(ai);
plot(data)
```

With just six lines of code, you can connect the Data Acquisition Toolbox to your hardware, acquire data, and plot the data. These six lines are described below.

1. Create a device object – Device objects allow you to connect the toolbox to specific hardware subsystems.
2. Add one or more channels to the device object – A device object can perform a useful task only when at least one channel is added to it.
3. Control acquisition behavior by assigning values to properties – Any property not explicitly assigned a value uses its default value.
4. Execute the device object –The device object must start before data can be acquired
5. Extract acquired data – Once data is acquired, it must be stored in a MATLAB variable.
6. Analyze the data.

These six basic steps are followed for even the most complex applications.

The Display Summary

Summary information for your data acquisition application is displayed whenever a reference to a device object or channel is not suppressed with a semicolon. You may find the display summary is a convenient way to quickly evaluate your acquisition status.

Understanding the Toolbox Capabilities

The Data Acquisition Toolbox provides several resources to assist you in getting started with the product:

- The `Readme` and `Contents` M-files
- Demos
- Documentation examples
- Quick Reference Guide

These resources are described below.

The Readme and Contents M-files

The `Readme.m` file contains general information about the MATLAB Data Acquisition Toolbox. The file consists of three parts. The first part describes how to install the toolbox. The second part describes general toolbox functionality. The third part lists the known limitations and bugs. `Readme.m` is invoked by typing `info daq` at the command line.

The `Contents.m` file describes the general functionality of the Data Acquisition Toolbox. This file is invoked by typing `help daq` at the command line.

Demos

The Data Acquisition Toolbox includes several useful demos, which are located in the `daqdemos` directory. You can launch any installed demo by typing

```
demos
```

at the command line. Alternatively, you can launch any Data Acquisition Toolbox command-line demo by typing

```
daqschool
```

at the command line. You can also run individual demos by typing the appropriate M-file name at the command line. The demos are described below.

Common Demos

The common demos illustrate features which are common to all supported device objects. These demos are listed below.

The demo...	Shows you...
`demodaq_action`	How to create action functions for displaying event information to the MATLAB command window and plotting data as it is acquired
`demodaq_intro`	How to determine which hardware and adaptors are installed on your machine, how to determine what device objects can be created for each installed adaptor, and how to obtain help on properties and functions
`demodaq_save`	How to save and load device objects and how to convert device objects to equivalent MATLAB commands

Analog Input Demos

The analog input demos are listed below.

The demo...	Shows you...
`daqrecord`	How to record data from the specified adaptor
`daqscope`	An example oscilloscope window for displaying input signals
`demoai_channel`	How to add and configure channels for an analog input object
`demoai_fft`	An example fast Fourier transform (FFT) of an input signal from a sound card
`demoai_intro`	How to configure properties, add channels, acquire data, and plot data for an analog input object

The demo...	Shows you...
demoai_logging	How to configure an analog input object for data logging
demoai_trig	How to acquire data using an immediate, manual, and software trigger, and how to configure various triggering properties for each trigger type

Analog Output Demos

The analog output demos are listed below.

The demo...	Shows you...
daqfcngen	An example function generator window which can be used with analog output objects
daqplay	How to output data to the specified adaptor
daqsong	How to output the data from HANDEL.MAT to a sound card
demoao_channel	How to add and configure channels for an analog output object
demoao_intro	How to configure properties, add channels, and output data to a soundcard
demoao_trig	How to output data using an immediate and manual trigger and how to configure various triggering properties for each trigger type

Documentation Examples

When you encounter examples or code snippets in this book, you may want to try them for yourself. An easy way to do this is to cut the appropriate text from the pdf or HTML versions of this book and paste them into the MATLAB workspace. The pdf and HTML content is accessed through the Help Desk. To invoke the Help Desk, type

```
helpdesk
```

at the MATLAB command line.

Some examples are constructed as “mini-applications”, which 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 part of the toolbox and are located in the `daqdemos` directory. In the documentation, they are identified in the margin by their M-file name. All documentation example M-files begin with `daqdoc`. To run an example, simply type the M-file name at the command line.

Quick Reference Guide

The quick reference guide gives you a brief but complete overview of the toolbox capabilities, functions, and properties. You may find it useful to print this guide and keep it handy when using the toolbox.

The quick reference guide can be accessed through the Data Acquisition Toolbox link off the Help Desk. The Help Desk is launched by typing

```
helpdesk
```

at the command line.

Evaluating Your Hardware Resources

You can evaluate your data acquisition hardware resources with the `daqhwinfo` function. Hardware resources include installed boards, hardware drivers, and adaptors. The information returned by `daqhwinfo` depends on the supplied arguments:

- `daqhwinfo`

  Returns general data acquisition information about the toolbox, MATLAB, and the installed adaptors.

- `daqhwinfo('adaptor')`

  Returns information for the specified adaptor.

- `daqhwinfo(obj)`

  Returns driver and hardware information for the device object `obj`.

- `daqhwinfo(obj, 'Property')`

  Returns information for the specified property for device object `obj`.

The hardware information related to the sound card adaptor is given below.

```
daqhwinfo('winsound')
ans =
       AdaptorDllName: 'D:\v5\toolbox\daq\private\mwwinsound.dll'
    AdaptorDllVersion: '1.0.2.1'
          AdaptorName: 'winsound'
           BoardNames: {'Sound Blaster Record'}
    InstalledBoardIds: '[0]'
ObjectConstructorName: {'analoginput('winsound',0)' [1x26 char]}
```

This information can be displayed at any time to assist you in configuring your data acquisition session.

The driver and hardware information for the sound card example on page 1-2 is shown below.

```
daqhwinfo(ai)
ans =

              AutoCalibrate: 'Off'
                       Bits: 16
           ConversionOffset: 0
                   Coupling: {'AC Coupled'}
                 DeviceName: 'Sound Blaster Record'
       DifferentialChannels: 0
                 DriverName: 'winsound'
                      Gains: []
                         ID: 0
                 InputRanges: [-1 1]
               MaxSampleRate: 44100
               MinSampleRate: 8000
             NativeDataType: 'Int16'
                   Polarity: {'Bipolar'}
                 SampleType: 'SimultaneousSample'
        SingleEndedChannels: 2
              SubsystemType: 'AnalogInput'
              TotalChannels: 2
    VendorDriverDescription: 'Windows Multimedia Driver'
        VendorDriverVersion: '2.0''
```

Getting Online Help

The available online help resources for Data Acquisition Toolbox include:

- M-file help
- HTML– and PDF–based help available through the Help Desk
- The `daqhelp` function
- The `propinfo` function

The `daqhelp` and `propinfo` functions are discussed below.

The daqhelp function

`daqhelp` can be used to display information for functions, properties, device objects, and constructors. A device object need not exist for you to obtain this information. For example, to display help for the `SampleRate` property

```
daqhelp(ai, 'SampleRate');
```

For more information about `daqhelp`, refer to Chapter 6, "Function Reference."

The propinfo function

`propinfo` returns information about properties. The information is returned as a structure containing the fields shown below.

Field	Description
`Type`	The property data type (e.g., double, string)
`Constraint`	Constraints on property values (e.g., none, bounded)
`ConstraintValue`	Property value constraint (e.g., range of valid values, elements of an enumerated list)
`DefaultValue`	The property default value
`ReadOnly`	If a property is read-only, a "1" is returned. Otherwise a "0" is returned.

Field	Description
ReadOnlyRunning	If a property cannot be set while the device object is running, a “1” is returned. Otherwise a “0” is returned.
DeviceSpecific	If the property is device-specific, then a “1” is returned. Otherwise a “0” is returned.

This information can be used to evaluate the allowed value(s) for a given property or to programmatically configure property values. For example, to display all the propinfo field values for the SampleRate property, type

```
info = propinfo(ai);
info.SampleRate
ans =
               Type: 'double'
         Constraint: 'Bounded'
    ConstraintValue: [8000 44100]
       DefaultValue: 8000
           ReadOnly: 0
    ReadOnlyRunning: 1
     DeviceSpecific: 0
```

For more information about propinfo, refer to Chapter 6, “Function Reference.”

2

Data Acquisition Toolbox Concepts

Overview

This chapter presents the main concepts behind the Data Acquisition Toolbox.

Easy Acquisition and Analysis

You can easily acquire data from your hardware and directly analyze the data using the full power of MATLAB.

Same Interface to Different Hardware

The toolbox provides the same interface to many different data acquisition devices, including sound cards, National Instruments E-series boards, and Hewlett-Packard's VXI hardware.

Object-Based Design

You create objects in MATLAB that are directly associated with your hardware device. These *device objects* provide a gateway to all of the hardware's functionality and allow you to control the behavior of the acquisition.

Configure Objects with Properties

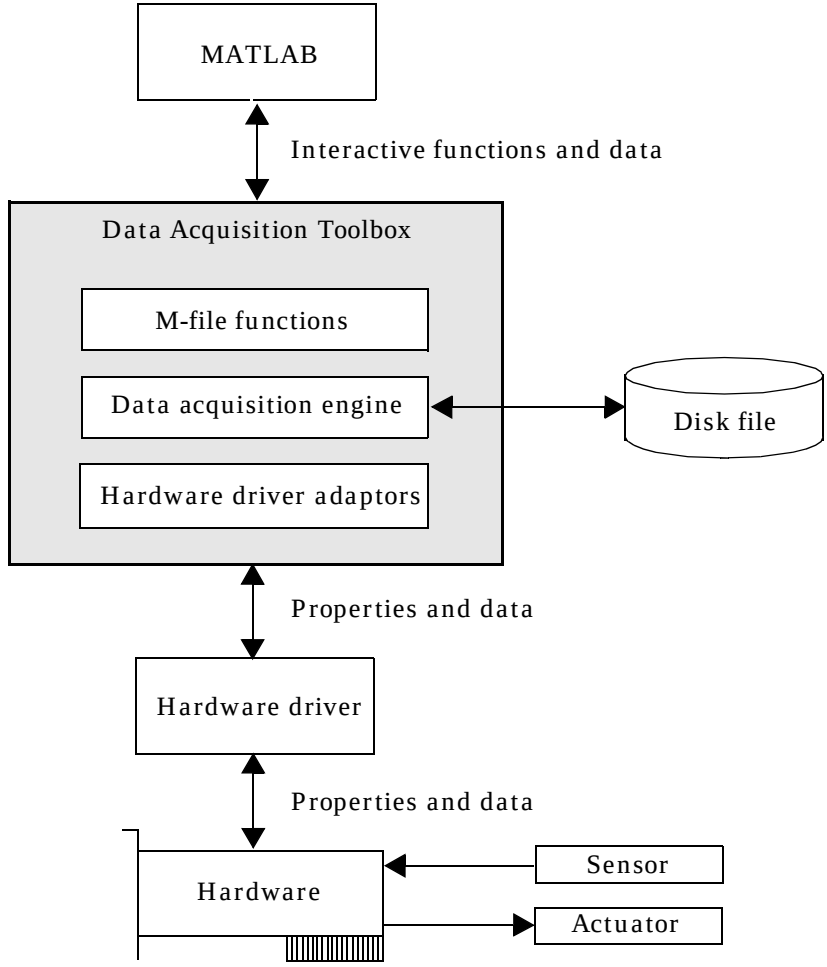
You configure object property values to control the behavior of your acquisition. The toolbox provides a collection of base properties that apply to all supported hardware, and device-specific properties that apply on a per-device basis.

Perform Tasks with Functions

You call specific toolbox functions to perform data acquisition tasks.

Toolbox Components

The Data Acquisition Toolbox consists of three distinct components: M-file functions, the data acquisition engine, and hardware driver adaptors. An understanding of these components and how they relate to each other, to MATLAB, and to your hardware will go a long way towards helping you understand how to use the toolbox more effectively. These components and relationships are shown below.



The diagram illustrates the order in which information flows from component to component. Information consists of data and *property values*. Property values allow you to control the behavior of your data acquisition application:

- Information flows from MATLAB to the hardware, and from the hardware to MATLAB.
- Data can be loaded into MATLAB from data acquisition hardware or from a disk file.

The Data Acquisition Toolbox components are described below.

M-file Functions

To perform any data acquisition task, you must first call one or more M-file functions from the MATLAB environment. The M-files you call depend on your specific data acquisition task. These functions allow you to:

- Create device objects which provide a gateway to all of the hardware's functionality and allow you to control the behavior of the acquisition. Device objects are discussed on page 2-8.
- Control your acquisition behavior by passing information to, or receiving information from your hardware
- Evaluate your acquisition status and resources by requesting information from your hardware

The information passed to, or requested from, the hardware is in the form of property values or data. Property values reflect the full functionality of both the Data Acquisition Toolbox and your hardware and are discussed on page 2-15. For detailed property descriptions, refer to Chapter 7 and Appendix A.

The Data Acquisition Engine

The data acquisition engine (or just *engine*) is a MEX-file DLL that:

- Stores the device objects and associated property values that control your data acquisition application (see page 2-8)
- Controls the synchronization of events (see page 2-19)
- Controls the storage of acquired data to disk or memory (see page 2-22)

While the engine performs these tasks, MATLAB can be used for other purposes such as analyzing the acquired data. That is, the engine and MATLAB are *asynchronous*.

One of the main features of the Data Acquisition Toolbox is the data analysis power provided by the MATLAB environment. Therefore, it is important to understand how acquired data becomes available to the MATLAB workspace and what is meant by "acquiring data."

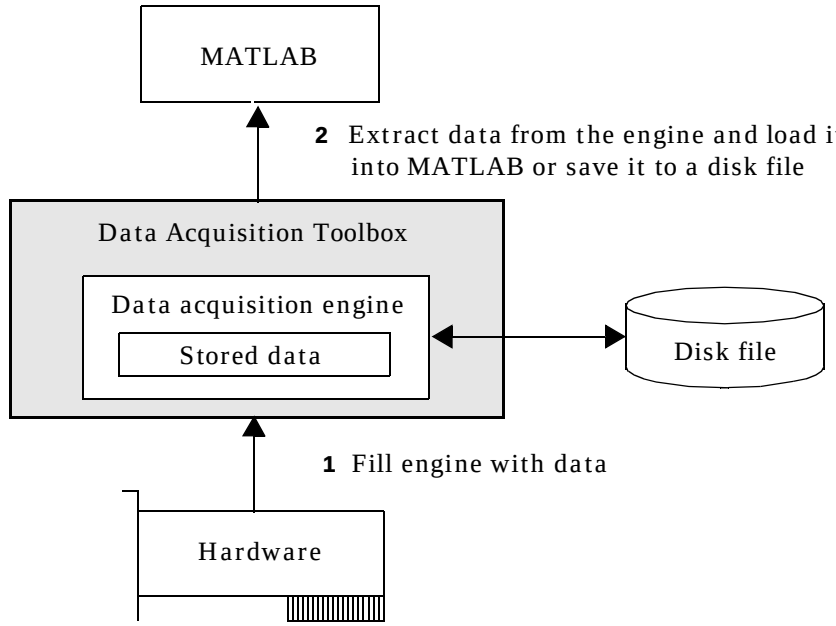
Data Flow

In a practical sense, 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 allocated memory, the acquisition rate, and the number of hardware channels from which you acquire data. However, the stored data is not automatically available in the MATLAB workspace. You must explicitly extract it from the engine in a timely way so that no data is lost.

To summarize, the flow of acquired data consists of two independent steps:

1. Data flows from the hardware to the engine
2. Data flows from the engine to MATLAB and/or to a disk file

These two steps are illustrated below.



Extracting data from the engine is described in "Managing Acquired Data" on page 4-21.

Saving data to a disk file and loading data from a disk file are described in "Logging Information to Disk" on page 4-62.

The Hardware Driver Adaptor

The hardware driver adaptor (or just *adaptor*) is the interface between the data acquisition engine and the hardware driver. The adaptor allows you to:

- Pass information from the engine to the hardware driver
- Pass information from the hardware driver to the engine. In particular, the adaptor informs the engine of any hardware-specific properties. If hardware-specific properties exist, then the properties (and their default values) are incorporated into the engine.

Supported Toolbox Adaptors

The Data Acquisition Toolbox provides adaptors for sound cards and National Instruments E-series hardware. The supported devices and adaptor name used by the toolbox are listed below.

Table 2-1: Supported Toolbox Devices and Adaptor Names

Device	Adaptor Name
Sound cards	`winsound`
National Instruments	`nidaq`

Displaying Installed Adaptors

You can find out what hardware adaptors are installed on your system with the `daqhwinfo` function.

```
hwinfo = daqhwinfo;
hwinfo.InstalledAdaptors
ans =
    'winsound'
    'nidaq'
```

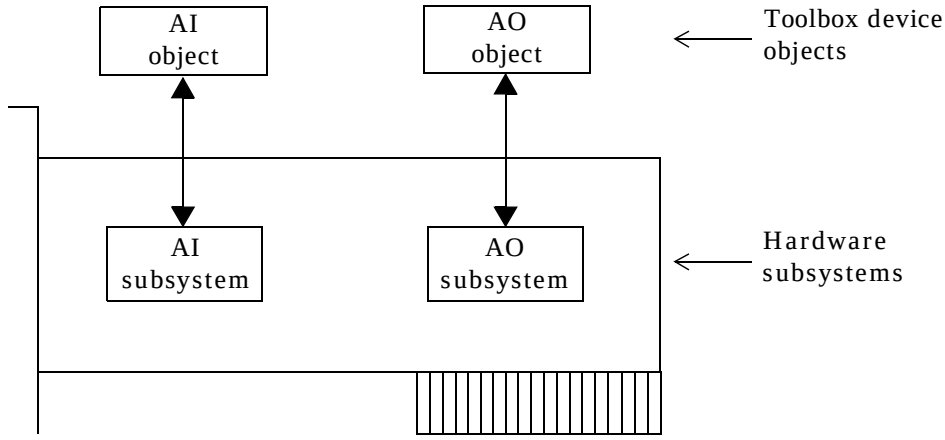
In this example, both sound card and National Instruments adaptors are installed.

Device Objects, Channels, and Lines

Data acquisition *device objects* are the basic toolbox elements you use to access your hardware device. *Channels* and *lines* are the basic hardware device elements with which data acquisition tasks are performed.

Device Objects

Device objects provide a gateway to the functionality of your hardware, and allow you to control the behavior of your acquisition. Each device object is associated with a specific hardware subsystem. The device objects supported by the Data Acquisition Toolbox allow you to access analog input (AI), analog output (AO), and digital I/O (DIO) subsystems. The association between device objects and hardware subsystems is shown below.



You can find out what subsystems are supported by your data acquisition hardware and how they are constructed with the `daqhwinfo` function. For example, the following command returns the constructor syntax for the susbsystems that are supported by a sound card.

```
hwinfo = daqhwinfo('winsound');
hwinfo.ObjectConstructorName(:)
ans =
    'analoginput('winsound',0)'
    'analogoutput('winsound',0)'
```

Creating a Device Object

To create a device object, you call specific M-file functions called object creation functions (or constructors). These M-files are based on MATLAB object-oriented programming capabilities, which is described in the *Using MATLAB* book. For example, to create an analog input object for a sound card, you issue the command

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

With device objects, you can control data acquisition behavior by configuring property values. For example, an analog input object contains all the necessary information for executing any supported analog input task. This object is then passed to the data acquisition engine which will, based on the contents of the object (the property values), carry out the desired task. For example, to set the sampling rate of your sound card to 8 kHz, issue the command

```
set(ai, 'SampleRate', 8000)
```

Properties and value are discussed in more detail on page 2-15 and Chapter 7, "Property Reference."

Before a data acquisition task can be performed, you must associate hardware channels or lines with the device object.

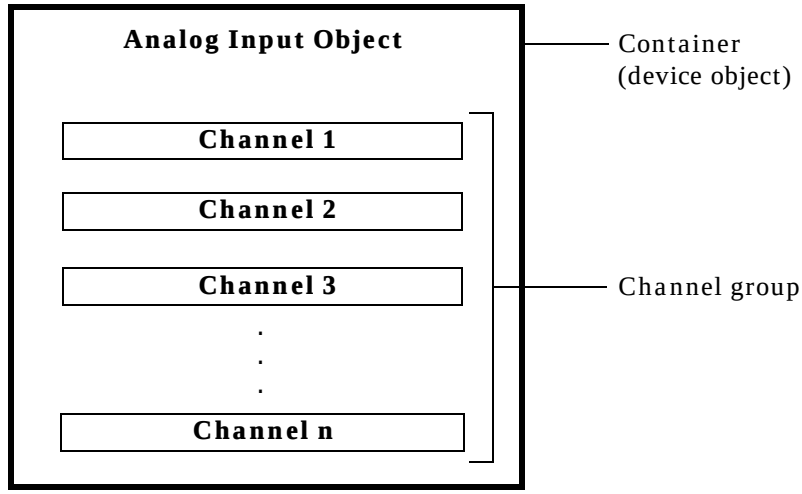
Channels and Lines

After a device object is created, you must add channels or lines to it. You add channels to analog input and analog output objects, and you add lines 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*. For example, to add two channels to the sound card object `ai`, issue the command

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

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 contained by it. For example, the sampling rate of an analog input object applies to all channels contained by that object. Therefore, the sampling rate is considered common functionality. In contrast, the channels and lines contained by the device object reflect the functionality only of a particular channel or line.

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



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

Note: You cannot perform a data acquisition task with a device object that does not contain channels or lines. Similarly, you cannot perform a data acquisition task with channels or lines that are not contained by a device object.

Toolbox Functions

The Data Acquisition Toolbox functions are M-files which allow you to access your hardware's capabilities and perform any data acquisition task. Toolbox functions can be divided into the groups shown below. For a complete description of all functions, refer to Chapter 6, "Function Reference."

Creating Device Objects

Each device object has a corresponding creation function (or *constructor*), named for the device object it creates. The toolbox creation functions are shown below.

Function	Description
`analoginput`	Create an analog input object.
`analogoutput`	Create an analog output object.

Configuring Device Objects, Channels and Lines

After a device object is created, it must configured. Configuring a device object requires adding channels or lines and setting property values to establish acquisition behavior. The toolbox configuration functions are shown below.

Function	Description
`addchannel`	Add hardware channels to an analog input or analog output object.
`addline`	Add hardware lines to a digital I/O object.
`get`	Return property values.
`set`	Configure or display property values.
`setverify`	Set and return the specified property value.

Executing the Device Object

Executing the device object involves functions that start, stop, and trigger your acquisition. The toolbox execution functions are shown below

Function	Description
`start`	Start the execution of a device object.
`stop`	Stop the execution of a device object.
`trigger`	Manually execute a trigger.

Working with Data

With any data acquisition application, you must send data to your hardware or receive data from your hardware. The toolbox functions that allow you to work with data are shown below.

Function	Description
`flushdata`	Remove data from the data acquisition engine.
`getdata`	Return data, time, and event information to the MATLAB workspace for an analog input object.
`getsample`	Return one sample for each analog input channel group member or digital I/O line group member.
`peekdata`	Immediately return data to the MATLAB workspace for an analog input object.
`putdata`	Queues data in the engine for eventual output to the D/A hardware.
`putsample`	Send one sample for each analog output channel group member or digital I/O line group member.

Getting Information and Help

Functions that provide information or help about installed hardware, driver adaptors, device objects, channels, lines, or properties are shown below.

Functions	Description
`daqhelp`	Display help for device objects, constructors, functions, and properties.
`daqhwinfo`	Display data acquisition hardware information.
`nidaq`	Display information about National Instruments adaptor
`propinfo`	Return property information for device objects, channels, or lines.
`winsound`	Display information for sound card adaptor

General Purpose

The remaining toolbox functions are shown below. Several of these functions are built-in MATLAB functions that have been overloaded (modified) for use with Data Acquisition Toolbox objects.

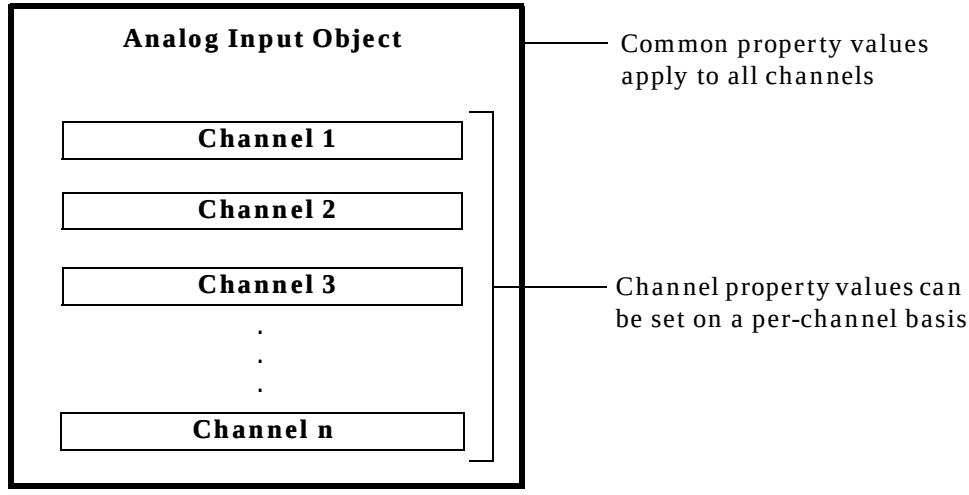
Function	Description
`clear`	Clear device objects from the MATLAB workspace.
`daqaction`	Display event information for specified event
`daqfind`	Return device objects, channels, or lines from the data acquisition engine to the workspace.
`daqread`	Read a Data Acquisition Toolbox (`.daq`) file.
`daqmem`	Allocate or display memory for one or more device objects.
`daqregister`	Register or unregister a Data Acquisition Toolbox adaptor.
`daqreset`	Remove data acquisition objects and `.dll` files from memory.

Function	Description
delete	Delete device objects, channels, or lines.
disp	Display device object, channel, and line information.
ischannel	Check for channels.
isline	Check for lines
isvalid	Check for valid device objects, channels, or lines.
length	Determine length of data acquisition objects.
load	Load device objects into the MATLAB workspace
makenames	Generate a list of channel or line names.
obj2code	Convert device objects, channels, or lines to MATLAB code and save to disk.
save	Save device objects to a MAT-file.
size	Determine size of data acquisition objects.

Property Names and Property Values

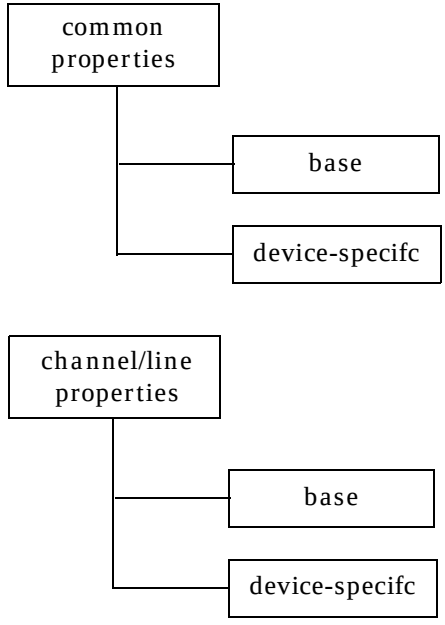
Property names and property values allow you to define and evaluate the behavior of your data acquisition application. You define acquisition behavior by assigning values to properties, and you can evaluate your configuration by displaying values for properties.

Data Acquisition Toolbox properties are divided into two main categories: common properties and channel/line properties. Common properties apply to every channel or line contained by a device object, while channel/line properties can be configured for individual channels or lines. The relationship between an analog input object, the channels it contains, and their properties is shown below.



Property values apply uniquely to a particular instance of a device object; setting a value for one object, channel, or line does not change this value for other device objects of the same type. If an invalid property value is specified, an error message is returned along with the invalid property name. All property values are verified that they are within the range of valid values when they are set. An error is returned if the value is out of range during a set.

Both common properties and channel/line properties can be subdivided into base properties and device-specific properties as shown below.



Common, channel/line, base, and device-specific properties are discussed below. For a complete description of all properties, refer to Chapter 7, “Property Reference.”

Displaying Common Properties

Common properties apply to every channel or line contained by the device object. You can display all configurable common property names and their possible values for the device object `obj` with the `set` command.

```
set(obj)
```

You can display the current value for each common property for the device object `obj` with the `get` command.

```
get(obj)
```

Common Base Properties

Common base properties apply 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.

When you display properties to the screen using `get` or `set`, the base properties are listed alphabetically before the device-specific properties.

Common Device-Specific Properties

Common device-specific properties apply only for specific hardware devices.

When you display properties to the screen using `get` or `set`, the common device-specifc properties (if they exist) are listed alphabetically after the base properties.

Displaying Channel/Line Properties

Channel/line properties can be configured for individual channels or lines contained by a device object. For example, the `InputRange` property can be set to a different value for each channel contained by an analog input object. You can display all configurable channel/line property names for the device object `obj` with the `set` command.

```
set(obj.Channel)
set(obj.Line)
```

You can display the current value for each channel/line property for the device object `obj` with the `get` command.

```
get(obj.Channel)
get(obj.Line)
```

Channel/Line Base Properties

Channel/line base properties apply to all supported hardware subsystems of a given type (analog input, analog output, etc.). For example, the `ChannelName` property exists for all analog input subsystems regardless of the vendor.

When you display properties to the screen using `get` or `set`, the base properties are alphabetically listed before the device-specific properties.

Channel/Line Device-Specific Properties

Channel/line device-specific properties apply only to specific hardware devices. When you display properties to the screen using `get` or `set`, the device-specifc properties (if they exist) are alphabetically listed after the base properties

Base Properties with Device-Specific Values

Some base properties have device-specific values. For example, National Instruments hardware supports three additional values for the base property `TriggerType`: `HwAnalogChannel`, `HwAnalogPin`, and `HwDigital`. Base properties with device-specific values are not device-specific properties.

Device-specific properties are presented in Appendix A. Device-specific values for base properties are included in Chapter 7, “Property Reference.”

Specifying Property Names

By convention, the first letter of each word that makes up a property name is capitalized. While this makes property names easier to read, MATLAB does not check for uppercase letters. In addition, you need use only enough letters to identify the property name uniquely, so you can abbreviate most property names. For example, you can set the `SampleRate` property for the device object `AI` any of these ways.

```
set(AI, 'SampleRate', 8000)
set(AI, 'samplerate', 8000)
set(AI, 'sampler', 8000)
```

In M-files, however, using the full property name can prevent problems with future releases of the Data Acquisition Toolbox if a shortened name is no longer unique because of the addition of new properties.

Default Property Values

Whenever you do not explicitly define a value for a property, then the default value is used. All properties have default values. However, the default values defined by the Data Acquisition Toolbox may 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.

Events and Actions

In general, data acquisition tasks are based on *events.* An event occurs at a specific time after a condition is met. Some of the event types supported by the Data Acquisition Toolbox include errors, triggers, action events, and start and stop events. Some of these events are described below.

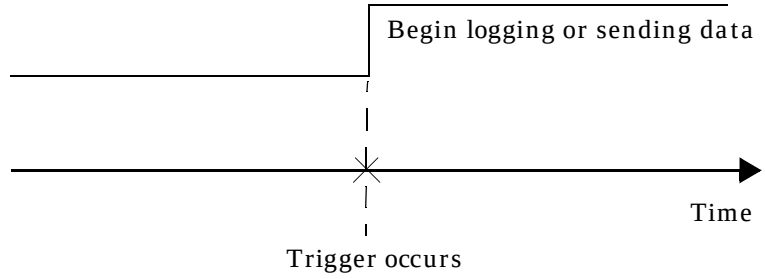
Events may result in one or more actions. For example, all the event types mentioned above have the action of calling an associated property, which in turn can execute an M-file function that you specify.

Errors

Errors have the action of stopping your acquisition. When an error occurs, an error message is displayed. The toolbox may display vendor-specific error messages. Vendor-specific errors are displayed when there is a hardware or hardware driver error.

Triggers

The definition of a trigger depends on the device object. For analog input objects, a trigger is defined as an event with the action of initiating data logging to memory and/or a disk file. For an analog output object, a trigger is defined as an event with the action of sending queued data to the D/A converter. This is illustrated in the figure below.



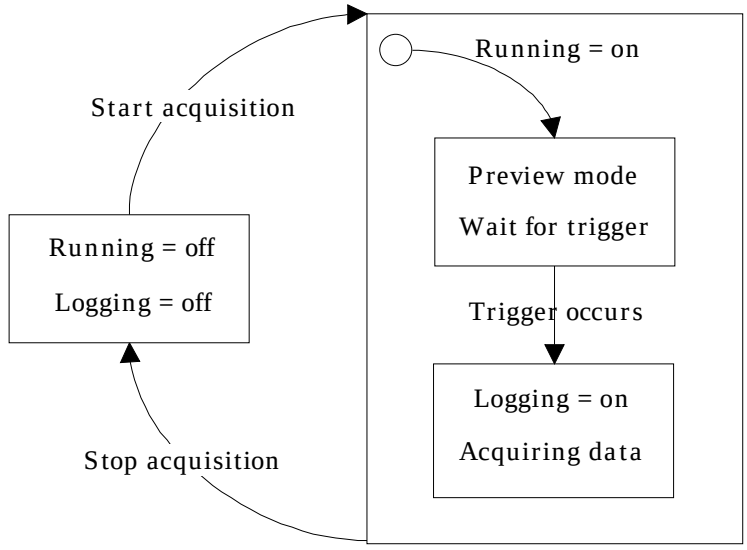
An important concept in the Data Acquisition Toolbox for analog input objects is the number of samples to acquire for each channel group member per trigger. This number is used to specify how much data you want to store in the engine and is described in detail in “Configuring Triggers” in Chapter 4.

State Transitions

At any time, your data acquisition application can be thought of as being in a particular *state*. The state of an object depends on events that may have occurred. Two states are defined for the Data Acquisition Toolbox.

- Running
- Logging (for analog input objects), or sending for analog output objects.

The running state is given by the `Running` property, while the logging (sending) state is given by the `Logging` (`Sending`) property. These data acquisition states and their transition paths are shown for an analog input acquisition in the figure below.



A state transition occurs when the data acquisition process transitions from one state to another due to property values that have changed. For example, a state transition occurs when your acquisition transitions from the “running” state to the “not running” state or from the “logging” state to the “not logging” state.

A state transition has the action of calling the associated action property. The value of an action property is the name of an M-file function that is to be

executed when the particular state transition occurs. Action properties provide you with tremendous flexibility in defining the capabilities of your acquisition. For example, you can specify an M-file that performs a fast Fourier transform (FFT) every time a predefined amount of data is acquired.

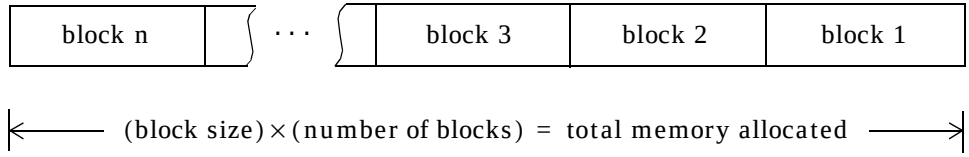
State diagrams like the one shown on page 2-20 are used throughout this book to help you visualize how data acquisition states can change based on the property values that are set. State transitions and events are described in detail in “Configuring Events and Actions” in Chapter 4.

The Data Block

When data is acquired, it must be temporarily stored in computer memory. The memory required for data storage depends on several factors. For example, if you are acquiring data from an analog input device, then the required memory depends on the number of channels that data is being acquired from, the sampling rate, and the size of each sample.

The Data Acquisition Toolbox allocates memory in terms of *data blocks*. A data block is defined as the smallest "slice" of memory that can be usefully manipulated by the data acquisition engine. For example, acquired data is logged to a disk file using an integral number of data blocks.

The total memory allocated is given by the product of two variables: the data block size and the number of data blocks. The total allocated memory is often referred to as a *buffer*. A representation of allocated memory using several data blocks is shown below where block 1 is the first allocated block and block n is the last allocated block.



The Data Acquisition Toolbox strives to make memory allocation as simple as possible. For this reason, the default data block size and number of blocks are automatically calculated by the engine. This calculation is based on the parameters of your acquisition and is meant to apply to most common data acquisition applications. Additionally, as data data is acquired, the number of blocks dynamically increases such that sufficient total memory is allocated to (at least) store the data to be acquired per trigger. 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 is set to infinity.
- You acquire data at the limits of your hardware, your computer, or the toolbox. Specifically, acquisitions at very high sampling rates must be

considered very carefully with respect to memory allocation to guarantee that samples are not lost.

You are free to override the memory allocation policy used by the engine and manually change the block size and number of blocks at ant time. However, you should do so only after careful consideration since system performance may be adversely affected, which may result in lost data.

Acquired data that is stored to memory must be extracted in a timely way. If not, the data is overwritten and permanently lost. Managing acquired data is described in Chapter 4, “Managing Acquired Data.”

The Data Acquisition Session

The data acquisition *session* encompasses all actions you must take to perform the necessary tasks for your data acquisition application. A typical data acquisition session consists of these steps:

1 Initialization

The device object is created and linked to a specific hardware device.

2 Configuration

Channels or lines are added to the device object, and property values are set to establish the desired behavior.

3 Execution

The device object is started and it runs according to the previously set properties. At this time you can preview data, acquire data from an A/D subsystem, or send data to a D/A subsystem.

4 Termination

The object is no longer needed and is deleted.

The example code below shows the basic steps you will take during a data acquisition session using an analog input device.

daqdoc2_1.m

Initialization: Create the analog input object `AI` for a sound card. The installed adaptors and hardware ID's are found with `daqhwinfo`.

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

Configuration: Add two channels to AI, define a two second acquisition, and assign values for the basic setup properties. The actual sampling rate is retrieved since it may be set to a value that differs from the specified value.

```
addchannel(AI, 1:2);
%addchannel(AI, 0:1);
duration = 2;
set(AI, 'TriggerType', 'Immediate');
set(AI, 'SampleRate', 8000);
ActualRate = get(AI, 'SampleRate');
set(AI, 'SamplesPerTrigger', duration*ActualRate);
```

Execution: Start AI, wait until AI stops running, and extract all data from the engine. Before start is issued, you may want to begin inputting data for a microphone or a CD player.

```
start(AI)
data = getdata(AI);
```

Termination: Plot the data and delete AI.

```
plot(data)
xlabel('Samples')
ylabel('Signal (volts)')
title('daqdoc2\_1')
delete(AI)
```

daqdoc2_1.m

Initialization, configuration, execution, and termination are discussed in more detail below.

Initialization

Before a hardware device can carry out a data acquisition task, it must be initialized to a known state. Initialization occurs just after a device object is created.

```
AI = analoginput('winsound')
```

After device object creation, the toolbox automatically performs these steps:

- The hardware driver is loaded into memory. Vendor-provided calls are used to initialize the hardware to a default state. This default state is typically published by the vendor.
- The hardware driver adaptor is loaded into memory.
- Hardware-specific properties and their default values are assigned to the device object.

Note: To initialize a device object, all you need to do is create the device object. The above steps are carried out automatically by the toolbox after device object creation.

Configuration

After initialization, a hardware device must be configured before a task can be executed. This essentially means setting up the device with the information necessary to carry out that task. In general, each type of device requires some configuration information that is specific to that device.

For example, typical configuration information for a National Instruments board might be to collect two seconds of data from analog input hardware channels 0, 1, and 2 at a sampling rate of 10 kHz using dual channel DMA and single-ended inputs. The associated code for this configuration is shown below.

```
AI = analoginput('nidaq', 1);
set(AI, 'SampleRate', 10000);
set(AI, 'InputType', 'SingleEnded');
set(AI, 'SamplesPerTrigger', 20000);
set(AI, 'TransferMode', 'DualDMA');
addchannel(AI, 0:2);
```

Any property value that is not explicitly set results in the default value for that property being used.

With the Data Acquisition Toolbox, configuration consists of two basic steps: mapping the hardware to a data acquisition object, and setting property values to establish the behavior of your acquisition. These steps are discussed below.

Mapping Hardware to Objects

To perform any data acquisition task, device objects must contain channels or lines. The resulting channel or line group consists of a mapping from hardware channels or lines to corresponding MATLAB indices. This mapping is done automatically when hardware channels or lines are added to a device object. For example, if hardware channels 0, 1, and 2 are added to the analog input object `AI`

```
addchannel(AI, 0:2)
```

they are automatically assigned the corresponding MATLAB indices of 1, 2, and 3, respectively.

When referencing channels or lines, it is the MATLAB indices that are used and not the hardware identifiers. Given this, you should keep in mind that MATLAB is “1-based” (indices begin at 1), while some hardware is “0-based” (hardware channels begin at 0).

The hardware channel or line assigned to a specific index depends on the hardware assignment order. For example, you can assign channels such that the first index corresponds to the first hardware channel, the second index corresponds to the fifth hardware channel, etc. If you are using scanning hardware, then this mapping defines the scan order.

Setting Property Values

You can configure an existing device object by setting its property values. As described on page 2-15, all device objects, channels, and lines have a base set of properties that can be set. In addition to the base set of properties, there may be a set of device-specific properties, which is determined by the hardware you are using. You can use the `set` command to return all properties for `AI` and their possible values.

```
set(AI)
```

Property values can be set using property name/property value (PV) pairs.

```
set(AI, 'SampleRate', 10000);
set(AI, 'SamplesPerTrigger', 20000, 'InputType', 'SingleEnded');
```

In addition to using the set command to assign values to properties, you can use the standard MATLAB subscripted assignment syntax.

```
AI.SampleRate = 10000;
AI.SamplesPerTrigger = 20000;
AI.InputType = 'SingleEnded';
```

The advantage of the set syntax is that you can assign multiple property values with one call to set whereas with subscripted assignment, you can assign only one property value at a time.

Detailed procedures for configuring Data Acquisition Toolbox objects are given in Chapter 4, "Doing More With Analog Input."

Getting Property Values

You can evaluate the current object configuration by getting the property values. For example, you can use the get command to return all the current property names and property values for AI.

```
get(AI)
```

You can also return specific property values.

```
get(AI, 'SampleRate')
get(AI, {'SamplesPerTrigger', 'InputType'})
```

In addition to using the get command to return property values, you can use the standard MATLAB subscripted reference syntax.

```
AI.SampleRate
AI.SamplesPerTrigger
AI.InputType
```

For either syntax, if a left-hand-side variable is defined, then the returned values are captured in that variable.

```
rate = get(AI, 'SampleRate')
type = AI.InputType
```

The advantage of the get syntax is you can return multiple property values with one command whereas with subscripted reference, you can return only one property value at a time.

Execution

Execution is divided into three main categories:

- Starting
- Previewing, acquiring, or sending data
- Stopping

These categories are discussed below.

Starting

You start a device object with the `start` command

```
start(AI)
```

When a `start` command is issued, the Data Acquisition Toolbox enters the *running* state. This means that both the hardware device and data acquisition engine are executing according to the configured properties. Once a trigger occurs, data logging to memory and/or a disk file is initiated for an analog input object, while data output is initiated for analog output objects.

Previewing, Acquiring, or Sending Data

While the device object and data acquisition engine are running, you can preview, acquire (for analog input), or send (for analog output) data. These topics are discussed below.

Preview or Acquire Data. For analog input objects, data can be previewed 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

```
data = peekdata(AI, 500);
```

Since previewing data is usually not a high priority task, `peekdata` does not guarantee that all requested data is returned. You can preview data any time after the trigger occurs as long as the object is running.

Data can also be extracted from the data acquisition engine with the `getdata` function. For example, to extract 500 samples from the engine

```
data = getdata(AI, 500);
```

getdata blocks execution control until all the requested data is returned. You can extract data any time after the trigger occurs.

Send Data. For analog output objects, data can be queued in the engine for output to an analog output (D/A) hardware device with the putdata function. For example, to queue 500 samples to the engine

```
data = sin(0:0.01:2*pi)';
putdata(AO, data(1:500))
```

You can queue data either before or after start is issued. Queued data can be output only after the trigger occurs.

Stopping

A device object can stop under one of these conditions:

- A stop command is issued
- The requested number of samples to acquire (analog input) or output (analog output) is reached
- A runtime error occurred

  Runtime errors can include hardware errors and timeouts. A timeout can occur if the data requested from the engine is not extracted in a predetermined amount of time.

When the device object stops, both the hardware device and data acquisition engine leave the "running" state and enter the "not running" state. At this point you can reconfigure your acquisition, or immediately issue another start command using the current configuration.

Termination

When your data acquisition tasks are complete, the device object(s) should be deleted to free memory and other physical resources. You can use the delete function to delete a device object.

```
delete(AI);
```

The delete function removes the specified device object from the engine and the MATLAB workspace. If this was the last object accessing the associated hardware, the associated hardware driver and hardware driver adaptor are closed and unloaded. You can only delete an object if it is not running.

3

Getting Started With Analog Input

Overview

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

The functionality related to the analog input device allows you to process an analog input signal and convert it to a digital representation for processing within the computer. The 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 basic analog input data acquisition tasks. These tasks include:

- Connecting the Toolbox to your hardware
- Setting core property values
- Acquiring data
- Evaluating the status of your acquisition

Understanding these basic tasks will go a long way towards helping you understand the functionality and capabilities of the Data Acquisition Toolbox. In presenting these tasks, only a core set of properties and functions are used. If at any time you want more information about any property or function, you can refer to the appropriate *User's Guide* reference section or the command line help.

Connecting to Your Hardware

Before data can be acquired, you must "connect" the Data Acquisition Toolbox to your hardware. Connecting to your hardware requires three actions:

- Registering the hardware driver adaptor
- Creating an analog input object
- Adding channels to the analog input object

These two topics are described below.

Registering the Hardware Driver Adaptor

When the Data Acquisition Toolbox is installed, the installed hardware driver adaptors are automatically registered. Adaptor registration involves the data acquisition engine locating all hardware driver adaptors so that it can make use of their services.

If for some reason automatic adaptor registration fails, you must manually register the adaptors. A hardware driver adaptor is manually registered with the command

```
daqregister('adaptor');
```

where `adaptor` is one of the supported hardware driver adaptors listed in Table 3-1. Hardware driver adaptors need to be registered only once after the Data Acquisition Toolbox is installed.

You can find out what hardware adaptors are installed on your system with the `daqhwinfo` function.

```
hwinfo = daqhwinfo;
adaptors = hwinfo.InstalledAdaptors
adaptors =
    'winsound'
    'nidaq'
```

Creating an Analog Input Object

Before executing an analog input (AI) task, an analog input object must be created. You create an AI object with the command

```
AI = analoginput('adaptor', ID);
```

where:

- `analoginput` is the analog input object creation function
- `adaptor` is the hardware driver adaptor name
- `ID` is the vendor-specific device label
- `AI` is the analog input object

The valid device object creation function arguments are given below.

Table 3-1: Analog Input Object Creation Function Argument Values

Value	Description	Supported Drivers		
		Sound Cards	NI	HP VXI
`adaptor`	Adaptor name	`winsound`	`nidaq`	`hpvxi`
`ID`	Device label	N/A	Device number	Device id

If the hardware driver adaptor has been successfully registered by the data acquisition engine, then an analog input object can be created. Once the device object is created, then:

- If this is the first device object to reference a particular driver, that driver is loaded into memory and initialized.
- The device object is linked to the specified hardware device through the hardware driver.
- A unique analog input object name is automatically created and assigned to the `Name` property. The name is created by concatenating the name of the adaptor, the device label, and the device object type. You can change the value of `Name` at any time. For sound cards, only the name of the adaptor and the device object type is used.

By default, no hardware channels are associated with a newly created analog input object. To associate channels with the device object, the `addchannel` function must be used.

Example: Create an Analog Input Object

Suppose you want to create the analog input object `AI` for a sound card. Since there is no device label for the sound card, the analog input object creation function requires only the adaptor name as an argument.

```
AI = analoginput('winsound');
```

An analog input object called `AI` now exists in the MATLAB workspace and the engine. The device object's name can now be displayed.

```
get(AI,'Name')
ans =
winsound0-AI
```

If you are familiar with Handle Graphics®, then `AI` may look like a handle to you. In fact, it is not a handle but a Data Acquisition Toolbox object. To verify this, type `whos AI` at the command line.

```
whos AI
  Name      Size         Bytes  Class
  AI        1x1           1332  analoginput object
Grand total is 52 elements using 1332 bytes
```

`AI` cannot yet be used to acquire data since it has no hardware channels associated with it. Additionally, property values must be set to establish the required behavior for your specific application.

Adding Channels to an Analog Input Object

After creating the analog input object, you must now add channels to it. As shown in the figure on page 2-10, a device object can be thought of as a container for channels. Adding channels to the analog input object is accomplished with the `addchannel` function.

```
chans = addchannel(object, hwch, index, names)
```

where:

- `object` is the analog input object
- `hwch` are the identifiers (ID's) of the hardware channels you are adding to the analog input object. Any valid MATLAB vector syntax can be used to specify the hardware channels.
- `index` (optional) are the MATLAB indices of the hardware channels.

- names (optional) are the descriptive names of the hardware channels.
- chans (optional) is a vector containing the analog input channels.

Note: Adding channels to a sound card involves certain device-specific restrictions. These restrictions are explained in the example on page 3-7.

The collection of channels contained by the device object is referred to as a *channel group*. Channel group members must all reside on the same hardware device and the channels are all sampled at the same rate. It is important to remember that when you add channels to an object, you are mapping physical hardware channels on your analog input hardware device to MATLAB indices. This mapping allows a certain amount of flexibility in terms of ordering the channels as described by these rules:

- Hardware channels can be assigned to multiple analog input objects.
- Hardware channels can be assigned multiple times to the same analog input object.
- Channel indices start at “1” and increase monotonically up to the number of channel group members.
- Hardware indices begin at 0 for National Instruments hardware, whereas for sound cards and Hewlett-Packard hardware, they begin at 1.

If no indices are specified with addchannel, then they are automatically defined such that they are monotonically increasing and contains no gaps. Each channel is referenced by its position in the channel group (its index), or optionally by its name if one is defined. Named channels are discussed in “Reference by Channel Name” on page 4-8. For detailed information about addchannel, refer to Chapter 6, “Function Reference.”

If you are using scanning hardware, the channel indices specify the order in which the hardware channels are sampled. The first hardware channel to be sampled has index "1", the second hardware channel to be sampled has index "2", and so on. You can change the channel scanning order by specifying the MATLAB indices explicitly with the index input argument. Channel reordering is described in “Reordering Channels” on page 4-10.

Example: Adding Channels for National Instruments Hardware

Suppose you create the analog input object `ai` for a National Instruments board having a device ID of 1.

```
ai = analoginput('nidaq', 1)
```

To perform any data acquisition task, you must add hardware channels to the device object. Many National Instruments E-series boards have 16 single-ended input channels that can be added. To add the first 8 channels to `ai`:

```
set(ai, 'InputType', 'SingleEnded')
addchannel(ai, 0:7)
```

These channels are automatically assigned the MATLAB indices 1-8. You can now add the remaining 8 hardware channels to `ai`.

```
addchannel(ai, 8:15)
```

These channels are automatically assigned the MATLAB indices 9-16. You can also use one call to `addchannel` to add all 16 channels.

```
addchannel(ai, 0:15)
```

The channel scanning order is given by the MATLAB indices. Therefore, hardware channel 0 is sampled first, hardware channel 1 is sampled second, and so on.

Example: Adding Channels for a Sound Card

Suppose you create the analog input object `ai` for a sound card.

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

To perform any data acquisition task, you must add hardware channels to the device object. Most sound cards have just two hardware channels that can be accessed. 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`, then the sound card is in mono mode. Additionally, the channel added must be channel 1.

```
addchannel(ai, 1)
```

At the toolbox level, a mono acquisition means that data is acquired from channel 1. Channel 1 is called the `Mono` channel. At the hardware level, you generally cannot determine the actual channel configuration and data can be acquired from channel 1, channel 2, or both depending on your sound card.

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

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

You can also use one call to `addchannel`.

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

In the latter case, channel 1 must be specified as the first element of the hardware ID vector.

A stereo acquisition means that data is acquired from both hardware channels. Channel 1 is called the `Left` channel and channel 2 is called the `Right` channel. While in stereo mode, if you want to delete a channel, then that channel must be channel 2. If you try to delete channel 1, an error is returned. For any other type of data acquisition device, you may add or delete channels in any order.

Controlling Acquisition Behavior with Properties

After the analog input object is created and hardware channels are added, you can assign values to properties. You can think of a property as a characteristic of a device object that governs its behavior. In a more practical sense, a property is anything that can be configured using the `set` command or subscripted assignment.

The properties that need to be configured depend on your particular data acquisition application. However, for many common analog input applications, there is a core group of properties that must be set. This core group is divided into two subgroups:

- Properties related to the basic setup
- Properties related to the data range and engineering units

These subgroups are described below.

Basic Setup Properties

For most data acquisition applications, there is a set of properties used to configure the basic setup for your session. These properties provide access to individual channels and control the sampling rate, triggers, and the number of samples to acquire per trigger. Properties related to the basic setup are given below.

Table 3-2: Analog Input Basic Setup Properties

Property	Description
`Channel`	Contains all the channels associated with the device object as created with `addchannel`
`SamplesPerTrigger`	Specifies the number of samples to acquire per channel for each trigger issued
`SampleRate`	Specifies the samples per second of the analog input hardware device
`TriggerType`	Specifies the type of trigger to be issued

Accessing Individual Channels

As discussed in "Property Names and Property Values" on page 2-15, the 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 can be configured for individual channels.

`Channel` contains all the channels associated with an analog input object as created with `addchannel`. Using `Channel`, you can:

- Configure property values for individual channels
- Display the possible values for each channel property with the `set` command
- Display the current value for each channel property with the `get` command

Using `Channel` to configure channel properties is illustrated in "Data Range and Engineering Units Properties" on page 3-12.

Setting the Sampling Rate

The per-channel rate at which data is acquired is controlled with the `SampleRate` property. `SampleRate` must be specified as samples per second. For example, to set the sampling rate of your sound card to 44,100 samples per second (44.1 kHz)

```
set(ai, 'SampleRate', 44100);
```

Data acquisition boards typically have predefined sampling rates that can be set. If you specify a sampling rate that does not match one of these predefined values but is within the range of valid values, then the toolbox automatically selects a valid sampling rate. If you specify a sampling rate that is outside the range of valid values, then an error is returned. The rules applied by the toolbox to select a valid sampling rate are described in Chapter 7, "Property Reference."

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

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

The default value for `SampleRate` is 8 kHz for most sound cards. For other hardware devices, the default value is supplied by the hardware driver.

Defining a Trigger

For analog input objects, a trigger is defined as an event that initiates data logging to memory and/or a disk file.

The `TriggerType` property specifies the type of trigger to be issued. The valid `TriggerType` values that are supported for all hardware devices are given below.

Table 3-3: TriggerType Values

Value	Description
`{Immediate}`	The trigger occurs just after the `start` command is issued. This is the default trigger type.
`Manual`	The trigger occurs just after you manually issue the `trigger` function.
`Software`	You must specify one or more channels as trigger sources. The trigger is issued when a signal satisfying the specified condition is detected on one of those channels.

For some hardware, additional trigger types are available. The value of `TriggerType` determines whether additional trigger properties must be set.

Triggering can be a complicated issue and it has many associated properties. For detailed information about triggering, refer to “Configuring Triggers” on page 4-30.

Setting the Samples to Acquire per Trigger

When a trigger executes, the number of samples to be acquired per trigger specified is acquired for each channel group member and stored in the data acquisition engine and/or written to a log file. The number of samples to acquire per trigger is configured with the `SamplesPerTrigger` property.

```
set(ai, 'SamplesPerTrigger', 8000);
```

The default value of `SamplesPerTrigger` is calculated by the engine such that one second of data is collected, and is based on the value of `SampleRate`. In general, to calculate the acquisition time for each trigger issued, you apply the formula

acquisition time = `SamplesPerTrigger/SampleRate`

Data Range and Engineering Units Properties

This section describes properties that control the range of data expected from your sensor, the range of the A/D converter, and the engineering units of the data returned to MATLAB.

When data is acquired through a sensor and digitized by an A/D converter, you must be aware of these three 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 hardware. For many devices, the A/D hardware range is specified by the gain and polarity.
- By default, most A/D hardware outputs data in units of volts. However, after the data is digitized, you may want to define a linear scaling that represents specific engineering units when data is returned to MATLAB.

The sensor and A/D ranges should be selected such that the maximum precision is obtained and the full dynamic range of the source signal is covered. The default A/D range is supplied by the hardware driver.

The sensor range is set with the `SensorRange` property, the A/D range is set with the `InputRange` property, and the engineering units are set with the `UnitsRange` property. These are all channel properties and can be set on a per-channel basis.

For example, suppose you are using a microphone that can measure sound levels up to 120 dB and can produce an output voltage ranging from -1 to 1 volts. If you are measuring street noise in your application, then you might expect that the sound level never exceeds 60 dB. Assuming a linear relationship between the sound level and sensor output voltage, then you are using only 50% of the dynamic range of your sensor (-0.5 to 0.5 volts). In addition, your A/D converter may have several valid voltage ranges that it can be set to. The best A/D hardware range is the one that encompasses the expected sensor range most closely. For this example, let's assume the

hardware only supports a range of -1 to 1 volts and that engineering units conversion is not required. The code shown below reflects this example.

```
ai = analoginput('winsound')
addchannel(ai, 1)
ch = ai.Channel(1)
set(ch, 'SensorRange', [-0.5 0.5])
set(ch, 'InputRange', [-1 1])
```

Setting the data range and engineering units is described in detail in "Engineering Unit Conversion" on page 4-59.

Acquiring Data

Acquiring data with the Data Acquisition Toolbox involves using functions that execute the analog input object, and functions that access the data stored in the engine. These functions are described below.

Starting the Acquisition

To start the device object and data acquisition engine, you must issue the `start` command.

```
start(ai);
```

Starting an acquisition means that the hardware device and data acquisition engine are both running. Running means that data is being processed by the hardware device. Running does not necessarily mean that data is being logged to the engine or a disk file. To log data, a trigger must occur. After the trigger occurs, the data can be loaded into MATLAB.

When a `start` command is issued, the `Running` property is automatically set to `On`. If the device is already running when `start` is issued, then an error is returned.

Previewing Stored Data

After starting the acquisition, you may want to preview the data stored in the data acquisition engine. You can preview data with the `peekdata` function. For example, if you want to graphically preview the most recently acquired 100 samples for each channel group member, you can use the `peekdata` and `plot` commands.

```
data = peekdata(ai, 100);
plot(data)
```

`peekdata` is a nonblocking function that returns execution control immediately to MATLAB. It does not guarantee that all data samples are acquired and processed without gaps. If the number of samples requested is greater than the number of samples in the current data block, the currently acquired samples are returned along with a warning message stating that all the requested samples were not available.

`peekdata` does not extract data from the engine. Extracting data from the engine is accomplished with the `getdata` function as described below.

Extracting Stored Data

After the trigger occurs, acquired data fills the memory allocated by the engine. Once the memory is filled, the data is overwritten. To prevent data from being overwritten, it must be extracted from the engine in a timely way. After the data is extracted, it can be processed in the MATLAB environment.

If you want to guarantee that all data samples are acquired and processed without gaps, then the `getdata` function must be used. For example, to extract the first 100 samples for each channel group member

```
data = getdata(ai, 100);
```

`Data` is an *m*-by-*n* array where *m* is the number of samples requested and *n* is the number of channels. `getdata` will not return execution control to MATLAB until the requested samples are acquired and returned to `data`. If the number of samples is not specified, then `getdata` defaults to the number specified in the `SamplesPerTrigger` property.

You can also return sample-time pairs with `getdata`.

```
[data, time] = getdata(ai, 100);
```

`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 logged by the engine. Time is measured continuously until the acquisition is stopped.

Stopping the Acquisition

A data acquisition process can stop under these conditions:

- A runtime error occurs
- The specified number of samples is acquired
- The `stop` command is issued

When the `stop` command is issued, the `Running` and `Logging` properties are automatically set to their default value `Off`.

Deleting the Device Object

When the data acquisition task is completed and the device object is no longer needed, it should be deleted to free the memory and other physical resources. You can delete a device object with the delete command.

```
delete(ai);
```

The `delete` function removes the device object from both the data acquisition engine and the MATLAB workspace. If this was the last object accessing the hardware, the associated hardware driver is closed and unloaded.

Note: You cannot delete a device object using the command form of delete: `delete object`

Analog Input Examples

This section illustrate how to perform basic data acquisition tasks using analog input subsystems and the Data Acquisition Toolbox. In doing so, many of the properties and functions presented in this chapter are used.

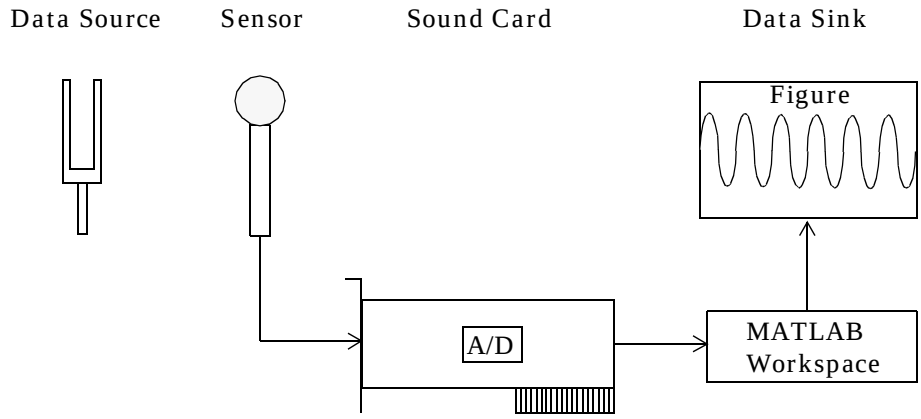
For most data acquisition applications using analog input subsystems, the same basic steps must be followed. These basic steps include:

- Installing and connecting the components of your data acquisition hardware. At a minimum, this involves connecting a sensor to a plug-in or external data acquisition device.
- Configuring your data acquisition session. This involves setting property values and using specific functions to acquire data.
- 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 an FFT on the acquired data to find the frequency components of the tuning fork. The setup for this task is shown below.



Configuring the Data Acquisition Session

As described in "The Data Acquisition Session" on page 2-24, using the Data Acquisition Toolbox to acquire data involves these steps:

- Create an analog input object
- Add hardware channels to the device object and assign values to properties
- Execute the device object
- Retrieve acquired data from the data acquisition engine

For this application, you will acquire one second of sound level data on one sound card channel. Since the tuning fork vibrates at a nominal frequency of 440 Hz, the sound card sampling rate can be set to its lowest sampling rate of 8000 Hz. Even at this lowest rate, we are guaranteed not to experience any aliasing effects since 8000 Hz is well above the expected Nyquist frequency. After you have set the tuning fork vibrating and placed it near the microphone, you will trigger the acquisition one time using a manual trigger. The complete data acquisition session for the sound card is shown below.

daqdoc3_1.m

Initialization: Create the analog input object `AI` for a sound card. The installed adaptors and hardware ID's are found with `daqhwinfo`.

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

Configuration: Add one channel to `AI`, 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 since it may be set to a value that differs from the specified value.

```
chans = addchannel(AI, 1); %chans = addchannel(AI, 0);
duration = 1; %1 second acquisition
set(AI, 'SampleRate', 8000);
ActualRate = get(AI, 'SampleRate');
set(AI, 'SamplesPerTrigger', duration*ActualRate);
set(AI, 'TriggerType', 'Manual');
blocksize = get(AI,'SamplesPerTrigger');
Fs = ActualRate;
```

Execution: Start AI, issue a manual trigger, and extract all data from the engine. Before start is issued, you should begin inputting data from the tuning fork into the sound card (whistling will work as well).

```
start(AI);
trigger(AI)
data = getdata(AI);
```

Termination: Delete AI.

```
delete(AI)
```

You should be aware that only one application can access the sound card at any time. For example, if you are using the Windows Sound Recorder and issue the start command as shown above, the toolbox returns this error.

```
??? Error using ==> daqdevice/start
Device 'Winsound' already in use.
```

You must close the Sound Recorder application and reissue start.

MATLAB Analysis

For this experiment, 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.

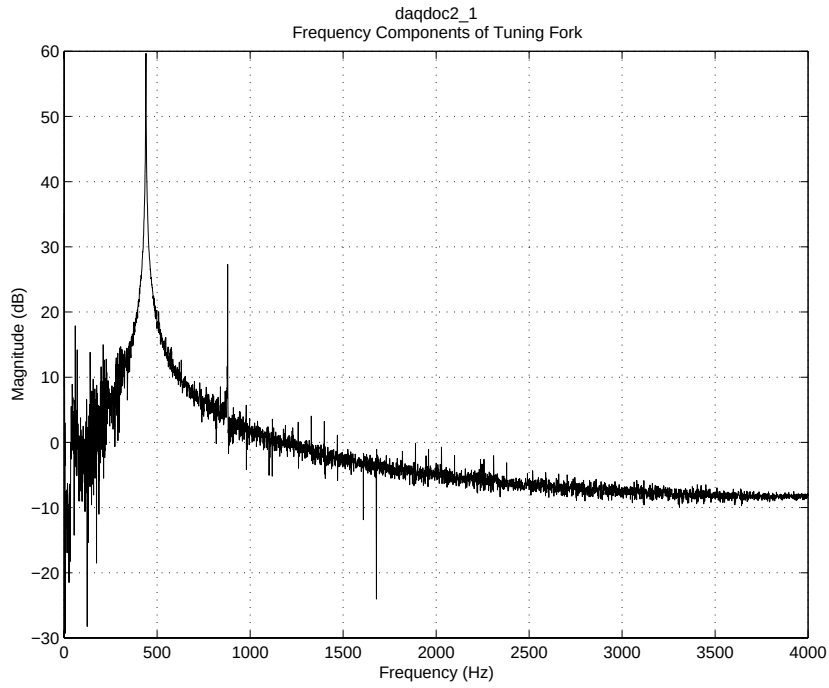
```
[f, mag] = daqdocfft(data, Fs, blocksize);
```

daqdocfft outputs the frequency and magnitude of data, which can then be plotted. daqdocfft is shown below.

```
function [f, mag] = daqdocfft(data,Fs,blocksize)
%DAQDOCFFT calculates the FFT of X using sampling frequency FS
%and the SamplesPerTrigger provided in BLOCKSIZE.
xFFT = fft(data);
xfft = abs(xFFT);
ind = find(xfft==0); % Avoid taking the log of 0;
xfft(ind) = 1e-17;
mag = 20*log10(xfft);
mag = mag(1:blocksize/2);
f = (0:length(mag)-1)*Fs/blocksize;
f = f(:);
```

The results are plotted below.

```
plot(f, mag)
grid on
ylabel('Magnitude (dB)')
xlabel('Frequency (Hz)')
title(sprintf('daqdoc3\\_1\nFrequency Components of Tuning
Fork'))
```



This plot shows the fundamental frequency around 440 Hz and the first overtone around 880 Hz. A simple way to find actual fundamental frequency is

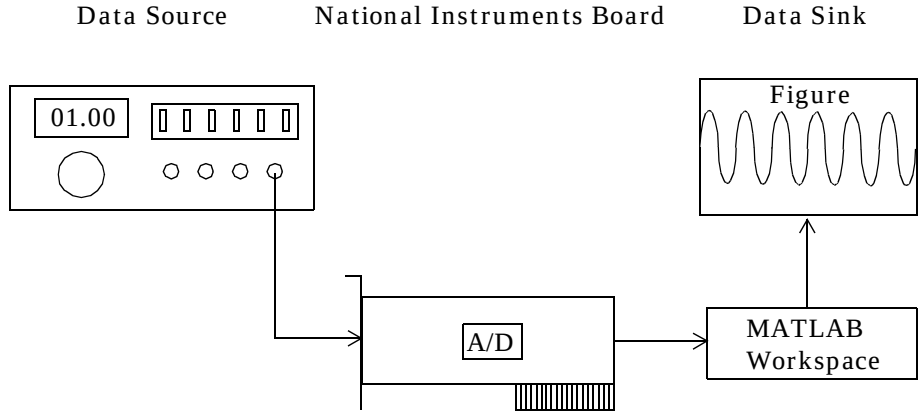
```
ymax = max(mag);
find(mag==ymax)
```

daqdoc3_1.m

The answer is 441 Hz.

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 an FFT on the acquired data to find the nominal frequency of the generated sine wave. The setup for this task is shown below



Configuring the Data Acquisition Session

As described in "The Data Acquisition Session" on page 2-24, using the Data Acquisition Toolbox to acquire data involves these steps:

- Create an analog input object
- Add hardware channels to the object
- Assign values to properties
- Execute the object
- Retrieve acquired data from the data acquisition engine

For this application, you will acquire one second of data on one input channel. The board will be set to a sampling rate of 10 kHz, which is well above the expected Nyquist frequency of the input signal. Therefore, aliasing effects should not be present. After you have connected the input signal to the board, you will trigger the acquisition one time using a manual trigger. The complete data acquisition session for the sound card is shown below.

daqdoc3_2.m

Initialization: Create the analog input object AI for a National Instruments board. The installed adaptors and hardware ID's are found with daqhwinfo.

```
AI = analoginput('nidaq', 1);
```

Configuration: Add one channel to AI, 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 since it may be set to a value that differs from the specified value.

```
chans = addchannel(AI, 0);
duration = 1; %1 second acquisition
set(AI, 'SampleRate', 10000);
ActualRate = get(AI, 'SampleRate')
set(AI, 'SamplesPerTrigger', duration*ActualRate);
set(AI, 'TriggerType', 'Manual');
% Create variables used for analysis
blocksize = get(AI,'SamplesPerTrigger');
Fs = ActualRate;
```

Execution: Start AI, issue a manual trigger, and extract all data from the engine. Before start is issued, you should begin inputting data from the function generator into the data acquisition board.

```
start(AI)
trigger(AI)
data = getdata(AI);
```

Termination: Delete AI.

```
delete(AI)
```

MATLAB Analysis

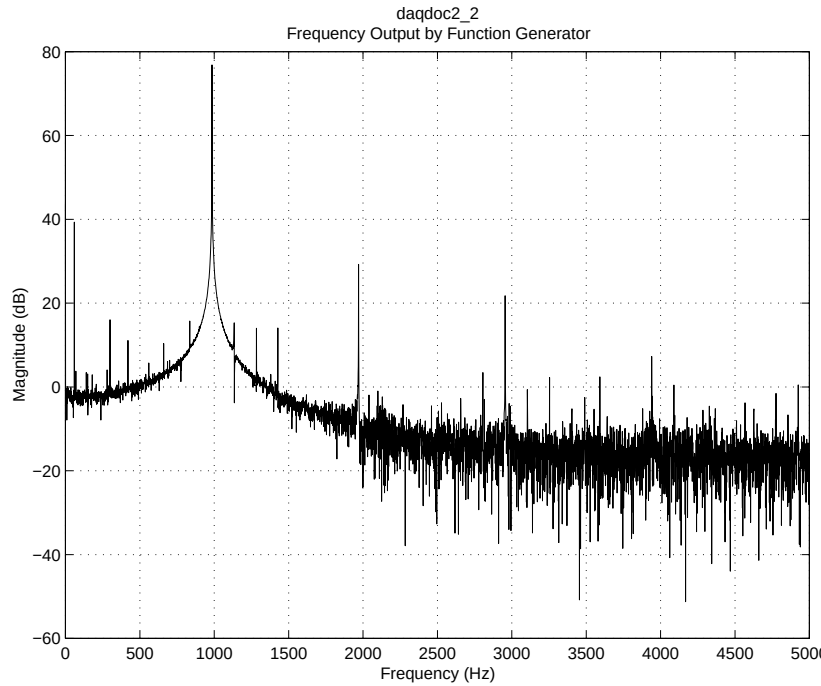
For this experiment, analysis consists of finding the frequency of the input signal and plotting the results. The signal frequency can be found with daqdocfft.

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

This function, which is shown on page 3-19, 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 can then be plotted.

The results are plotted below.

```
plot(f, mag)
grid on
ylabel('Magnitude (dB)')
xlabel('Frequency (Hz)')
title(sprintf('daqdoc2\\_2\nFrequency Output by Function
Generator'))
```



This plot shows the nominal frequency around 500 Hz. A simple way to find actual frequency is shown below.

```
ymax = max(mag);
find(mag==ymax)
```

daqdoc3_2.m

The answer is 994 Hz.

Evaluating Your Acquisition Status

At any time after a device object is created, you can evaluate the status of your data acquisition session by:

- Returning the values of certain properties
- Invoking the display summary

The properties related to evaluating your acquisition status and the display summary are discussed below.

Acquisition Status Properties

Acquisition status properties allow you to evaluate whether the toolbox is in the running state or the logging state, as well as how much total data has been acquired and how much data is available in the engine. The properties related to evaluating your acquisition status are given below.

Table 3-4: Analog Input Acquisition Status Properties

Property	Description
`Logging`	Indicates whether data is logged to memory and/or a disk file
`Running`	Indicates whether the device object and data acquisition engine are running
`SamplesAcquired`	Indicates the number of samples acquired per channel
`SamplesAvailable`	Indicates the number of samples available per channel in the data acquisition engine

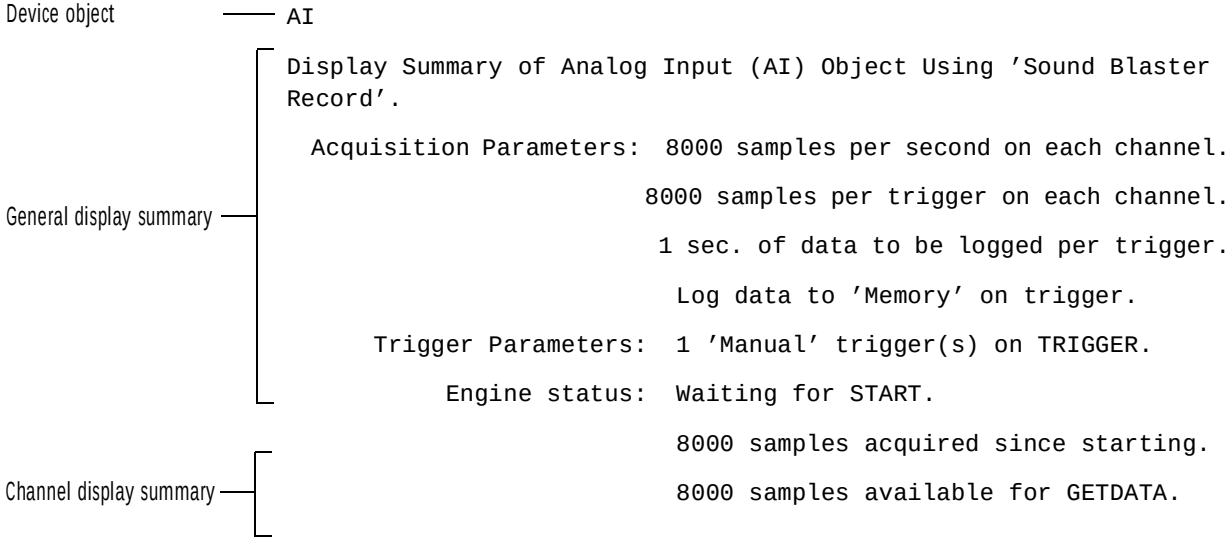
Display Summary

You can invoke the display summary by typing the name of the device object at the command line. The information displayed reflects many of the core properties described on “Controlling Acquisition Behavior with Properties” on page 3-9, and is designed so you can quickly evaluate the status of your data acquisition session. The displayed information is divided into two sections:

- General summary information
- Information about each channel contained by the device object

General summary information includes the device name, the sampling rate, the number of samples to acquire per trigger, the acquisition duration, the destination for logged data, the trigger type, the engine status (whether it is logging data, waiting for a trigger, etc.), the number of samples acquired since starting, and the number of samples available to extract from the engine. Channel information includes the hardware channel mapping, the channel name, and property values related to engineering units.

The display summary for the example on page 3-17 is shown below.



The general summary information reflects the acquisition status after the requested samples have been acquired but before the data is extracted from the engine.

If you only want to evaluate the status of the channels, you must type the name of the channel array.

```
AI.Channel
```

Evaluating Your Hardware Resources

You can evaluate your data acquisition-related hardware resources with the `daqhwinfo` function. Hardware resources include installed hardware, hardware drivers, and adaptors. This information can be displayed at any time to assist you in configuring and evaluating your acquisition.

The information returned by `daqhwinfo` depends on the supplied arguments (if any):

- `daqhwinfo` returns general data acquisition information about the toolbox, MATLAB, and the installed adaptors.
- `daqhwinfo('adaptor')` returns information for the specified adaptor.
- `daqhwinfo(object)` returns driver and hardware information for `object`.
- `daqhwinfo(object, 'Property')` returns information for the specified property.

The hardware information related to the sound card adaptor is given below.

```
daqhwinfo('winsound')
ans =
      AdaptorDllName: 'D:\v5\toolbox\daq\private\mwwinsound.dll'
   AdaptorDllVersion: '1.0.2.1'
         AdaptorName: 'winsound'
          BoardNames: {'Sound Blaster Record'}
   InstalledBoardIds: '[0]'
ObjectConstructorName: {'analoginput('winsound',0)'  [1x26 char]}
```

This information can be displayed at any time to assist you in configuring your data acquisition session.

The driver and hardware information for the sound card example on page 3-17 is shown below.

```
daqhwinfo(AI)
ans =
              AutoCalibrate: 'Off'
                       Bits: 16
           ConversionOffset: 0
                   Coupling: {'AC Coupled'}
                 DeviceName: 'Sound Blaster Record'
       DifferentialChannels: 0
                 DriverName: 'winsound'
                      Gains: []
                         ID: 0
                 InputRanges: [-1 1]
              MaxSampleRate: 44100
              MinSampleRate: 8000
             NativeDataType: 'Int16'
                   Polarity: {'Bipolar'}
                 SampleType: 'SimultaneousSample'
        SingleEndedChannels: 2
              SubsystemType: 'AnalogInput'
              TotalChannels: 2
    VendorDriverDescription: 'Windows Multimedia Driver'
        VendorDriverVersion: '2.0'
```

For more information about `daqhwinfo`, refer to Chapter 6, "Function Reference."

Getting Function and Property Help

You can access help on Data Acquisition Toolbox functions and properties using some or all of these resources:

- The reference sections in this book
- M-file help
- HTML-based property help available through the Help Desk
- The `daqhelp` function
- The `propinfo` function

The `daqhelp` and `propinfo` functions are discussed below.

The daqhelp function

`daqhelp` can be used to display information for both functions and properties. `daqhelp` can also be used to display information about device objects and constructors. A device object need not exist for you to obtain this information. For example, before creating a device object, the command

```
daqhelp analoginput/peekdata
```

displays the M-file help for `analoginput/peekdata`, while the command

```
daqhelp analoginput.TriggerDelayUnits
```

displays the help for the analog input property named `TriggerDelayUnits`. After creating a device object, you can display all the associated help by specifying the device object name as an argument to `daqhelp`. For example, to display all the associated help for the sound card example on page 3-17:

```
daqhelp(AI)
```

The commands listed below are some of the ways `daqhelp` can be used to display help for specific functions and properties for `AI`.

```
daqhelp(AI, 'TriggerDelayUnits');
daqhelp(AI, 'getdata');
daqhelp(AI, 'set');
```

For more information about `daqhelp`, refer to Chapter 6, "Function Reference."

The propinfo function

`propinfo` returns information about device object or channel properties. The information is returned as a structure containing the fields shown below.

Property	Description
`Type`	The property data type (e.g., double, string)
`Constraint`	Constraints on property values (e.g., none, bounded)
`ConstraintValue`	Property value constraint (e.g., range of valid values, elements of an enumerated list)
`DefaultValue`	The property default value
`ReadOnly`	If a property is read-only, a “1” is returned. Otherwise a “0” is returned.
`ReadOnlyRunning`	If a property cannot be set while the device object is running, a “1” is returned. Otherwise a “0” is returned.
`DeviceSpecific`	If the property is device-specific, then a “1” is returned. Otherwise a “0” is returned.

This information can be used simply to evaluate the allowed value(s) for a given property, or to programmatically configure property values. For example, the following command displays all the `propinfo` field values for the `SampleRate` property for the sound card example on page 3-17.

```
info = propinfo(AI);
info.SampleRate
ans =
               Type: 'double'
         Constraint: 'Bounded'
    ConstraintValue: [8000 44100]
       DefaultValue: 8000
           ReadOnly: 0
    ReadOnlyRunning: 1
     DeviceSpecific: 0
```

To display the data type of the `InputRange` property:

```
info1 = propinfo(AI.Channel);
info1.InputRange.Type
ans =
double array
```

For more information about `propinfo`, refer to Chapter 6, "Function Reference."

4

Doing More With Analog Input

Overview

This chapter presents the complete analog input functionality available to you with the Data Acquisition Toolbox. Properties and functions are presented in a way that reflects the typical procedures you will use to set up an analog input data acquisition session. These procedures are grouped as:

- Setting property values
- Referencing analog input objects and channels
- Configuring hardware properties
- Managing acquired data
- Configuring triggers
- Configuring events and actions
- Engineering unit conversion
- Logging information to disk

If you are new to the Data Acquisition Toolbox, you should read Chapter 3, “Getting Started With Analog Input” first.

Setting Property Values

As discussed in "Property Names and Property Values" in Chapter 2, the Data Acquisition Toolbox supports two basic types of properties: base properties and device-specific properties. Base properties apply to all supported hardware devices while device-specific properties are incorporated into the toolbox on a per-device basis. For analog input objects, the base properties are divided into two groups: common properties and channel properties. Common properties apply to all channels in a channel group while channel properties apply on a per-channel basis.

You can set property values for the device object or one or more channels using the `set` command or subscripted assignment. For channels, you can use the `set` command to assign multiple property values for multiple channels. This feature is discussed below.

Setting Multiple Property Values for Multiple Channels

Using the `set` command, you can assign different values for multiple properties and multiple channels.

```
chans = obj.Channel(index);
set(chans, {P1}, {V1},...);
```

where:

- `chans` is an *m*-by-1 array of channels
- `{P1}` is a 1-by-*n* cell array of properties
- `{V1}` is an *m*-by-*n* cell array of property values

Using this syntax, each row of `V1` corresponds to a different channel, each column of `V1` corresponds to a different property, and V_{mn} is the value to set for property *n* of channel *m*.

For example, suppose you create the analog input object `AI` for a sound card, and configure it for stereo operation. The code shown below illustrates how you

can assign a different value to the SensorRange and ChannelName properties for each channel using a single call to set.

```
myAI = analoginput('winsound');
addchannel(myAI, 1:2);
chans = myAI.Channel(1:2);
set(chans, {'SensorRange', 'ChannelName'}, {[-2 2], 'Chan1';...
[0 4], 'Chan2'});
```

Alternatively, you can set these property values one channel at a time

```
set(chans(1), 'SensorRange', [-2 2], 'ChannelName', 'Chan1');
set(chans(2), 'SensorRange', [0 4], 'ChannelName', 'Chan2');
```

or one channel and one property at a time.

```
set(chans(1), 'SensorRange', [-2 2]);
set(chans(1), 'ChannelName', 'Chan1');
set(chans(2), 'SensorRange', [-0 4]);
set(chans(2), 'ChannelName', 'Chan2');
```

Note: You cannot set multiple property values with one call using subscripted assignments.

Referencing Analog Input Objects and Channels

To configure an analog input object to perform data acquisition tasks, you must set the appropriate property values. To set common property values, you must reference the analog input object by its MATLAB variable name. To set channel property values, you must reference individual channels by indexing into the `Channel` property. You can also set property values by:

- Referencing a copy of a device object
- Referencing a copy of one or more channels
- Referencing channels by their channel names

In all cases, referencing occurs through the `set` command or subscripted assignment.

Copying device objects and channels, and referencing channels by name are described below.

Copying Device Objects and Channels

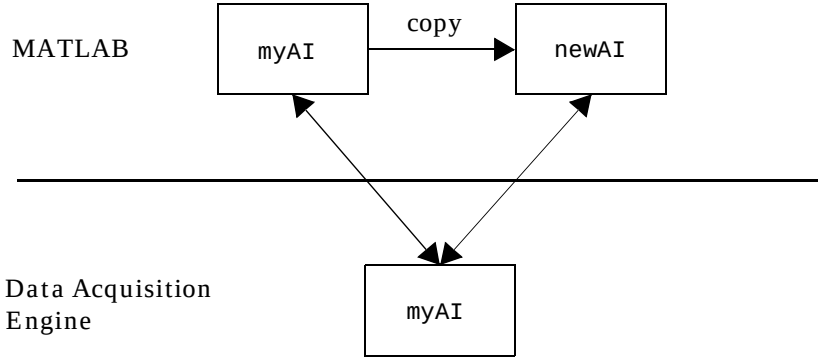
You can keep multiple versions of a device object or channels in the MATLAB workspace by copying them to a new MATLAB variable. Copying device objects and channels are discussed below.

Copying Device Objects

The example code below illustrates how to copy a device object. It is important to note that the copied object is identical to the original object. This can be seen by setting a property for the original object and returning the value of that property from the copied object.

```
myAI = analoginput('nidaq', 1);
addchannel(myAI, 0:1);
newAI = myAI;
set(myAI, 'SamplesPerTrigger', 10000);
get(newAI, 'SamplesPerTrigger')
ans =
       10000
```

The reason that myAI and newAI return the same property value is because both device objects reference the same data acquisition engine object as illustrated in the figure below.



Note: Every data acquisition object that appears as a variable references an object in the data acquisition engine.

If you delete the original device object (or a copy), then the engine object is deleted. Any copies of the device object that remain in the workspace cannot be used to perform a data acquisition task since the object they referenced no longer exists in the engine. In other words, the copies are no longer associated with any hardware. Copied objects that do not reference an engine object are called *invalid objects*. The example code below illustrates this situation.

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

Note: You cannot delete valid objects using the `clear` command. `clear` removes the variable from the MATLAB workspace but it does not remove objects from the engine. To remove objects from the engine, the `delete` command must be used.

Copying Channels

As shown on page 4-4, it is often useful to copy channels to a new variable with a short name and then reference the channels by that name. For example, to copy `myAI`'s entire channel group to a new variable `chans`

```
chans = myAI.Channel;
```

Clearly, it is more convenient to type `chans` than `myAI.Channel`. When referencing `chans`, you can set only channel property values. To configure common property values, you must reference `myAI` or a copy of `myAI`.

You can copy the first channel contained by `myAI` to a new variable `chan1` by indexing into the `Channel` property.

```
chan1 = myAI.Channel(1);
```

You can also copy the channel from `chans`.

```
chan1 = chans(1);
```

If you delete one or more channels from the original device object (or a copy), then those channels are deleted from the engine object. Any copies of the channels that remain in the workspace cannot be used to perform a data acquisition task since the channels they referenced no longer exist in the engine. For example, channel 1 can be deleted from the engine using any of these commands

```
delete(myAI.Channel(1))
delete(chans(1))
delete(chan1)
```

Deleting channels is discussed in detail on page 4-12.

Reference by Channel Name

In some circumstances it may be useful to reference one or more channels by name rather than by index. Names are associated with channels through the `ChannelName` property. You can assign channel names through the `addchannel` function or by directly assigning values to `ChannelName`.

For example, suppose you want `TrigChan` to be the name of the first channel in `myAI`'s channel group. After naming the channel, you can configure its property values by referencing the channel by its name. This feature is illustrated in the example below.

`daqdoc4_1.m`

Initialization: Create the analog input object `myAI` for a sound card. The installed adaptors and hardware ID's are found with `daqhwinfo`.

```
myAI = analoginput('winsound');
%myAI = analoginput('nidaq', 1);
```

Configuration: Add two hardware channels to `myAI`, define an eight second acquisition duration, copy channel 1 to the variable `chan1`, assign a name to channel 1, and set a property value by referencing the channel name. The actual sampling rate is retrieved since it may be set to a value that differs from the specified value.

```
addchannel(myAI, 1:2);
%addchannel(myAI, 0:1);
duration = 8;
ActualRate = get(myAI, 'SampleRate');
chan1 = myAI.Channel(1);
set(chan1, 'ChannelName', 'TrigChan');
set(myAI, 'SamplesPerTrigger', duration*ActualRate);
set(myAI.TrigChan, 'UnitsRange', [0 100]);
myAI.TrigChan.Units = 'dB'
```

Execution: Start `myAI` and wait for the device object to stop running. Eight seconds of data is collected for each channel contained by `myAI`.

```
start(myAI)
while strcmp(myAI.Running, 'On')
end
```

Termination: Display summary information and delete `myAI`.

daqdoc4_1.m

```
myAI
delete(myAI)
```

Alternatively, you can define a channel name with the `addchannel` function.

```
addchannel(myAI, 0, {'TrigChan'});
addchannel(myAI, 1:4);
```

Note: You cannot reference a channel by name if that name does not have the appropriate format. A channel name must contain only letters, numbers, and underscores and must begin with a letter.

To quickly generate a list of channel names, you can use the `makenames` function. This function is described below.

The makenames Function

You can generate a list of channel names with the `makenames` function.

```
names = makenames('prefix', index);
```

where:

- `names` is a column cell array containing the channel names.
- `prefix` is a string which constitutes the first part of the channel name.
- `index` are indices for the channel name which are appended to the end of the `prefix` string. `index` must consist of nonnegative integers.

For example, suppose you want to add hardware channels 5-7 to `myAI` and assign the names `Chan6`, `Chan7`, and `Chan8` to these new channels. You can create the channel names using `makenames` and then pass the resulting cell array to `addchannel`.

```
names = makenames('Chan', 6:8);
addchannel(myAI, 5:7, names);
```

Alternatively, you can assign names to channels after they have been added to the object.

```
addchannel(myAI, 5:7);
names = makenames('Chan', 6:8);
set(myAI.Channel(6:8), {'ChannelName'}, names);
```

Reordering Channels

As described in "Adding Channels to an Analog Input Object" on page 3-5, when channels are added to a device object with the `addchannel` function, a mapping is defined between hardware channel IDs and MATLAB indices. This mapping allows you to configure specific hardware channels to suit your data acquisition needs.

If you are using scanning hardware, then this mapping also establishes the channel scanning order. The hardware scan order is given by the elements of the `HwChannel` property. Namely, the first `HwChannel` element is sampled first, the second `HwChannel` element is sampled second, and so on. The scan order can be easily displayed by typing the name of the device object at the MATLAB command line and observing the elements of `HwChannel`. Under certain circumstances, you may want to explicitly reorder an existing channel group. There are three ways hardware channels can be reordered:

- Explicitly swapping the MATLAB indices
- Adding channels to an existing channel group
- Deleting channels from an existing channel group

These methods of reordering channels are discussed below.

Note: Channels cannot be reordered for sound cards. The rules for adding and deleting hardware channels for sound cards are described in "Example: Adding Channels for a Sound Card" on page 3-7.

Swapping MATLAB Indices

You can reorder an existing channel group by swapping the MATLAB indices of the channels with the `Channel` property.

For example, suppose you created the analog input object `myAI` and added hardware channels 0-2 to it. If the second hardware channel is to be a trigger, but you want that channel to be the first one sampled, then you can swap the indices for channels one and two.

```
myAI = analoginput('nidaq', 1);
addchannel(myAI, 0:2);
myAI.Channel(1:3) = myAI.Channel([2 1 3])
```

Note that when swapping indices, you must explicitly specify the complete index permutation. You cannot specify just the indices you want to swap.

Adding Channels to an Existing Channel Group

Channel reordering can occur when you add channels to an existing channel group. When adding channels, these rules must be followed:

- The resulting channel indices must begin at 1.
- The resulting channel indices must increase monotonically up to the total number of channels contained by the device object.

If channels are added to an existing channel group and channel indices are not provided, then the MATLAB indices are automatically assigned the next available values, in order. For example, suppose hardware channels 3 and 4 are added to `myAI` using the next available indices.

```
addchannel(myAI, 3:4);
```

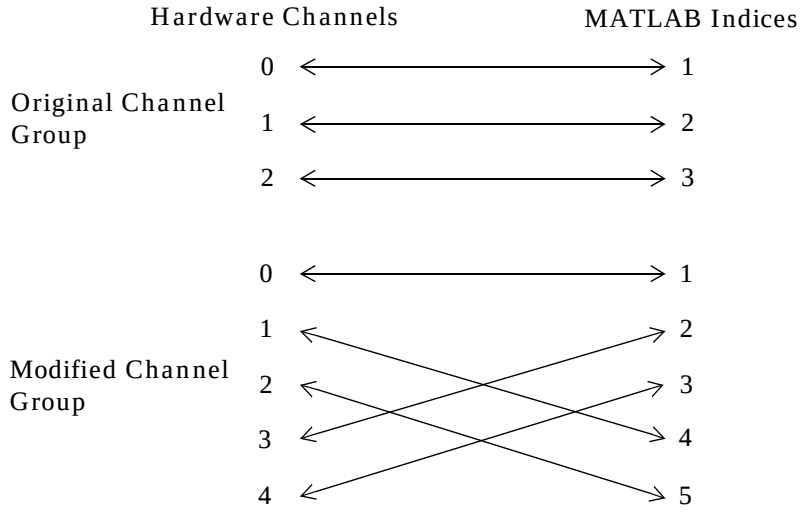
The channel group now contains hardware channels 0 to 4 which correspond to indices 1 to 5, respectively.

If channels are added to an existing channel group and channel indices are provided but previously assigned, then the previously assigned channels are reindexed to the next available values, in order. For example, suppose hardware channels 3 and 4 are added to `myAI` and assigned indices 2 and 3.

```
addchannel(myAI, 3:4, 2:3);
```

The channel group now contains hardware channels 0 to 4 which correspond to indices 1, 4, 5, 2, and 3, respectively.

The figure below illustrates this situation.



Deleting Channels From an Existing Channel Group

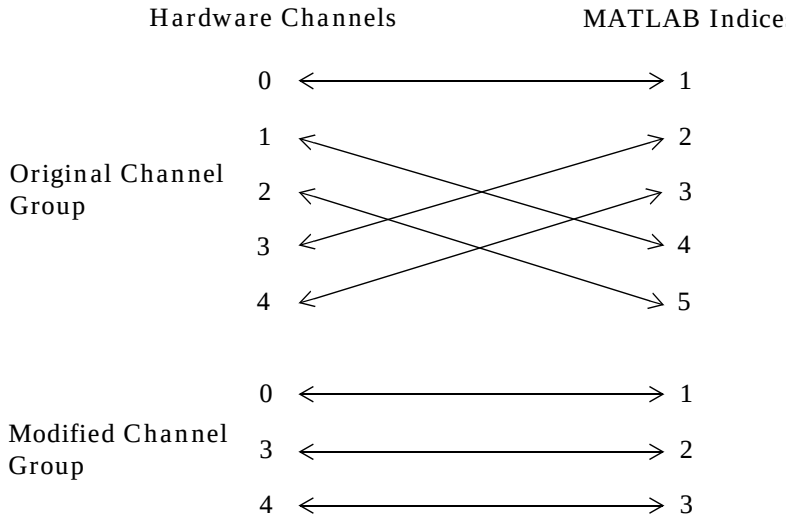
You can delete channels from a channel group with the `delete` command. When a channel is deleted, it is removed from the channel group and the indices of the remaining channels are reindexed such that the rules described on page 4-11 are satisfied.

For example, suppose you want to delete hardware channels 1 and 2 from the modified channel group shown above.

```
chans = myAI.Channel(4:5);
delete(chans);
```

The new channel group now contains hardware channels 0, 3, and 4 which correspond to the indices 1,2, and 3.

The figure below illustrates this situation.



As mentioned in "Copying Device Objects" on page 4-5, you cannot delete valid channels using the `clear` command. `clear` removes the variable from the MATLAB workspace but it does not remove objects from the engine.

Note: As in Handle Graphics, the `clear` and `delete` functions are not equivalent.

Configuring Hardware Properties

The hardware you are using contains functionality that satisfies your specific application needs. Typical hardware functionality includes device type (analog input, analog output, digital I/O), the number of channels or lines, and the maximum sample rate, among others. This section presents the base properties that allow you to configure your hardware. These properties are shown below.

Table 4-1: Analog Input Hardware Configuration Properties

Property	Description
`ChannelSkew`	The delay (in seconds) between scanned channels
`ChannelSkewMode`	Mode that defines the channel skew
`InputRange`	Range of A/D converter
`InputType`	Hardware channel input configuration
`SampleRate`	Sampling rate per channel (in Hz)

Except for `InputRange`, these properties are described in detail below. `InputRange` is described in "Engineering Unit Conversion" on page 4-59.

Setting and Verifying Hardware Properties

For the `ChannelSkew`, `InputRange`, and `SampleRate` properties, data acquisition devices have a finite (though sometimes large) number of valid values that can be set. If you specify a property value that does not match one of the valid device values, then the data acquisition engine will choose a valid value automatically. The rules the engine uses to select a valid property value are described for each property in Chapter 7, "Property Reference".

You can use the `propinfo` function to obtain information about the valid values for some of these properties. For example, suppose you create the analog input

object AI for a sound card. You can use propinfo to display information about the SampleRate property.

```
AI = analoginput('winsound')
propinfo(AI, 'SampleRate')
ans =
               Type: 'double'
         Constraint: 'Bounded'
    ConstraintValue: [8000 44100]
       DefaultValue: 8000
           ReadOnly: 0
    ReadOnlyRunning: 1
     DeviceSpecific: 0
```

The Constraint field describes the constraints on the property value, while the ConstraintValue field gives the property value constraint. In the example above, Bounded means there are a large number of valid values for SampleRate and ConstraintValue gives the upper and lower limits (the bounds) for SampleRate. For more information about propinfo, refer to Chapter 6, "Function Reference."

Note: For some sound cards, you can set the sampling rate to any value between the minimum and maximum values defined by the hardware. This is due to onboard decimation.

To find out if the engine sets a property to a value that differs from the one you specified, you should retrieve the value after it is set. For example, after setting the sampling rate for a National Instruments device, you should retrieve the actual rate set since you may want to use this value to set additional property values.

```
AI = analoginput('nidaq', 1)
addchannel(AI, 0:7)
set(AI, 'SampleRate', 5000)
ActualRate = get(AI,'SampleRate')
duration = 10 % Ten second acquisition
set(AI, 'SamplesPerTrigger', duration*ActualRate)
```

As an alternative to the syntax shown above, you can use the setverify function. setverify is equivalent to the commands

```
set(obj, 'Property', Value)
Actual = get(obj, 'Property')
```

Using setverify, the previous example is written as

```
AI = analoginput('nidaq', 1)
addchannel(AI, 0:7)
ActualRate = setverify(AI,'SampleRate', 5000)
duration = 10 % Ten second acquisition
set(AI, 'SamplesPerTrigger', duration*ActualRate)
```

Configuring Hardware Input Channels

You can physically configure your hardware input channels with the InputType property. The hardware-specific values for this property are shown below.

Table 4-2: InputType Values

Vendor/Device	Value
Sound Cards	AC-Coupled
National Instruments	{SingleEnded}|Differential| NonReferencedSingleEnded

For sound cards, the only valid InputType value is AC-Coupled. For National Instruments boards, the input channels can be configured as single-ended or differential. The E-series 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 inputs are configured as differential, you can only add channels 0-7, 16-23, 32-39, and 48-55. You should use the differential mode whenever possible since this mode helps to reduce signal noise. Consult your hardware manual for details on how to connect your signal source to your data acquisition hardware.

Note: If the `InputType` value is changed, and that change decreases the number of channels contained by the analog input object, then a warning is displayed and all channels are deleted.

Sampling Channels

Sampling channels can be one of the most important and complicated considerations for many data acquisition applications. Within the Data Acquisition Toolbox, sampling channels involves properties related to the sampling rate and the channel skew. These topics are discussed below.

Sampling Rate

The sampling rate is given by the `SampleRate` property and is defined as the number of samples acquired per second from each channel group member. However, the maximum rate that channels can be sampled at depends on the type of hardware you are using.

If you are using simultaneous sample and hold (SSH) 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, configure it for stereo operation, and set the sampling rate to 44,100 Hz.

```
AI = analoginput('winsound');
addchannel(AI, 1:2);
set(AI, 'SampleRate', 44100);
```

Both channels are sampled at 44,100 Hz.

If you are using scanning hardware, 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 device, 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.

```
AI = analoginput('nidaq',1);
set(AI, 'InputType', 'SingleEnded');
addchannel(AI, 0:9);
set(AI, 'SampleRate', 10000);
```

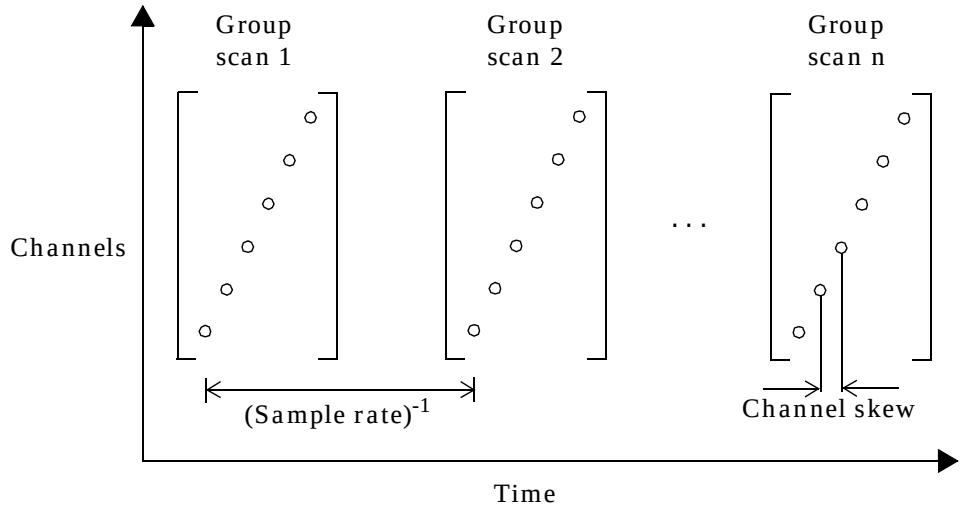
Typically, this maximum rate can be achieved only under ideal conditions. In practice, the sampling rate depends on many variables including the settling time, the gain, and the channel skew as well as the number of channels you acquire data from. Channel skew is discussed in the next section.

Note: Whenever the SampleRate value is changed, the BufferingConfig values are recalculated by the engine if BufferingMode is set to Auto.

Channel Skew

Channel skew is defined as the time (in seconds) between consecutive scanned channels in a channel group.

The relationship between channel skew and sampling rate is illustrated below.



Channel skew is controlled with the `ChannelSkewMode` property. The hardware-specific values for this property are shown below.

Table 4-3: ChannelSkewMode Values

Value	Description
`None`	No channel skew is defined. This is the only valid value for simultaneous sample and hold (SSH) hardware.
`Equisample`	The channel skew is automatically calculated as (sampling rate × numbers of channels)$^{-1}$
`Manual`	The channel skew must be set with the `ChannelSkew` property
`Minimum`	The channel skew is given by the smallest value supported by the hardware

If ChannelSkewMode is set to Minimum or Equisample, then ChannelSkew contains the appropriate read-only value. If ChannelSkewMode is set to Manual, the channel skew is given by the ChannelSkew property.

ChannelSkew is settable only for scanning hardware and not for simultaneous sample and hold (SSH) hardware. For SSH hardware, ChannelSkewMode can only be None and the only valid ChannelSkew value is 0.

Additional Hardware Properties

Depending on the hardware you are using, there may be additional hardware-related properties. To find these properties:

- Type set(obj). The hardware-specific properties are included at the bottom of the property list. For example, sound cards have a BitsPerSample property which allows you to select the number of bits used to represent each sample.
- Refer to Appendix A.

Managing Acquired Data

There are two basic ways data is managed within the Data Acquisition Toolbox. One way is to preview the data with the `peekdata` function, the other way is to extract the data from the engine with the `getdata` function. After data is extracted from the engine, it can be analyzed, saved to disk, etc. In addition to these two functions, there are several properties associated with managing acquired data. These properties are listed below.

Table 4-4: Data Management Properties

Property	Description
`SamplesAcquired`	Total number of samples acquired per channel
`SamplesAvailable`	Samples available in the engine to extract per channel
`SamplesPerTrigger`	Number of samples to acquire per channel for each trigger

Previewing data and extracting data from the engine are described below.

Previewing Data

Before you begin extracting and processing acquired data, you may 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 set up correctly. Once you are convinced that your data acquisition session is configured correctly, you may still want to monitor the data even as it is being analyzed or saved to disk.

Previewing data is managed with the `peekdata` function. After `start` is issued, `peekdata` can be called. Since `peekdata` is a nonblocking function that immediately returns control to the MATLAB environment, some data may be missed or repeated. The syntax for `peekdata` is

```
data = peekdata(obj, samples);
```

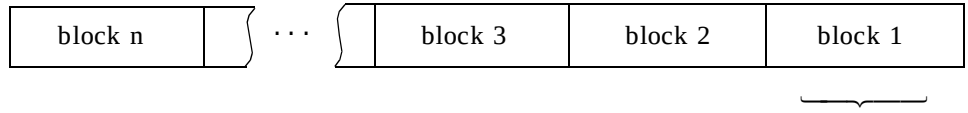
where `samples` is the number of samples to preview.

If the number of samples requested is greater than the number of samples available in the engine, the most recent samples are returned along with a

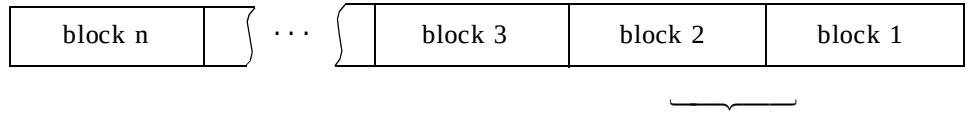
warning message stating that all the requested samples were not available. peekdata does not remove samples from the engine.

peekdata and Data Blocks

When a trigger is issued, acquired data fills data blocks in the engine. Data blocks are discussed in "The Data Block" on page 2-22. When a peekdata function call is processed, the most recent samples requested are immediately returned but the data is not removed from the engine. That is, a "snapshot" of the most recent requested samples are returned. This situation is illustrated below where block 1 is the first block filled with data and block n is the most recent block filled with data.



If another snapshot is requested, then the next most recent samples are returned, as shown below.



Example: Polling the Data Block

Under certain circumstances, you may want to poll the data block. Polling the data block is useful when calling peekdata since 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.

An example that polls the data block is given below.

daqdoc4_2.m

Initialization: Create the analog input object AI for a sound card. The available adaptors hardware ID's are found with daqhwinfo.

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

Configuration: Add one hardware channel to AI, define a ten second acquisition, set up a plot, and store the plot handle and title handle in the variables P and T, respectively.

```
addchannel(AI, 1);
%addchannel(AI, 0);
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));
T = title([sprintf('daqdoc4\\_2\nNumber of peekdata calls: '),
num2str(0)]);
xlabel('Samples'); axis([0 1000 -1 1]); grid on
```

Execution: Start AI and update the display for each 1000-sample block stored in the engine by polling SamplesAcquired. The drawnow command forces MATLAB to update the plot. Since peekdata is used, all acquired data may 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('daqdoc4\\_2\nNumber of peekdata
   calls: '), num2str(i)]);
   drawnow
   i = i + 1;
end
while strcmp(AI.Running, 'On')
end
```

daqdoc4_2.m

Termination: Delete myAI.

```
delete(AI)
```

As you run this example, you may not preview all 80,000 samples stored in the engine. This is because the engine is filled with 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 be processed in some way immediately after it is collected. For example, you may 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 in a timely fashion so that no samples are missed.

Data is extracted from memory with the getdata function. getdata is a blocking function that returns control to the MATLAB environment only when the requested data is available. Therefore, samples are not missed or repeated.

The syntax for getdata is

```
[data, time, abstime, events] = getdata(obj, samples, type);
```

where:

- samples (optional) is the number of samples to return. If samples is not specified, then the number of samples to return is given by the SamplesPerTrigger property.
- type (optional) specifies the format of the returned data. If type is specified as 'native', then data is returned in the native data format of the device. If type is not specified, samples are returned as doubles.
- data is an *m*-by-*n* array containing acquired data where *m* is the number of samples and *n* is the number of channels.
- time is an *m*-by-1 array containing the time values for all *m* samples acquired. time = 0 corresponds to the time the first sample was logged by the data acquisition engine. Time is measured continuously until the acquisition is stopped. Absolute times are available through the TriggerTime property.

- `abstime` is the absolute time of the trigger associated with `data`.
- `events` is a structure containing a list of events that have occurred up to the time of the `getdata` call. The possible events that can be returned are identical to those stored by the `EventLog` property.

The relationship between the samples acquired and the relative time for each sample is shown below for `m` samples and `n` channels.

Data array. Each column represents one channel

$$\begin{bmatrix} d_{11} & d_{12} & & d_{1n} \\ d_{21} & d_{22} & & d_{2n} \\ d_{31} & d_{32} & \cdots & d_{3n} \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ d_{m1} & d_{m2} & & d_{mn} \end{bmatrix}$$

Relative time array

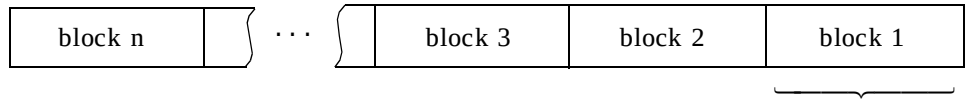
$$\begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ . \\ . \\ . \\ t_m \end{bmatrix}$$

As soon as a trigger is issued and samples are stored in the engine, the `SamplesAcquired` property keeps a running count of the total number of samples per channel that have been acquired, while the `SamplesAvailable` property tells you how many samples can be extracted from the engine per channel with `getdata`. When a `getdata` call is issued, `SamplesAvailable` is reduced by the number of samples requested.

getdata and Data Blocks

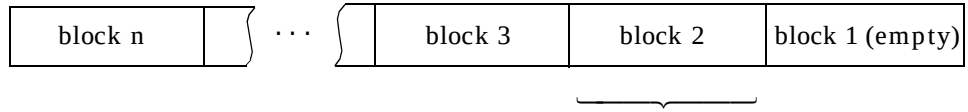
When a trigger occurs, acquired data fills data blocks in the engine. Data blocks are discussed in “The Data Block” on page 2-22. When a `getdata` function call is processed, the requested samples are returned when the data is available, and then removed from the engine. The situation for when `getdata` extracts a

full block of data is shown below block 1 is the first block filled with data and block n is the most recent block filled with data.



Read one complete block of data

If another `getdata` call is issued, then those samples are returned such that no data is missed. This situation is shown below.



Read the next complete block of data

Example: Previewing and Extracting Data

Suppose you have a time-consuming data acquisition application. By previewing the data, you can ascertain whether the acquisition is proceeding as expected without waiting for all the data to be acquired. If it is not, then you can abort the session and diagnose the problem.

The example below illustrates how you might use `peekdata` and `getdata` together in such an application.

`daqdoc4_3.m`

Initialization: Create the analog input object `AI` for a sound card. The available adaptors are found with `daqhwinfo`.

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

Configuration: Add one hardware channel to AI, define a ten second acquisition, set up the plot and store the plot handle in the variable P. The amount of data displayed is given by preview.

```
chan = addchannel(AI,1);
%chan = addchannel(AI,0);
duration = 10; % Ten second acquisition
set(AI,'SampleRate',8000);
ActualRate = get(AI,'SampleRate');
set(AI,'SamplesPerTrigger',duration*ActualRate);
preview = duration*ActualRate/100;
figure
set(gcf, 'doublebuffer', 'on');
subplot(211);
P = plot(zeros(preview,1)); grid on
title(sprintf('daqdoc4\\_3\nPreview Data'))
xlabel('Samples')
ylabel('Signal Level (Volts)')
```

Execution: Start AI and update the display using peekdata every time an amount of data given 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 may not be displayed.

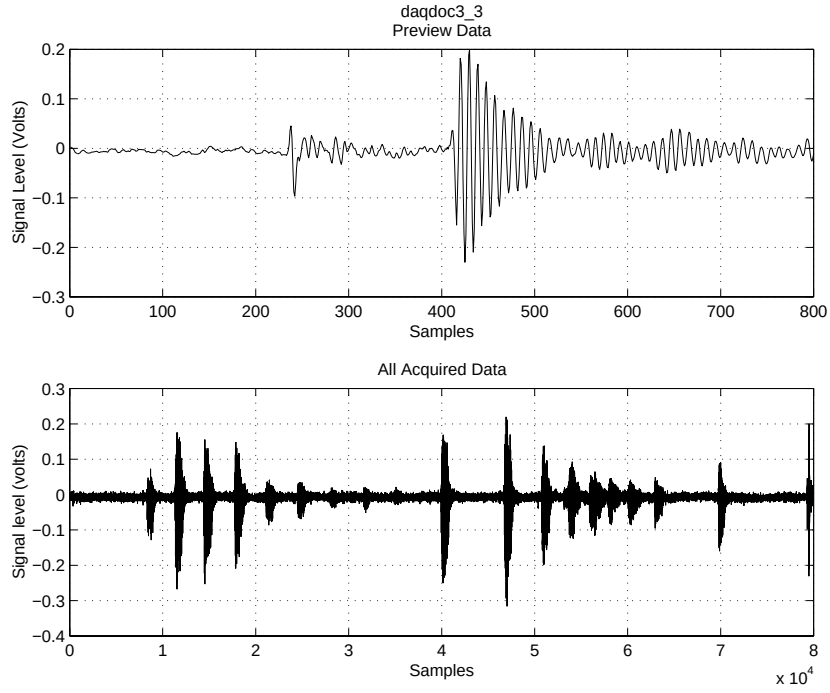
```
start(AI);
while AI.SamplesAcquired < preview
end
while AI.SamplesAcquired < duration*ActualRate
   data = peekdata(AI, preview);
   set(P, 'ydata', data);
   drawnow;
end
data = getdata(AI);
```

Termination: Wait for AI to stop running, plot all the extracted data, and delete the device object.

daqdoc4_3.m

```
subplot(212); plot(data);  grid on
title('All Acquired Data')
xlabel('Samples')
ylabel('Signal level (volts)')
delete(AI)
```

The data is shown below.



Calculating Time

Relative and absolute time information can be returned to you via the getdata function and the TriggerTime property. getdata can return a vector of relative times associated with the extracted data, as well as the absolute time of the

first trigger. TriggerTime contains the absolute times for each trigger executed. Absolute time and relative time are explained below.

Absolute Time

Absolute times are recorded for each trigger issued. Absolute time can be returned with both the getdata function and TriggerTime property. TriggerTime is discussed in detail in "Configuring Triggers" on page 4-30.

Absolute time is returned in datenum form. It is often convenient to convert a datenum value to a string using datestr. The example below illustrates how you can return data, relative time and absolute time using getdata.

```
[data, time, abstime] = getdata(AI);
abstime
abstime =
  7.3000e+005
```

You can then convert the absolute time to a string using datestr.

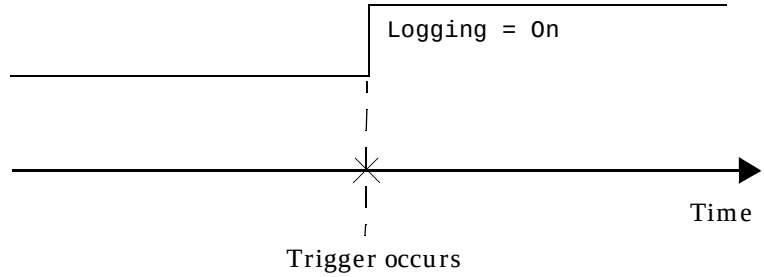
```
abstime_str = datestr(abstime,13)
abstime_str =
11:55:14
```

Relative Time

Relative time is returned with the getdata function. time = 0 corresponds to the time the first sample is logged by the data acquisition engine. Time is measured continuously until the acquisition is stopped. You can always find the absolute time that a sample was acquired at by adding its relative time to the absolute time of the preceding trigger. Absolute trigger times are given by the TriggerTime property.

Configuring Triggers

An analog input trigger is defined as the event that initiates data logging. Data can be logged to memory and/or 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 engine and/or a disk file.



When defining a trigger, you must specify the trigger type. Additionally, you may specify:

- Trigger conditions
- Trigger condition values
- A trigger delay

Properties that allow you to configure a trigger are shown below.

Table 4-5: Analog Input Trigger Properties

Property	Description
`TriggerChannel`	Array of channels serving as trigger sources
`TriggerCondition`	Condition that must be satisfied for a trigger to be issued (depends on `TriggerType`)
`TriggerConditionValue`	Trigger condition value
`TriggerDelay`	Delay for trigger. A negative delay defines a pretrigger while a positive delay defines a postrigger

Table 4-5: Analog Input Trigger Properties

`TriggerDelayUnits`	Determines how a trigger delay is measured (time or samples)
`TriggerAction`	M-file that is executed after the trigger is issued
`TriggerRepeat`	Number of times to repeat the trigger
`TriggerTime`	Array of observed absolute trigger times
`TriggerType`	Source of the trigger

Additionally, the `SamplesPerTrigger` property specifies the number of samples to acquire per trigger for each channel group member.

Except for `TriggerAction`, these trigger-related properties are discussed in the following sections. `TriggerAction` is discussed in "Configuring Events and Actions" on page 4-48.

Trigger Types and Trigger Conditions

Trigger types specify how data logging is initiated and is specified with the `TriggerType` property. You can think of the trigger type as the source of the trigger.

Depending on the `TriggerType` value, there may be associated trigger conditions. Trigger conditions determine the condition under which the specified trigger is issued. For some trigger conditions, you may need to specify a trigger condition value. Trigger conditions are specified with the `TriggerCondition` property. Trigger condition values are specified with the `TriggerConditionValue` property.

The supported trigger types and trigger conditions for analog input objects are shown below.

Table 4-6: TriggerType and TriggerCondition Values

TriggerType	TriggerCondition	Description
`Manual`	`None`	The trigger occurs just after the `trigger` function is issued
`{Immediate}`	`None`	The trigger occurs just after the `start` command is issued

Table 4-6: TriggerType and TriggerCondition Values (Continued)

Software		You must specify one or more channels as trigger sources. The trigger is issued when a signal satisfying the specified condition is detected on one of the specified channels.
	Rising	The signal must be above the specified value and rising
	Falling	The signal must be below the specified value and falling
	Leaving	The signal must be leaving the specified range of values
	Entering	The signal must be entering the specified range of values

For some hardware, additional trigger types and trigger conditions are available. Refer to Chapter 7, "Property Reference" for these device-specific values.

The default `TriggerType` value is `Immediate`. An immediate trigger automatically occurs just after the `start` command is issued. If `TriggerType` is `Manual`, the trigger occurs just after you issue the `trigger` command. An example using a manual trigger is given in "Acquiring Data with a Sound Card" on page 3-17. Software triggers are triggers that are issued by the engine when the associated trigger condition is satisfied. Using software triggers is illustrated in the example below.

Note: After a trigger is issued, the samples specified by `SamplesPerTrigger` are acquired before another trigger is issued.

Example: Voice Activation Using a Software Trigger

Suppose you want to collect voice data with a sound card, and the condition for collecting this data is based on how loudly you speak. An application of this kind is called *voice activation*.

The example code shown below demonstrates how you set up an acquisition with a sound card based on voice activation. The sample rate is set to its maximum value and data is logged only after an acquired sample is greater than or equal to 0.2 volts with a rising (positive) slope. This condition is carried out by means of a software trigger. A portion of the collected data is then extracted from the engine and plotted.

daqdoc4_4.m

Initialization: Create the analog input object AIVoice for a sound card. The installed adaptors and hardware ID's are found with daqhwinfo.

```
AIVoice = analoginput('winsound');
%AIVoice = analoginput('nidaq', 1);
```

Configuration: Add one hardware channel to AIVoice, and copy it to the variable chan, define a two second acquisition, and set up a software trigger. The source of the trigger is chan, and the trigger is issued when a rising voltage level has a value of at least 0.2 volts.

```
chan = addchannel(AIVoice, 1);
%chan = addchannel(AIVoice, 0);
set(AIVoice, 'SampleRate', 44100);
duration = 2; % two second acquisition
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);
```

Execution: Start AIVoice, wait for AIVoice to stop running, and extract the first 1000 samples from the engine as sample-time pairs. Display the number of samples remaining in the engine.

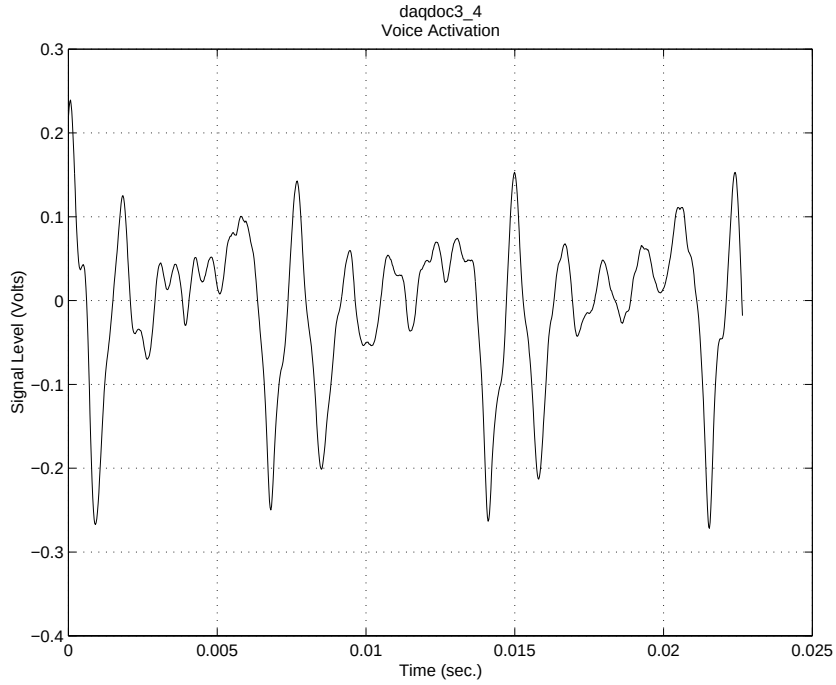
```
start(AIVoice);
while strcmp(AIVoice.Running, 'On')
end
[d, t] = getdata(AIVoice, 1000);
remsamp = num2str(AIVoice.SamplesAvailable);
disp(['Number of samples remaining in engine: ', remsamp])
```

Termination: Plot all the extracted data and delete `AIVoice`.

```
plot(t, d)
drawnow
xlabel('Time (sec.)')
ylabel('Signal Level (Volts)')
title(sprintf('daqdoc4\\_4\nVoice Activation'))
grid on
delete(AIVoice)
```

daqdoc4_4.m

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 volts, and this value corresponds to time = 0. Note that after the `getdata` command is issued, there remains 87,200 samples in the engine.

```
AIVoice.SamplesAvailable
ans =
       87200
```

Trigger Delays

Under certain circumstances, you may want to specify a *trigger delay*. Trigger delays allow you to control exactly when data is logged after a trigger is issued. You can log data before a trigger is issued or after a trigger is issued. Trigger delays are specified with the `TriggerDelay` property.

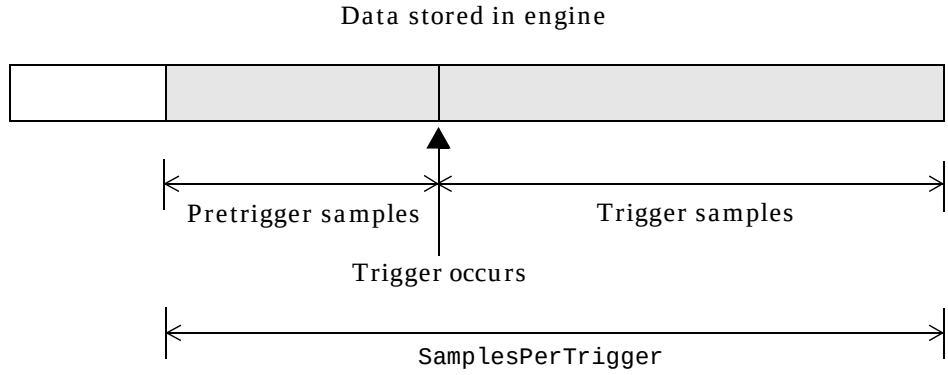
Logging data before a trigger is issued is called *pretriggering*, while logging data after a trigger is issued is called *postriggering*. Pretriggers are specified with negative `TriggerDelay` values, while postriggers are specified with positive `TriggerDelay` values. Logging can be delayed in time 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. Pretrigger data and postrigger data are described below.

Pretrigger Data

In some circumstances, you may want to capture data before the trigger occurs. Such data is called pretrigger data.

For example, you may want to examine the noise seen by your detector before it provides well-behaved output. Since it can be difficult to characterize a noisy signal in terms of triggering on a voltage level, it may be easier to capture that data as pretrigger data. When specifying pretrigger data to capture, the

SamplesPerTrigger value reflects both the data captured before and after the trigger occurs. Pretriggering is illustrated in the figure below.



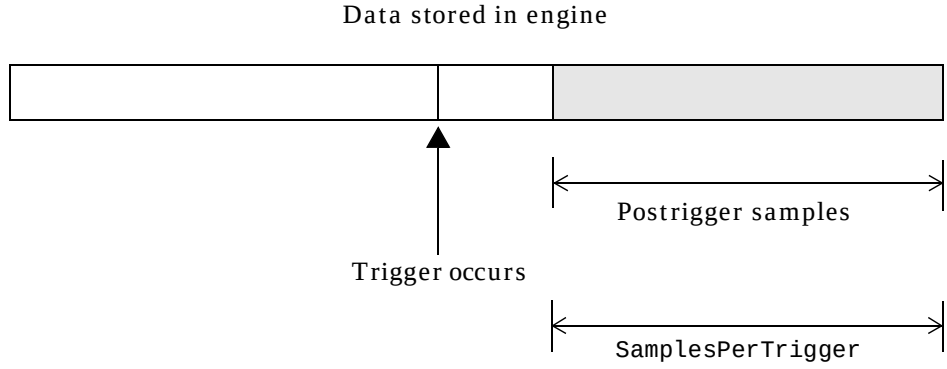
A trigger cannot be issued while the current trigger data is being processed.

Note: You cannot capture pretrigger data when the TriggerType value is Immediate.

Postrigger Data

You may want to capture data after the trigger occurs. Such data is called postrigger data.

When specifying postrigger data to capture, the `SamplesPerTrigger` value and the postrigger samples are equal. Postriggering is illustrated in the figure below.



Example: Voice Activation and Pretriggers

This example modifies the previous voice activation example such that 500 pretrigger samples are collected.

`daqdoc4_5.m`

Initialization: Create the analog input object `AIVoice` for a sound card. The available adaptors are found with `daqhwinfo`.

```
AIVoice = analoginput('winsound');
%AIVoice = analoginput('nidaq', 1);
```

Configuration: Add one hardware channel to `AIVoice`, and copy it to the variable `chan`, define a two second acquisition, and set up a software trigger. The source of the trigger is `chan`, and the trigger is issued when a rising voltage

level has a value of at least 0.2 volts. Additionally, 500 pretrigger samples are collected.

```
chan = addchannel(AIVoice, 1);
%chan = addchannel(AIVoice, 0);
set(AIVoice, 'SampleRate', 44100);
duration = 2; % two second acquisition
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, 'TriggerDelay', -500)
set(AIVoice, 'TriggerDelayUnits', 'Samples');
```

Execution: Start AIVoice, wait for AIVoice to stop running, and extract the first 1000 samples from the engine as and return them as sample-time pairs.

```
start(AIVoice);
while strcmp(AIVoice.Running, 'On')
end
[d, t] = getdata(AIVoice, 1000);
```

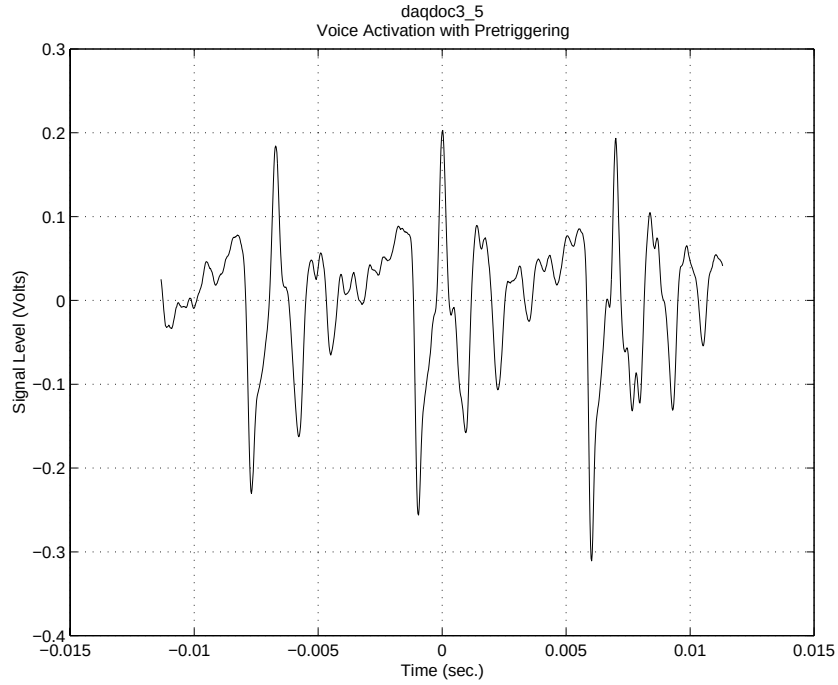
Termination: Plot all the extracted data and delete AIVoice.

```
plot(t, d)
xlabel('Time (sec.)')
ylabel('Signal Level (Volts)')
title(sprintf('daqdoc4\\_5\nVoice Activation with
Pretriggering'))
grid on
delete(AIVoice)
```

daqdoc4_5.m

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 since time = 0 corresponds to the time the trigger occurs and data logging is initiated.



Repeating Triggers

Triggers may be configured to occur once (one-shot acquisition) or repeated times. Trigger repeats are controlled with the `TriggerRepeat` property. If `TriggerRepeat` is set to its default value of 0, then the trigger occurs once when the trigger condition is met. If `TriggerRepeat` is set to a positive integer value, then the trigger is repeated the specified number of times when the trigger condition is met. If `TriggerRepeat` is set to `Inf` then the trigger repeats continuously when the trigger condition is met and the data acquisition can be stopped only with an explicit `stop` command.

Example: Voice Activation and Repeating Triggers

In this example, two triggers are issued. An amount of data given by `SamplesPerTrigger` is acquired for each trigger and stored in a separate variable. The `Timeout` value is set to five seconds. Therefore, if `getdata` does not return the specified number of samples in five seconds plus the amount of time required to acquire the data, the acquisition will be aborted.

`daqdoc4_6.m`

Initialization: Create the analog input object `AIVoice` for a sound card. The available adaptors are found with `daqhwinfo`.

```
AIVoice = analoginput('winsound');
%AIVoice = analoginput('nidaq', 1);
```

Configuration: Add one hardware channel to `AIVoice`, and copy it to the variable `chan`, define a one second acquisition, and set up a software trigger. The source of the trigger is `chan`, and the trigger is issued when a rising voltage level has a value of at least 0.2 volts. Additionally, the trigger is repeated once when the trigger condition is met.

```
chan = addchannel(AIVoice, 1);
%chan = addchannel(AIVoice, 0);
set(AIVoice, 'SampleRate', 44100);
duration = 0.5; % half second acquisition for each trigger
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);
```

Execution: Start `AIVoice`, wait for `AIVoice` to stop running, 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 the data acquired from both

triggers can be extracted with one `getdata` call by specifying `getdata(AIVoice, 44100)`.

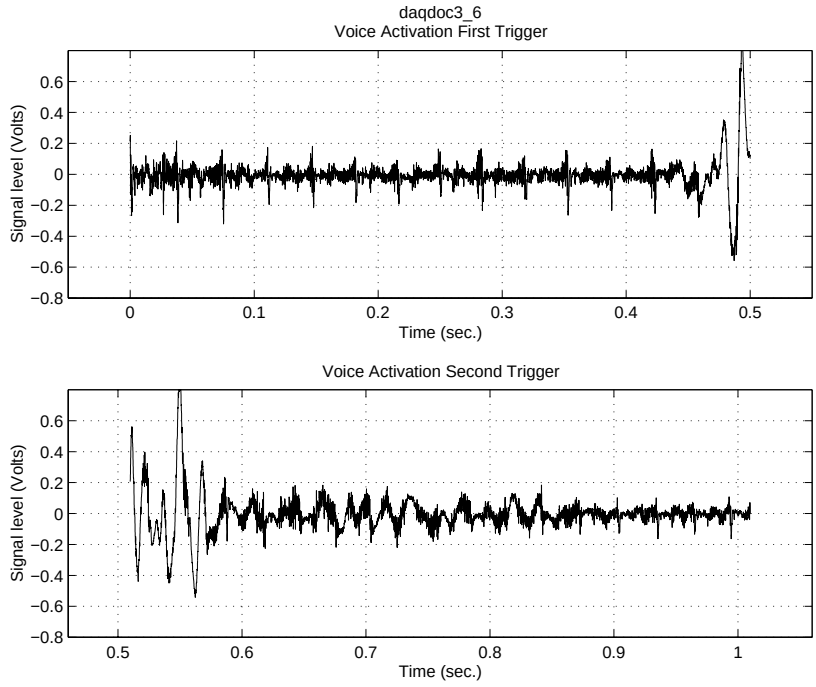
```
start(AIVoice);
while strcmp(AIVoice.Running, 'On')
end
[d1, t1] = getdata(AIVoice);
[d2, t2] = getdata(AIVoice);
```

Termination: Plot the data for both triggers and delete `AIVoice`.

```
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(sprintf('daqdoc4\\_6\nVoice 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');
delete(AIVoice)
```

daqdoc4_6.m

The data acquired for both triggers is shown below



You can keep track of the number of triggers issued and the absolute times they occurred with the `TriggerTime` property. To find out how many triggers occurred, you can use MATLAB's `length` command

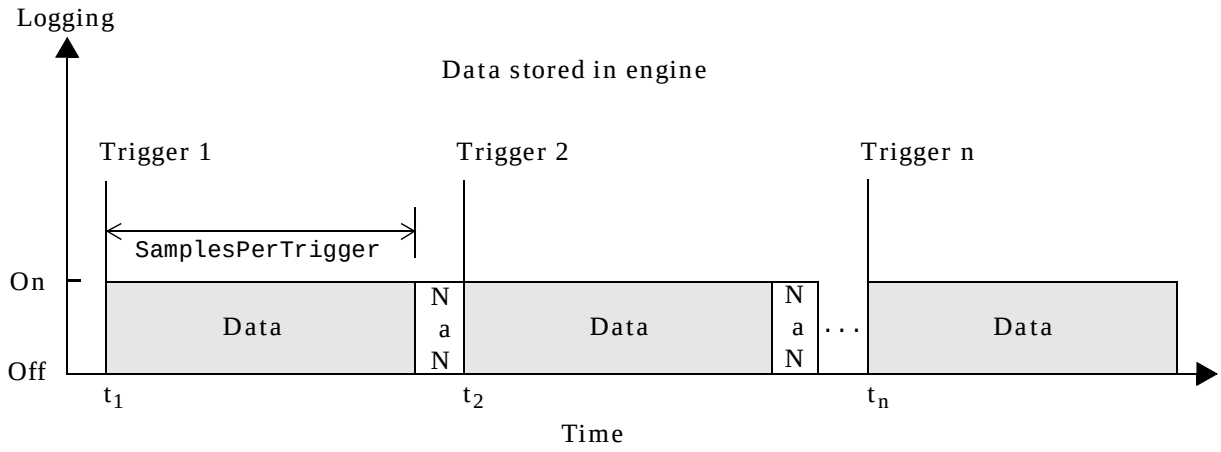
```
NumTrigs = length(AIVoice.TriggerTime)
```

To find out the trigger times, you must convert the elements of the `TriggerTime` array from the `datenum` format to a string. For example, the time of the first trigger is calculated below.

```
times = AIVoice.TriggerTime;
time1_str = datestr(times(1), 13)
time1_str =
14:08:50
```

You can also return data from multiple triggers with one call to `getdata` by specifying the appropriate amount of data. This number is given by (`SamplesPerTrigger`) × (`TriggerRepeat` + 1). When you return data that spans multiple triggers, a NaN is inserted in the data stream between triggers. Therefore, an extra "sample" (the NaN) is stored in the engine and returned by `getdata`. Identifying these NaNs allows you to identify where and when each trigger was issued in the data stream.

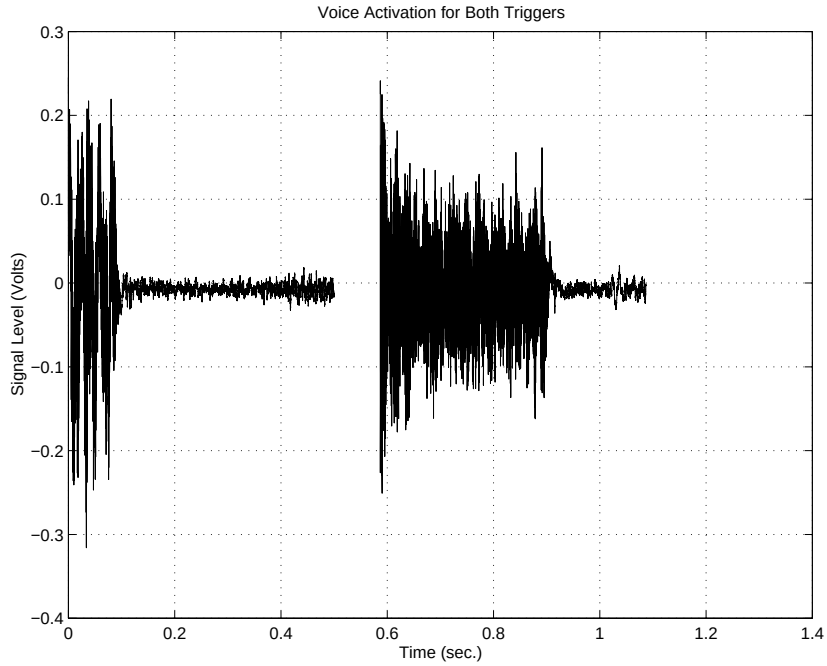
The figure below illustrates the data stored by the engine during a multiple-trigger acquisition, and the relationship between `SamplesPerTrigger` and the time recorded by `TriggerTime`. The first element of `TriggerTime` corresponds to t_1, the second trigger time corresponds to t_2, and so on.



The commands shown below modify the previous example so that multiple-trigger data is extracted from the engine with one call to `getdata`.

```
returndata = ActualRate*duration*(AIVoice.TriggerRepeat + 1)
start(AIVoice);
[d, t] = getdata(AIVoice, returndata);
% Set up data plots
plot(t, d);
xlabel('Time (sec.)');
ylabel('Signal Level (Volts)');
title('Voice Activation for Both Triggers');
grid on;
```

The data is shown below.



The following code illustrates how you can find the relative trigger times, measured in fractions of seconds, from the absolute trigger times.

```
abstimes = AIVoice.TriggerTime;
reltimes = (abstimes-abstimes(1))*3600*24
reltimes =
         0
    0.5980
```

Another way to find the relative trigger times is to search for NaNs in the returned data. You can find the index location of the NaN in `d` or `t` using the `isnan` function.

```
index = find(isnan(d))
index =
       22051
```

With this information, you can find the relative time that the second trigger was issued.

```
t2time = t(index+1)
t2time =
    0.5980
```

Configuring Triggers for National Instruments Hardware

When using National Instruments (NI) hardware, there are additional trigger types and trigger conditions that you must be aware of. These device-specific property values are described below and in Appendix A.

Note: Refer to current National Instruments hardware documentation for detailed information about these triggering choices.

Trigger Types and Trigger Conditions

The additional trigger types and trigger conditions for analog input objects associated with National Instruments hardware fall into two main types: hardware digital triggering and hardware analog triggering. These trigger types and the associated trigger conditions are described below.

Table 4-7: TriggerType and TriggerCondition Values for NI Hardware

TriggerType	TriggerCondition	Description
HwDigital	None	The trigger occurs when a digital (TTL) signal is detected.

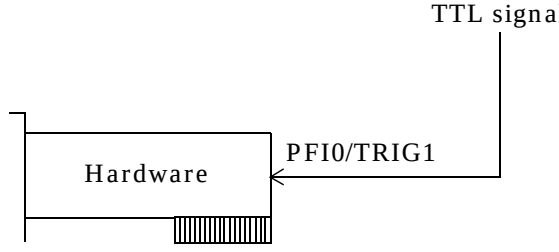
Table 4-7: TriggerType and TriggerCondition Values for NI Hardware (Continued)

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.

Hardware Digital Triggering. If TriggerType is set to HwDigital, the trigger is given by a TTL signal. The code below illustrates how to configure such a trigger.

```
ai = analoginput('nidaq',1);
addchannel(ai, 0:7);
set(ai, 'TriggerType', 'HwDigital')
```

The diagram below illustrates how a digital trigger signal is connected to an AT-MIO/AI E series board.



AT-MIO/AI E Series

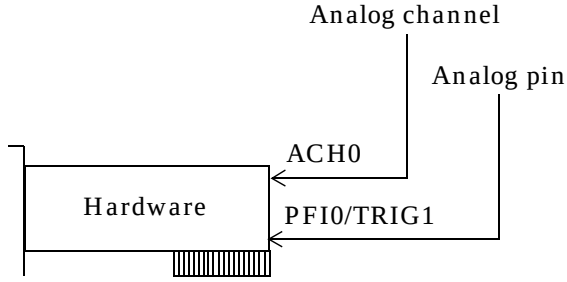
Hardware Analog Triggering. If `TriggerType` is set to `HwAnalogPin`, the trigger is given by a low-range analog signal (typically between -10 to 10 volts). The code below illustrates how to configure such a trigger.

```
ai = analoginput('nidaq',1);
addchannel(ai, 0:7);
set(ai, 'TriggerType', 'HwAnalogPin')
set(ai, 'TriggerCondition', 'InsideRegion')
set(ai, 'TriggerConditionValue', [-4.0 4.0])
```

If `TriggerType` is set to `HwAnalogChannel`, the trigger is given by an analog signal and the trigger channel is defined as the first channel of the channel list (MATLAB index of one). The valid range of the analog trigger signal is given by the full-scale range of the trigger channel. The code below 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.

```
ai = analoginput('nidaq',1);
addchannel(ai, 0:7);
set(ai.Channel(1), 'ChannelName', 'TrigChan')
set(ai, 'TriggerType', 'HwAnalogChannel')
set(ai, 'TriggerConditionValue', 0.2)
```

The diagram below illustrates how analog trigger signals can be connected to an AT-MIO/AI E series board.



AT-MIO/AI E Series

Configuring Events and Actions

Data acquisition tasks are based on *events*. An event occurs at a particular time after a condition is met and may result in one or more actions. The events associated with analog input objects are:

- **Data-Available Event**

  A block of data is available for processing in the MATLAB workspace
- **Timer Event**

  A predefined amount of time passed
- **Start or Stop Event**

  A `start` or `stop` function was issued
- **Trigger Event**

  A trigger occurred
- **Runtime Error Event**

  A runtime error occurred. Runtime errors include timeouts and hardware errors incurred while acquiring data. These errors are distinguished from configuration errors.
- **Over-Range Event**

  A channel group member experienced an over-range condition
- **Data-Missed Event**

  Data was missed by the engine

Recording and retrieving event information is described below.

Recording and Retrieving Event Information

For some of the events listed above, certain information is automatically stored in the `EventLog` property. `EventLog` is a structure that contains two fields: `Type` and `Data`. The `Type` field contains the event type. The event types that are logged for analog input objects are start, stop, trigger, runtime error, input over-range, and data missed events. Data-available events and timer events are not logged to `EventLog`.

The `Data` field contains the event-specific subfields shown below.

Table 4-8: EventLog Subfields

Events	Data Subfield	Description
All events	TimeStamp	The absolute time the event occurred.
	SampleStamp	The number of samples acquired when the event occurred. The sample count begins when the start function is issued. A value of -1 indicates the engine was not running when the event ocurred.
Trigger	Channel	Which channels caused the trigger to be issued (used only for software triggers).
	Trigger	The trigger number
Input over-range	Channel	The channels that experienced an input over-range condition
	Direction	Indicates if the over-range condition switched from “On” to “Off” or from “Off” to “On”.
Data missed	Channel	The channel(s) from which data was missed.

The example below illustrates how you can retrieve event information.

Example: Retrieving Event Information

Suppose you want to examine the events logged for the voice activation example on page 4-32. You can do this by accessing the EventLog property.

```
events = AIVoice.EventLog
```

By examining the contents of the Type field, you can list the events that occurred during this acquisition.

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

Note: Unless a runtime error occurs, all acquisitions must have a start, trigger, and stop event.

The Data field of the trigger event recorded the absolute time the trigger occurred, the number of samples acquired since the start event occurred, the index of the trigger channel, and the trigger number.

```
trigdata = events(2).Data
trigdata =
      TimeStamp: 7.3001e+005
    SampleStamp: 19342
        Channel: 1
        Trigger: 1
```

Action Properties and Functions

When an event occurs, you can execute a related function known as an *action function*. Action functions are M-files that can perform essentially any task during your data acquisition session. For example, you can use action functions to process data, display data, or display a message. All events have an associated action property.

You can specify the action function to be executed when an event occurs by including the name of an M-file as the value for the associated action property.

```
set(obj, 'Property', 'actionfcn')
```

where:

- `Property` is the action property that is automatically called when the associated event occurs.
- `actionfcn` is the name of the M-file to be executed when the event occurs. `actionfcn` can be an M-file that you create or the `start`, `stop`, or `trigger` functions included with the toolbox.

You can also specify additional parameters to be passed as input arguments to the action function. In this case, the action function and additional parameters must be specified as elements of a cell array.

```
set(obj, 'Property', {'actionfcn', arg1, arg2 ,..., argn});
```

The additional parameters can be any data type since they are elements of a cell array.

Action functions require at least two arguments. The first argument must be a device object. The second argument is a variable that captures the event

information contained by EventLog. This event information pertains only to the event that caused the action function to be executed. The action function prototype is shown below.

```
function actionfcn(obj, events)
```

If additional parameters are passed to the action function, then these arguments must be included after the two required arguments.

```
function actionfcn(obj, events, arg1, arg2 ,..., argn)
```

Note: If additional input parameters are passed to an action function, you are not required to use varargin. Instead, you can explicitly include the parameters in the function prototype as shown above.

Example action functions are given later in this section. The analog input properties related to action functions are shown below.

Table 4-9: Action-Related Properties

Property	Description
DataMissedAction	Specifies the M-file to be executed when the engine misses data
RuntimeErrorAction	Specifies the M-file to be executed when a runtime error occurs
SamplesAcquiredAction	Specifies the M-file to be executed when the number of samples specified by SamplesAcquiredActionCount is acquired
SamplesAcquiredActionCount	Sample-based frequency with which SamplesAcquiredAction is called
StartAction	Specifies the M-file to be executed just prior to the device and engine start
StopAction	Specifies the M-file to be executed just before the device and engine stop
TimerAction	Specifies the M-file to be executed when the time specified by TimerPeriod occurs

Table 4-9: Action-Related Properties

`TimerPeriod`	Time-based frequency with which `TimerAction` is called
`TriggerAction`	Specifies the M-file to be executed when a trigger occurs

When using action functions, you should be aware of these issues:

- All action functions are executed in the order in which they are issued.
- Depending on the action, all action functions except those related to timer events are quaranteed to execute.
- Action function execution may be delayed if the action is CPU-intensive such as updating a figure.
- Since a common application for time-based events is to display data, and displaying data is a CPU-intensive task, some of these events may not execute if your system is significantly slowed.

The relationship between events and action properties is described below for each event type.

Data-Available Event

A data-available event is generated immediately after a specified number of samples is acquired.

This event executes the M-file specified for the `SamplesAcquiredAction` property every time the number of samples specified by `SamplesAcquiredActionCount` is available. `SamplesAcquiredAction` should be used if you must access each sample that is acquired. For example, if you are processing all acquired data with an FFT, you can specify the M-file containing the FFT calculation with this property. Processing data this way may impact system performance.

Timer Event

A time-based event is periodically generated whenever a specified period of time has passed.

This event executes the M-file specified for the `TimerAction` property every time the number of seconds specified for the `TimerPeriod` property passes. Time is measured relative to when the hardware device starts running. The `TimerAction` property does not guarantee that all the data will be accessed.

Therefore, you should use `SamplesAcquiredAction` if all acquired data must be accessed.

Note: Since a common application for time-based events is to display data, and displaying data is a CPU-intensive task, some of these events may not execute if your system is significantly slowed.

Start and Stop Events

Start and stop events are described together since these two events occur for all data acquisition sessions. A start event is generated immediately after the `start` command is issued, while the stop event is generated immediately after the `stop` command is issued.

The start event executes the M-file specified for the `StartAction` property prior to starting the hardware and engine. Once this M-file has completed execution, the `Running` property is set to `On`.

The stop event executes the M-file specified for the `StopAction` property. Under most circumstances, the M-file is not guaranteed to complete execution until sometime after the hardware and engine stop and the `Running` property is set to `Off`.

Trigger Event

A trigger event is generated immediately after a trigger is issued.

This event executes the M-file specified for the `TriggerAction` property. Under most circumstances, the M-file is not guaranteed to complete execution until sometime after `Logging` is set to `On` by the engine.

Runtime Error Event

A runtime error event is generated immediately after a runtime error occurs. Runtime errors include hardware errors and timeouts. Runtime errors do not include configuration errors.

This event executes the M-file specified for the `RuntimeErrorAction` property.

Input Over-Range Event

An input over-range event is generated immediately after an over-range condition is detected for any channel group member. An over-range condition occurs when a signal exceeds the range specified for the `InputRange` property.

This event executes the M-file specified for the `InputOverRangeAction` property.

Note: Over-range detection occurs only when a value is specified for `InputOverRangeAction` and the analog input object is running.

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.

This event executes the M-file specified for the `DataMissedAction` property. By default, the `stop` command is issued.

Examples: Using Action Properties and Functions

Examples that show you how to create action functions and configure action-related properties are given below.

Displaying Event Information with an Action Function

This example illustrates how the action function prototype allows you to easily display event information. The example shown below uses `daqaction` to display information for trigger, runtime error, and stop events. The default `SampleRate` and `SamplesPerTrigger` values are used, which results in a one second acquisition for each trigger issued.

`daqdoc4_7.m`

Initialization: Create the analog input object `AI` for a sound card. The available adaptors are found with `daqhwinfo`.

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

Configuration: Add one hardware channel to AI, repeat the trigger three times, find the time for the acquisition to complete, and define daqaction as the M-file to execute when a trigger, runtime error, or stop event occurs.

```
chan = addchannel(AI, 1);
%chan = addchannel(AI, 0);
set(AI, 'TriggerRepeat', 3);
time = (AI.SamplesPerTrig/AI.SampleRate)*(AI.TriggerRep + 1);
set(AI, 'TriggerAction', 'daqaction');
set(AI, 'RuntimeErrorAction', 'daqaction');
set(AI, 'StopAction', 'daqaction');
```

Execution: Start AI and wait for it to stop running. The pause command allows the output from daqaction to be displayed.

```
start(AI)
pause(time + 1)
```

Termination: Delete AI.

daqdoc4_7.m

```
delete(AI)
```

The output is shown below.

```
Trigger event occurred at 10:17:57
Trigger event occurred at 10:17:58
Trigger event occurred at 10:17:59
Trigger event occurred at 10:18:00
Stop event occurred at 10:18:01
```

Specifying a Toolbox Function as an Action

You can specify the start, stop, or trigger toolbox functions as actions. This example illustrates how you can specify a toolbox function as an action. As

shown in the example code below, when an over-range condition occurs on the sound card channel, the acquisition is stopped.

```
AI = analoginput('winsound');
set(AI, 'SampleRate', 8000);
duration = 10; % Ten second duration
chan = addchannel(AI, 1);
ActualRate = get(AI, 'SampleRate');
set(AI, 'SamplesPerTrigger', duration*ActualRate);
set(AI, 'InputOverRangeAction', 'stop');
if strcmp(AI.Running, 'On')
end
start(AI)
```

Passing Additional Parameters to an Action Function

This example illustrates how additional arguments are passed to the action function, and generates timer events to display data. The M-file to execute is called `daqdoc4_8plot` and this function is executed every 0.5 seconds.

`daqdoc4_8.m`

Initialization: Create the analog input object `AI` for a sound card. The installed adaptors and hardware ID's are found with `daqhwinfo`.

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

Configuration: Add one hardware channel to AI, define a ten second acquisition, and execute the M-file daqug3_8plot every 0.5 seconds. Note that the variables size, P, and T are passed to the action function.

```
chan = addchannel(AI, 1);
%chan = addchannel(AI, 0);
set(AI, 'SampleRate', 8000);
duration = 10; % Ten second duration
ActualRate = get(AI, 'SampleRate');
set(AI, 'SamplesPerTrigger', duration*ActualRate);
set(AI, 'TimerPeriod', 0.5);
size = (AI.SampleRate)*(AI.TimerPeriod);
figure
P = plot(zeros(size,1));
T = title([sprintf('daqdoc4\\_8\nNumber of action functions
calls: '), num2str(0)]);
xlabel('Samples'); ylabel('Signal (Volts)')
grid on
set(gcf, 'doublebuffer', 'on');
set(AI, 'TimerAction', {'daqdoc4_8plot', size, P, T});
```

Execution: Start AI. The drawnow command in daqdoc4_8plot forces MATLAB to update the display.

```
start(AI)
pause(duration + 0.5)
```

Termination: Clear the action function from the workspace and delete AI.

daqdoc4_8.m

```
clear daqdoc4_8plot
delete(AI)
```

The function daqdoc4_8plot is given below.

```
function daqdoc4_8plot(obj, event, blocksize, plothandle,
titlehandle)
persistent index
if isempty(index)
   index = 0;
end
index = index + 1;
if ~ishandle(titlehandle)
   title([sprintf('daqdoc4\\_8\nNumber of action function
   calls: '), num2str(index)])
   data = getdata(obj, blocksize);
else
   set(titlehandle, 'String',[sprintf('daqdoc4\\_8\nNumber of
   action function calls: '), num2str(index)])
end
data = getdata(obj, blocksize);
size(data);
set(plothandle, 'ydata', data)
drawnow
```

Note: With action functions, you can perform essentially any data acquisition task. However, if you repeatedly display information to the MATLAB command line, you will usually not be able to issue the stop command. You can verify this by removing the semicolon from the size(data) command and running daqdoc4_8.

Engineering Unit Conversion

The Data Acquisition Toolbox provides you with a method for converting analog input signals into values which represent specific engineering units (Newtons, degrees Celsius, etc.). When specifying engineering units (EU), there are three aspects to consider:

- The sensor range
- The A/D range
- The engineering units range.

You can set engineering unit conversions for each channel of data. Therefore, the properties related to EU conversions can be set on a per-channel basis. These properties are given below.

Table 4-10: Engineering Units Properties

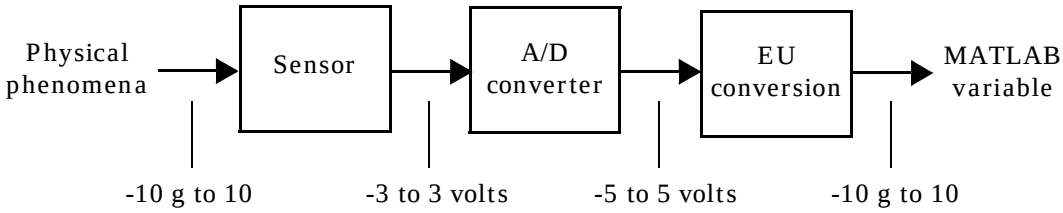
Property	Description
`SensorRange`	Expected range of data from sensor
`InputRange`	Range of A/D converter
`Units`	Engineering units name
`UnitsRange`	Range of data in MATLAB workspace

Note: Only linear engineering unit conversions are supported. You can perform nonlinear conversions by creating the appropriate M-file function.

The `InputRange` property is often expressed in terms of a device's gain and polarity. If you set `InputRange` to a particular value, then gain and/or polarity will change accordingly. For example, if an analog input subsystem has a built-in voltage range of 10 volts, and if `InputRange` is set to `[0 10]`, then polarity is automatically set to unipolar, but gain is unchanged.

Example: Performing a Linear Conversion

Suppose your task is to measure acceleration with an accelerometer. The sensor can measure acceleration up to +-10g (1 g = 9.80 m/s/s) and this range produces an output voltage from -3 to 3 volts. In addition, assume that you will use the full range of the accelerometer and the gain and polarity of your data acquisition hardware can be configured to match the sensor's output range. Furthermore, you want to display the acquired data in units of g's. This application is illustrated in the figure below.

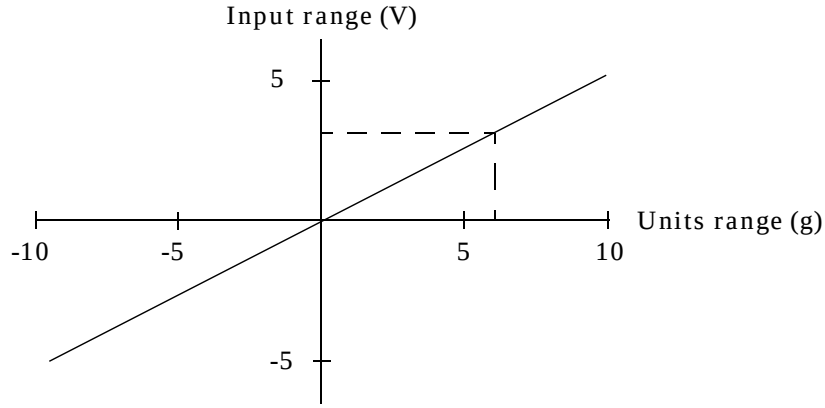


This configuration is summarized below.

Table 4-11: Engineering Units Configuration

Name	Property	Property Value
Sensor range	`SensorRange`	`[-3 3]`
A/D range	`InputRange`	`[-5 5]`
Units range	`UnitsRange`	`[-10 10]`

The figure below shows the conversion between the input range of the A/D converter and the units range.

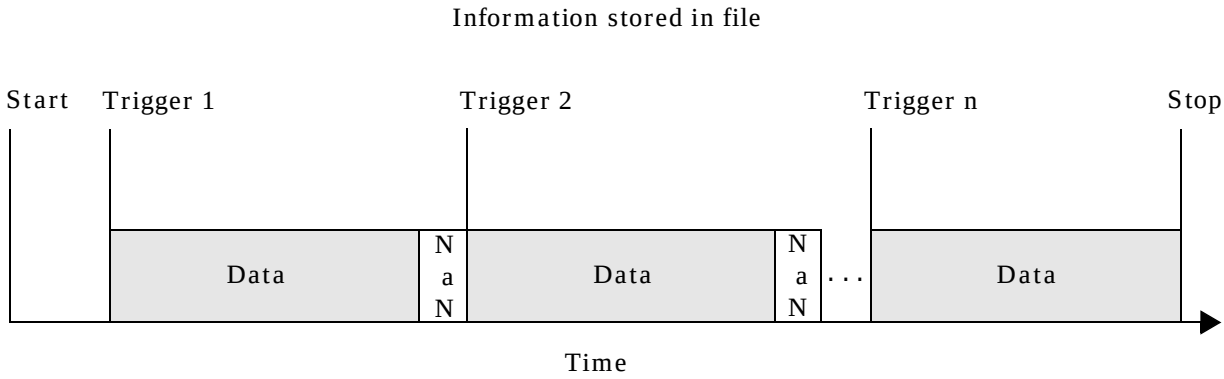


The code snippet below illustrates how you might configure the engineering units properties for a National Instruments board.

```
AI = analoginput('nidaq',1)
addchannel(AI, 1);
set(AI, 'SampleRate', 10000);
set(AI.Channel,'SensorRange', [-5 5]);
set(AI.Channel,'InputRange', [-5 5]);
set(AI.Channel,'UnitsRange', [-10 10]);
set(AI.Channel,'Units', 'g''s (1g = 9.80 m/s/s)');
```

Logging Information to Disk

The information logged to a file can be thought of as a stream of data and events as shown below.



The properties related to logging analog input information to disk are given below.

Table 4-12: Logging Properties

Property	Values	Description
LogFileName	string	Output log file name
Logging	On|{Off}	Determines if data is being logged or not
LoggingMode	Disk|{Memory}|Disk&Memory	Specifies where data is logged
LogToDiskMode	{Overwrite}|Index	Controls how disk file writes are performed

You can log data, events, hardware information, and engine information to a disk file by specifying a name for the LogFileName property and setting LoggingMode to Disk or Disk&Memory.

You can choose whether an output file is overwritten or if multiple log files will be 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`. A new file is created for each `start` command issued and each analog input device object must be logged to a separate file.

Disk writes are performed as soon as possible after the current data block is filled.

Specifying an Output File Name

The output file naming scheme follows these rules:

- You may 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` will be used as the extension.

Additionally, if `LogToDiskMode` is set to `Index`, then the output filename also follows these rules:

- If no number is specified as part of `LogFileName`, then two digits will be appended starting at "01". For example, if `LogFileName` is specified as `Name.daq`, then `Name01.daq` is the name of the first output file.
- If a number is specified as part of `LogFileName`, then the number is incremented by one for each successive output file.

If a runtime error occurs during data logging, the engine will return an error message and stop data logging. Possible errors include the disk filling up, or if `LogToDiskMode` is set to `Index` and the output file specified by `LogFileName` already exists.

Retrieving Logged Information

You can retrieve data, time, events, and engine and hardware information from an existing log file using the `daqread` function.

Any part or all of the information stored in a log file can be returned with one call to `daqread`. In some cases you may want to issue separate calls to `daqread` to return sample-time pairs, and event, hardware, and engine information. Therefore, the syntax for these two function calls is discussed in separate sections below.

Retrieving Data and Time Information

The logged data can be characterized by the sample number or the time the sample was acquired. Data and time information can be returned with the command

```
[data, time, abstime] = daqread('file', 'P1, V1, 'P2', V2,...);
```

where:

- `data` is the logged data
- `time` (optional) is the relative time associated with the returned data
- `abstime` (optional) is the absolute time the triggers were issued. `abstime` is returned as a `datenum` value and can be converted to a string using `datestr`.
- `file` is the name of the log file.
- `'P1', V2, 'P2', V2,...`(optional) are the property name/property value pairs listed below.

Property	Data Type	Description
`Samples`	MATLAB vector	Specifies the sample range
`Time`	MATLAB vector	Specifies the relative time range.
`Triggers`	MATLAB vector	Specifies the trigger range.
`Channels`	MATLAB vector or cell array	Specifies the channel range. Channel names can be specified as a cell array.
`DataFormat`	{`double`}|`native`	Specifies the data format as doubles or native.
`TimeFormat`	{`vector`}|`matrix`	Specifies the time format as a vector or a matrix.

The `Samples`, `Time`, and `Triggers` properties are mutually exclusive. `Time` must be specified as a `datenum` value. This allows event times to be used when specifying the `Time` range. If none of these three properties are specified, then all the data will be returned.

The `daqread` call shown above reads the specified data from `file` and returns an m-by-n matrix to `data`, where *m* specifies the number of samples and *n*

specifies the number of channels. If data from multiple triggers is read, each trigger is separated by a NaN. The amount of data returned and the format of that data is specified with property/value pairs given in the preceding table.

Retrieving Event, Object, and Hardware Information

You can retrieve event, object, and hardware information along with data and time information using the syntax shown below.

```
[data, time, abstime, events, daqinfo] = daqread('file', 'P1',
V1,...);
```

`events` is a structure containing event information associated with the logged data and `daqinfo` is a structure containing object and hardware information. All other arguments are described in the previous section.

`daqinfo` contains two fields: `ObjInfo` and `HwInfo`. `ObjInfo` is a structure containing property information for the device object and any channels it contains. `HwInfo` is a structure containing hardware information. The hardware information returned is identical to that given by `daqhwinfo(obj)`.

`events` contains the appropriate event structure based on the samples, triggers or time specified, and is identical to the information returned by `daqinfo.ObjInfo.EventLog`. The entire event log is returned only if samples, time, and triggers are not specified.

Alternatively, you can return only object and hardware information with the command

```
daqinfo = daqread('file', 'info');
```

The entire event log is returned to `daqinfo.ObjInfo.EventLog`.

Note: You cannot call `daqread` if the specified file is open.

Example: Logging and Retrieving Information

This section illustrates how to log information to a disk file and then return the logged information to MATLAB using various calls to `daqread`.

In the example below, a sound card is configured for stereo acquisition, data is logged to memory and to a disk file, four triggers are issued, and two seconds

of data is collected for each trigger at a sample rate of 8 kHz. All the logged data is then read from the file as sample-time pairs.

daqdoc4_9.m

Initialization: Create the analog input object `ai` for a sound card. The available adaptors are found with `daqhwinfo`.

```
ai = analoginput('winsound');
%ai = analoginput('nidaq', 1);
```

Configuration: Add two hardware channels to `ai`, define a two second acquisition for each trigger, set the trigger to repeat three times, and log data and events to the file `file00.daq`.

```
ch = addchannel(ai, 1:2);
%ch = addchannel(ai, 0:1);
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, 'LogToDiskMode', 'Overwrite');
set(ai, 'LoggingMode', 'Disk&Memory');
```

Execution: Start `ai`, wait for `ai` to stop running, and extract all the data stored in the engine as sample-time pairs.

```
start(ai)
while strcmp(ai.Running, 'On')
end
[data, time] = daqread('file00.daq');
```

Termination: Plot the data and delete `ai`.

```
subplot(211); plot(data);
title(sprintf('daqdoc4\\_9\nLogging and Retrieving Data'))
xlabel('Samples'); ylabel('Volts')
subplot(212); plot(time, data);
xlabel('Time (seconds)'); ylabel('Signal (Volts)')
delete(ai)
```

daqdoc4_9.m

Returning Data Based on Samples

You can return data based on samples using the `Samples` property. The `daqread` call shown below returns samples 1000 to 2000 for both sound card channels.

```
[data, time] = daqread('file00.daq', 'Samples', [1000 2000]);
figure; subplot(211); plot(data);
xlabel('Samples'); ylabel('Volts')
subplot(212); plot(time, data);
xlabel('Time (seconds)'); ylabel('Volts')
```

Returning Data Based on Channels

You can return data based on channels using the `Channels` property. The `daqread` call shown below returns samples 1000 to 2000 for the second sound card channel.

```
[data, time] = daqread('file00.daq', 'Samples', [1000 2000],
'Channels', 2);
figure; subplot(211);plot(data);
xlabel('Samples'); ylabel('Volts')
subplot(212); plot(time, data);
xlabel('Time (seconds)'); ylabel('Volts')
```

Alternatively, you can specify the channel name.

```
[data, time] = daqread('file00.daq', 'Samples', [1000 2000],
'Channels', {'Right'});
```

Returning Data Based on Triggers

You can return data based on triggers using the `Triggers` property. The `daqread` call shown below returns all the data associated with the second and third triggers for both sound card channels.

```
[data, time] = daqread('file00.daq', 'Triggers', [2 3]);
figure; subplot(211); plot(data);
xlabel('Samples'); ylabel('Volts')
subplot(212); plot(time, data);
xlabel('Time (seconds)'); ylabel('Volts')
```

Returning Data Based on Time

You can return data based on time using the Time property. The values specified for Time must be datenum values.

For this example, the first 25% of the data acquired for the first trigger will be returned. The first Time value corresponds to the first trigger event and can be found with the EventLog property.

```
events = ai.EventLog
data = events(2).Data;
time1 = data.TimeStamp;
```

The second Time value is constructed by adding one-half second to the first Time value. Note that when a number is added to a datenum value, it must be normalized as shown below.

```
time2 = time1 + 0.5/(24*60*60);
```

You can now return the first 25% of the data acquired for the first trigger.

```
[data, time] = daqread('file00.daq', 'Time', [time1 time2]);
figure; subplot(211); plot(data);
xlabel('Samples'); ylabel('Volts')
subplot(212); plot(time, data);
xlabel('Time (seconds)'); ylabel('Volts')
```

Returning Event, Object, and Hardware Information

As shown on page 4-65, you can return event, object, and hardware, information by specifying the appropriate output arguments to daqread. To return all event information, you must return all the logged data.

```
[data, time, abstime, events, info] = daqread('file00.daq');
{events.Type}
ans =
'Start' 'Trigger' 'Trigger' 'Trigger' 'Trigger' 'Stop'
```

If part of the data is returned, then only the events associated with the requested data are returned.

```
[data, time, abstime, events, info] = daqread('file00.daq',
'Trigger', [1 3]);
{events.Type}
ans =
'Trigger' 'Trigger' 'Trigger'
```

Alternatively, you can return the entire event log as well as object and hardware information by including `'info'` as an input argument to `daqread`.

```
daqinfo = daqread('file00.daq', 'info')
hwinfo = daqinfo.HwInfo
objinfo = daqinfo.ObjInfo
eventinfo = daqinfo.ObjInfo.EventLog
```

5

Analog Output

Overview

Analog output (AO) devices convert digital data stored on the computer to a real-world analog signal. These devices perform the inverse conversion of analog input (AI) devices. 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.

The functionality related to the analog output process allows you to send digitally represented data out of the computer system as an analog output signal. The Data Acquisition Toolbox provides access to analog output devices through an analog output object.

The purpose of this chapter is to show you how to perform analog output data acquisition tasks. In presenting these tasks, only the properties and functions that are specific to analog output objects are presented. If at any time you want more information about any property or function, you can refer to the appropriate reference section in this book or the command line help.

To learn more about basic toolbox functionality, you should refer to Chapter 2, “Data Acquisition Toolbox Concepts.”

Connecting to Your Hardware

Before data can be output from MATLAB to yout D/A converter, you must "connect" the Data Acquisition Toolbox to your hardware. Connecting to your hardware requires two actions: registering the hardware driver adaptor and creating an analog output object. Registering the hardware driver adaptor is described on page 3-3. Creating an analog output object is described below.

Creating an Analog Output Object

Before executing an analog output (AO) task, you must create an analog output object.

```
AO = analogoutput('adaptor', ID);
```

where:

- `analogoutput` is the analog output object creation function.
- `adaptor` is the hardware driver adaptor name.
- `ID` is the vendor-specific device label.
- `AO` is the analog output object.

The valid input argument values for the analog output creation function are given below.

Table 5-1: Analog Output Object Creation Function Argument Values

Value	Description	Supported Drivers		
		Sound Cards	NI	HP VXI
`adaptor`	Adaptor name	`winsound`	`nidaq`	`hpvxi`
`ID`	Device label	N/A	Device number	Device id

If the hardware driver adaptor has been successfully registered by the data acquisition engine, then you can create an analog output object. Once the device object is created, a unique analog output object name is automatically created and assigned to the `Name` property.

By default, no hardware channels are associated with a newly created analog output object. To associate channels with the device object, you must use the `addchannel` function.

Example: Create an Analog Output Object

Suppose you want to create the analog output object `AO` for a sound card. Since there is no device label for the sound card, the analog output object creation function requires only the adaptor name as an argument.

```
AO = analogoutput('winsound');
```

An analog output object called `AO` now exists in the MATLAB workspace and the engine. You can display the device object's name using

```
get(AO, 'Name')
ans =
winsound0-AO
```

You cannot yet use `AO` to output data since it has no hardware channels associated with it. Additionally, you must set property values to establish the required behavior for your specific application. Adding channels to an analog output object is described below.

Adding Channels to an Analog Output Object

After creating the analog output object, you must now add channels to it. As shown in the figure on page 2-10, a device object can be thought of as a "container" for channels. Adding channels to the analog output object is accomplished with the `addchannel` function.

```
chans = addchannel(obj, hwch, index, names)
```

where:

- `obj` is the analog output object.
- `hwch` are the IDs of the hardware channels you are adding to the analog output object. Any valid MATLAB vector syntax can be used to specify the hardware channels.
- `index` (optional) are the MATLAB indices of the hardware channels.
- `names` (optional) are the descriptive names of the hardware channels.
- `chans` (optional) is a vector containing the analog output channels.

Note: Adding channels to a sound card involves certain device-specific restrictions. These restrictions are explained in the example on page 3-7.

The collection of channels contained by the device object is referred to as a *channel group*. Channel group members must all reside on the same hardware device and the channels are all sampled at the same rate. It is important to remember that when you add channels to an object, you are mapping physical hardware channels on your analog output hardware device to MATLAB indices. This mapping allows a certain amount of flexibility in terms of ordering the channels as described by these rules:

- Hardware channels can be assigned to multiple analog output objects.
- Hardware channels can be assigned multiple times to the same analog output object.
- Channel indices start at 1 and increase monotonically up to the number of channel group members.
- Hardware indices begin at 0 for National Instruments hardware, whereas for sound cards and Hewlett-Packard hardware, they begin at 1.

If no indices are specified with `addchannel`, they are automatically defined such that the indices are monotonically increasing and contain no gaps. Each channel is referenced by its position in the list (its index), or optionally by its name if one is defined. Named channels are discussed in "Reference by Channel Name" on page 4-8. For detailed information about `addchannel`, refer to Chapter 6, "Function Reference."

Example: Adding Channels for National Instruments Hardware

Suppose you create the analog output object `ao` for a National Instruments board having a device ID of 1.

```
ao = analogoutput('nidaq', 1)
```

To perform any data acquisition task, you must add hardware channels to the device object. Many National Instruments E-series boards have two output channels that can be added. To add two channels to `ao`:

```
addchannel(ao, 0:1)
```

These channels are automatically assigned the MATLAB indices 1 and 2.

Note that the channel scanning order is given by the MATLAB indices. Therefore, hardware channel 0 is sampled first and hardware channel 1 is sampled second.

Example: Adding Channels for a Sound Card

Suppose you create the analog output object ao for a sound card.

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

To perform any data acquisition task, you must add hardware channels to the device object. Most sound cards have just two output channels that can be added. If one channel is added, the sound card is said to be in *mono* mode. If two channels are added, then the sound card is said to be in *stereo* mode. The rules for adding these two channels differ from those of other data acquisition devices. These rules are described on page 3-7.

Controlling Output Behavior with Properties

After the analog output object is created and hardware channels have been added, the object must be configured to define its behavior. Configuring a device object involves assigning values to properties. In general, you can think of a property as a characteristic of an object that governs its behavior. In a more practical sense, a property is anything that can be set using the `set` command or subscripted assignment.

The properties that need to be set depend on your particular data acquisition application. However, for many common analog output applications, there is a core group of properties that must be set. This core group is divided into two subgroups:

- Properties related to the basic setup
- Properties related to the data range and engineering units.

These subgroups are described below.

Basic Setup

For most data acquisition applications, there is a set of properties used to configure the basic setup for your session. These properties provide access to individual channels, control the sampling rate, and define the trigger type. Properties related to the basic setup are given below.

Table 5-2: Analog Output Basic Setup Properties

Property	Description
`Channel`	Contains all the channels associated with the device object as created with `addchannel`
`SampleRate`	Specifies the samples per second of the analog input hardware device
`TriggerType`	Specifies the type of trigger to be issued

Accessing individual channels, setting the sampling rate, and defining a trigger are discussed below.

Accessing Individual Channels

As discussed in "Property Names and Property Values" on page 2-15, the 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.

`Channel` contains all the channels associated with an analog output object as created with `addchannel`. Using `Channel`, you can access individual channels to:

- Configure property values for individual channels
- Display the possible values for each channel property with the `set` command
- Display the current value for each channel property with the `get` command

Using `Channel` to configure channel properties is illustrated in "Data Range and Engineering Units" on page 5-9.

Setting the Sampling Rate

The rate that data is output per channel is set with the `SampleRate` property. `SampleRate` must be specified as samples per second. For example, to set the sampling rate of your sound card to 44,100 samples per second (44.1 kHz)

```
set(ao, 'SampleRate', 44100);
```

Data acquisition boards typically have predefined sampling rates that can be set. If you specify a sampling rate that does not match one of these predefined values but is within the range of valid values, then the toolbox automatically selects a valid sampling rate. If you specify a sampling rate that is outside the range of valid values, then an error is returned. The rules applied by the toolbox to select a valid sampling rate are described in Chapter 7, "Property Reference."

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

```
ActualRate = get(ao, 'SampleRate')
```

The default value for `SampleRate` is 8 kHz for most sound cards. For other hardware devices, the default value is supplied by the hardware driver.

Defining a Trigger

For analog output objects, a trigger is defined as an event that initiates sending (outputting) data to the D/A converter. The sending status is indicated by the `Sending` property. When a trigger occurs, `Sending` is set to `On`.

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 5-3: TriggerType Values and Description

Values	Description
`{Immediate}`	The trigger occurs just after the `start` command is issued. This is the default value.
`Manual`	The trigger occurs just after you manually issue the `trigger` function.

For some hardware, additional trigger types may be available.

Data Range and Engineering Units

Properties associated with engineering units allow you to control the range of data output to your D/A converter and the valid ranges that it can be set to. When data is output to a D/A converter, you must be aware of these two issues:

- The range of the data you are outputting to the D/A converter.
- The valid range(s) that your D/A hardware can be set to. For many devices, the D/A hardware range is specified by the gain and polarity.

You should select the data and D/A hardware ranges such that the output data is not clipped, and you use the largest possible dynamic range of the output hardware.

The output data range is set with the `UnitsRange` property, and the D/A range is set with the `OutputRange` property, These are both channel properties and can be set on a per-channel basis. The range set for `UnitsRange` maps the data to the range set for `OutputRange` thereby scaling the output data accordingly.

For example, suppose you are outputting data to one analog output channel. The output data ranges between -2 and 2 volts and the D/A converter has a

maximum range of -1 volt to 1 volt. If the data is not scaled, it will be clipped since its range exceeds that of the D/A hardware. To solve this problem, you must use the `UnitsRange` property. Setting `UnitsRange` to `[-2 2]` scales the output data such that a value of -2 maps to -1 at the hardware level and a value of 2 maps to 1 at the hardware level. The code below illustrates how to configure the appropriate properties.

```
ao = analogoutput('winsound')
addchannel(ao, 1);
ch = ao.Channel(1);
set(ao, 'SampleRate', 8000);
ActualRate = get(ao, 'SampleRate');
data = 2*sin(0:(2*pi/ActualRate):2*pi)';
set(ch, 'UnitsRange', [-2 2]);
set(ch, 'OutputRange', [-1 1]);
```

Setting the data range and engineering units is described in detail in "Engineering Unit Conversion" on page 5-38.

Outputting Data

Outputting data involves functions that execute the analog output object, and functions that queue the data in the engine. These functions are described below.

Starting the Output

To start the device object and data acquisition engine

```
start(ao)
```

Starting the output process means that the hardware device and data acquisition engine are both running. Running does not necessarily mean that data is being output to the D/A converter. For that to occur, a trigger must be issued. Triggers are described on page 5-9 and page 5-28.

When a `start` command is issued, the `Running` property is automatically set to `On`. If the device is already running when `start` is issued, then an error is returned.

Queueing Data for Output

After the trigger executes, you can output data to the D/A converter. For data to be output, it must first be queued in the engine with the `putdata` function.

```
putdata(ao, data);
```

`data` is an *m*-by-*n* array where *m* is the number of samples to output and *n* is the number of output channels.

You can use `putdata` to queue data either before or after the `start` command is issued. `putdata` will not return execution control to MATLAB until the requested samples are queued. Additionally, if data is queued before `start` is issued, then `putdata` cannot be called again until the queued data is finished outputting. `putdata` is described in detail in "Managing Output Data" on page 5-24 and in Chapter 6, "Function Reference."

Stopping the Output

The analog output object can stop under these conditions:

- A runtime error occurs.
- The specified number of samples is output.
- The `stop` command is issued.

When the output has stopped, the `Running` and `Sending` properties are automatically set to their default value `Off`.

Deleting the Device Object

When the output task is completed and the device object is no longer needed, you should delete the device object to free memory and other physical resources. You can delete the analog output object with the `delete` command.

```
delete(ao)
```

`delete` removes the device object from both the data acquisition engine and the MATLAB workspace. If this was the last object accessing the hardware, the associated hardware driver is closed and unloaded.

Note: You cannot delete a device object using the command form of delete `delete ao`.

Analog Output Examples

This section illustrate how to perform basic data acquisition tasks using analog output subsystems and the Data Acquisition Toolbox. In doing so, many of the properties and functions presented in this chapter are used.

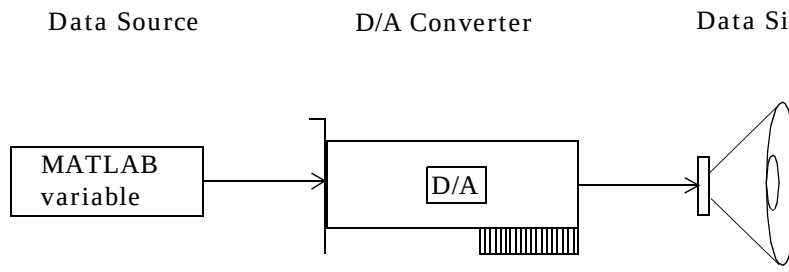
For most data acquisition applications using analog output subsystems, the same basic steps must be followed. These basic steps include:

- Installing and connecting the components of your data acquisition hardware. At a minimum, this involves connecting an actuator to a plug-in or external data acquisition device.
- Configuring your data acquisition session. This involves 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.



As described in “The Data Acquisition Session” on page 2-24, using the Data Acquisition Toolbox to access the capabilities of your D/A hardware involves these steps:

- Create an analog output object
- Add hardware channels to the object

- Set property values
- Execute the object
- Output data to the D/A hardware

For this example, one hardware channel is added to the analog output object, two seconds of data is queued in the engine with two putdata calls, and a manual trigger is issued to initiate the output. The complete data acquisition session for the sound card is shown below.

daqdoc5_1.m

Initialization: Create the analog output object AO for a sound card. The installed adaptors and hardwrae ID's are found with daqhwinfo.

```
AO = analogoutput('winsound');
```

Configuration: Add one channel to AO, 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.

```
chans = addchannel(AO, 1);
duration = 4;
set(AO, 'SampleRate', 8000);
set(AO, 'TriggerType', 'Manual');
set(AO.Channel(1), 'UnitsRange', [-2 2]);
ActualRate = get(AO, 'SampleRate');
len = ActualRate*duration;
data = 2*sin(linspace(0, 2*pi, len))';
putdata(AO, data);
```

Execution: Start AO, issue a manual trigger, and wait for the device object to stop running.

```
start(AO);
trigger(AO);
while strcmp(AO.Running, 'On')
end
```

Termination: Delete AO.

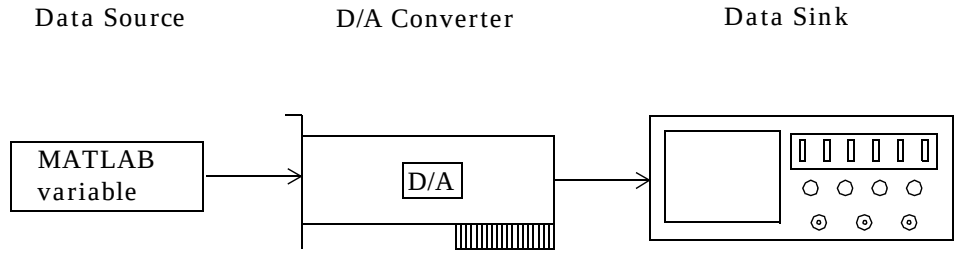
daqdoc5_1.m

```
delete(AO)
```

Note: You must add channels to `AO` before queueing data with the first `putdata` call.

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.



As described in "The Data Acquisition Session" on page 2-24, using the Data Acquisition Toolbox to access the capabilities of your D/A hardware involves these steps:

- Create an analog output object
- Add hardware channels to the object
- Set property values
- Execute the object
- Output data to the D/A hardware

For this example, one hardware channel is added to the analog output object, two seconds of data is queued in the engine with two `putdata` calls, and a manual trigger is issued to initiate the output. The complete data acquisition session for the sound card is shown below.

daqdoc5_2.m

Initialization: Create the analog output object AO for a National Instruments board. The installed adaptors and hardware ID's are found with daqhwinfo.

```
AO = analogoutput('nidaq',1);
```

Configuration: Add one channel to AO, 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.

```
chans = addchannel(AO, 0);
duration = 4;
set(AO, 'SampleRate', 10000);
set(AO, 'TriggerType', 'Manual');
set(AO.Channel(1), 'UnitsRange', [-2 2]);
ActualRate = get(AO, 'SampleRate');
len = ActualRate*duration;
data = 2*sin(linspace(0, 2*pi, len))';
putdata(AO, data);
```

Execution: Start AO, issue a manual trigger, and wait for the device object to stop running.

```
start(AO);
trigger(AO);
while strcmp(AO.Running, 'On')
end
```

Termination: Delete AO.

daqdoc5_2.m

```
delete(AO)
```

Evaluating Your Acquisition Status and Resources

At any time after a device object is created, you can evaluate the status of your data acquisition session by:

- Returning the values of certain properties
- Invoking the display summary

The properties related to evaluating your acquisition status and the display summary are discussed below.

Acquisition Status Properties

Acquisition status properties allow you to evaluate whether the toolbox is running or outputting data, as well as how much total data has been output and how much data is queued in the engine. The properties related to evaluating your acquisition status are given below.

Table 5-4: Analog Output Acquisition Status Properties

Property	Description
`Running`	Indicates whether the hardware device and data acquisition engine are running
`SamplesAvailable`	Indicates the number of samples available per channel in the data acquisition engine
SamplesOutput	Indicates the number of samples per channel that have been sent to the hardware device
`Sending`	Indicates data is being output to the hardware device

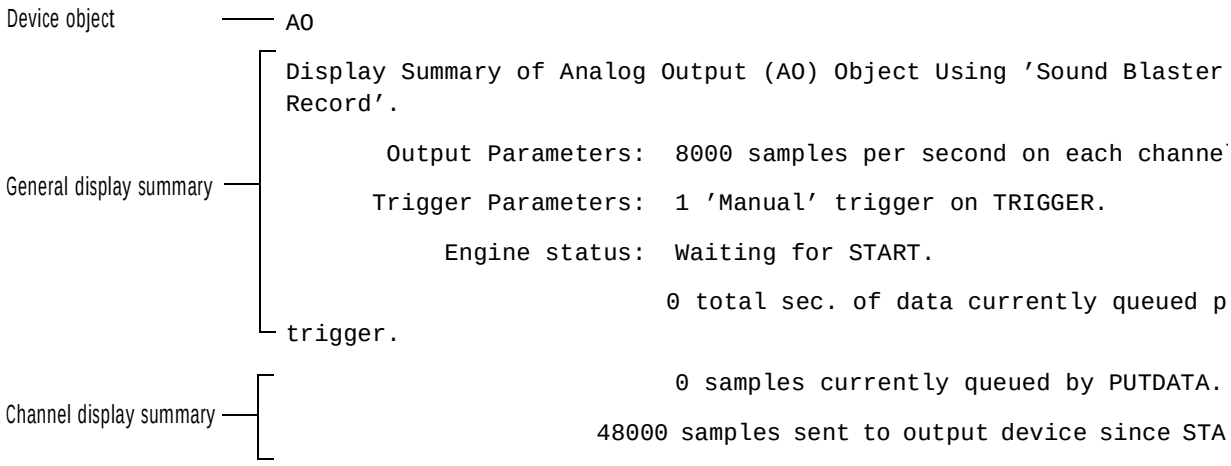
Display Summary

You can invoke the display summary by typing the name of the device object at the command line. The information displayed reflects many of the core properties described on “Controlling Output Behavior with Properties” on page 5-7, and is designed so you can quickly evaluate the status of your data acquisition session. The displayed information is divided into two sections:

- General summary information
- Information about each channel contained by the device object

General summary information includes the device name, the sampling rate, the acquisition duration, the trigger type, the engine status (whether it is logging data, waiting for a trigger, etc.), the number of samples queued in the engine, and the number of samples output since starting. Channel information includes the hardware channel mapping, the channel name, and property values related to engineering units.

The display summary for the example on page 5-13 is shown below.



The device object and channel information reflects the acquisition status after the queued samples were output.

If you want to display only the channel summary information, you must use the `Channel` property.

```
AO.Channel
```

Hardware Information

You can display hardware-related information with the `daqhwinfo` function. This function is described in "Evaluating Your Acquisition Status" on page 3-24.

Getting Function and Property Help

You can access help on Data Acquisition Toolbox functions and properties using some or all of these resources:

- The reference sections in this book
- M-file help
- HTML-based property help available through the Help Desk
- The `daqhelp` function
- The `propinfo` function

For more information about getting property and function help, refer to “Getting Function and Property Help” on page 3-28.

Configuring Hardware Properties

The hardware you are using contains functionality that satisfies your specific application needs. Typical hardware functionality includes device type (analog input, analog output, digital I/O), the number of channels or lines, and the maximum sample rate, among others. This section presents the base properties that allow you to configure your analog output hardware. These properties are shown below.

Table 5-5: Analog Output Hardware Configuration Properties

Property	Description
`DefaultChannelValue`	Value to output when a channel runs out of data
`OutOfDataMode`	Setting to use when the object runs out of data
`OutputRange`	Range of D/A converter
`SampleRate`	Sampling rate per channel (in Hz)

Except for `OutputRange`, these properties are described in detail below. `OutputRange` is described in "Engineering Unit Conversion" on page 5-38.

Setting and Verifying Hardware Properties

For the `OutputRange` and `SampleRate` properties, data acquisition devices have a finite (though sometimes large) number of valid values that you can set. If you specify a property value that does not match one of the valid device values, then the engine will choose a valid value automatically. The rules the engine uses to select a valid property value are described for each property in Chapter 7, "Property Reference."

You can use the `propinfo` function to obtain information about the valid values for these properties. For example, suppose you create the analog output object

AO for a sound card. You can use propinfo to display information about the SampleRate property.

```
AO = analogoutput('winsound')
propinfo(AO, 'SampleRate')
ans =
               Type: 'double'
         Constraint: 'Bounded'
    ConstraintValue: [8000 44100]
       DefaultValue: 8000
           ReadOnly: 0
    ReadOnlyRunning: 1
     DeviceSpecific: 0
```

To find out if the engine sets a property to a value that differs from the one you specified, you should get the value after it is set. For example, after setting the sampling rate for a sound card, you should retrieve the actual rate set since you may want to use this value to set additional property values.

```
AO = analogoutput('winsound')
addchannel(AO, 1)
set(AO, 'SampleRate', 8192)
ActualRate = get(AO,'SampleRate');
data = sin(0:(2*pi/ActualRate):2*pi)';
% Find the time required to output data
time = length(data)/ActualRate;
```

As an alternative to the syntax shown above, you can use the setverify function. setverify is equivalent to the commands

```
set(obj, 'Property', Value)
Actual = get(obj, 'Property')
```

Using setverify, the previous example is written as

```
AO = analogoutput('winsound')
addchannel(AO, 1)
ActualRate = setverify(AO,'SampleRate', 8192)
data = sin(0:(2*pi/ActualRate):2*pi)';
% Find the time required to output data
time = length(data)/ActualRate;
```

Setting the Sampling Rate

The sampling rate is given by the SampleRate property and is defined as the number of samples output per second for each hardware channel. The maximum rate that channels can be sampled at depends on the specific hardware you are using and is discussed on page 4-17.

If you are using simultaneous sample and hold (SSH) 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 output object AO for a sound card, configure it for stereo operation, and set the sampling rate to 44,100 Hz.

```
AO = analogoutput('winsound');
addchannel(AO, 1:2);
set(AO, 'SampleRate', 44100);
```

Both channels are sampled at 44,100 Hz.

Note: For some sound cards, you can set the sampling rate to any value between the minimum and maximum values defined by the hardware. This is due to onboard filtering.

Setting the Out-of-Data Value

When data has finished being output to a channel, you can choose the setting of the D/A hardware with the OutOfDataMode property. If OutOfDataMode is set to Hold, then the last output value is held. This is the default behavior for many analog output devices. If OutOfDataMode is set to Stop, the device is stopped and the value written to the D/A converter is device-dependent. For some devices, the D/A value drifts to zero, while for other devices, the last value is held until it is reset. If OutOfDataMode is set to DefaultValue, the value specified for DefaultChannelValue is output. If supported by the hardware, this value is also held. DefaultChannelValue is a channel property and can be configured on a per-channel basis.

For example, suppose you create the analog output object AO for a National Instruments board and add one channel to it. The code below illustrates how to

send a last value of 0.5 volts using `DefaultChannelValue` after queued data has finished being output.

```
AO = analogoutput('nidaq', 1);
ch = addchannel(AO, 1);
set(AO, 'SampleRate', 10000);
ActualRate = get(AO, 'SampleRate');
set(AO, 'OutOfDataMode', 'DefaultValue');
set(ch, 'DefaultChannelValue', 0.5);
data = sin(0:(2*pi/ActualRate):2*pi)';
putdata(AO, data);
start(AO)
```

Note: `OutOfDataMode` does not apply to sound cards.

Additional Hardware Properties

Depending on the hardware you are using, there may be additional hardware-related properties. To find these properties, you can:

- Type `set(obj)`. The hardware-specific properties are included at the bottom of the property list. For example, sound cards have a `BitsPerSample` property which allows you to select the number of bits used to represent each sample.
- Refer to Appendix A

Managing Output Data

Data that is to be output by an analog output object is managed with the putdata function. In addition to this function, there are several properties associated with managing output data. These properties are listed below.

Table 5-6: Data Management Properties

Property	Description
MaxSamplesQueued	Indicates the maximum number of samples that can be queued for output
RepeatOutput	Number of additional times the queued data is output
SamplesAvailable	Number of samples in the queue waiting to be output
SamplesOutput	Number of samples that have been output
Timeout	An additional time to wait (in seconds) for putdata to return. The timeout value is added to the time required to fill one data block in the engine.

Using putdata

To output data to the D/A hardware, you must use the putdata function. The syntax for putdata is

```
putdata(obj, source)
```

where:

- obj is the analog output object.
- source is the MATLAB variable containing the data to be output.

putdata outputs data specified by source to the D/A hardware. source is a MATLAB variable and must consist of a column of data for each channel group member. A data source for m samples and n channels is illustrated below.

$$\begin{bmatrix} d_{11} & d_{12} & & d_{1n} \\ d_{21} & d_{22} & & d_{2n} \\ d_{31} & d_{32} & \cdots & d_{3n} \\ \vdots & \vdots & & \vdots \\ d_{m1} & d_{m2} & & d_{mn} \end{bmatrix}$$

Data source. Each column represents one channel.

If output data is not within the range of the UnitsRange property, then the data is clipped to the maximum or minimum value specified by UnitsRange. An error is returned if a NaN is included in the data stream. The data contained by source can be the native data type of the hardware.

You can use putdata to queue data in memory either before or after the start command is issued. In either case, no data is output until the trigger executes.

If you call putdata before start is issued, then the specified data is queued in the engine unless:

- MaxSamplesQueued is reached. If this value is exceeded, an error is returned.
- The limitations of your hardware or computer are reached.

You can output the queued data repeatedly by using the RepeatOutput property. If RepeatOutput is greater than 0, then all data queued before start is issued will be requeued (repeated) the specified number of times. You can use multiple putdata calls to queue data.

Note: The value for RepeatOutput must be set before putdata is called and before start is called.

After the queued data is sent to the D/A hardware, the output process stops and data can once again be queued. While queued data is being output, a call to putdata results in an error.

If putdata is called after start is issued, then you cannot use RepeatOutput and the queue must initially be empty. If MaxSamplesQueued is exceeded, putdata becomes a blocking function until there is enough space in the engine to queue the additional data.

As soon as the trigger executes, the SamplesOutput property keeps a running count of the total number of samples per channel that have been output. Additionally, the SamplesAvailable property tells you how many samples are ready to be output from the engine per channel. When data is output, SamplesAvailable is reduced by the number of samples sent to the hardware.

Example: Using putdata

The example below illustrates how you can use putdata to queue 8000 samples, and then output the data a total of five times using the RepeatOutput property.

daqdoc5_3.m

Initialization: Create the analog output object AO for a sound card. The installed adaptors and hardware ID's are found with daqhwinfo.

```
AO = analogoutput('winsound');
%AO = analogoutput('winsound');
```

Configuration: Add one channel to AO, define an output time of one second, assign values to the basic setup properties, generate data to be queued, and issue two putdata calls. Since five triggers and two putdata calls are issued, a total of ten seconds of data is output.

```
chans = addchannel(AO, 1);
%chans = addchannel(AO, 0);
duration = 1;
set(AO, 'SampleRate', 8000);
ActualRate = get(AO,'SampleRate');
len = ActualRate*duration
set(AO, 'RepeatOutput', 4);
data = sin(linspace(0, 2*pi, len))';
putdata(AO, data);
putdata(AO, data);
```

Execution: Display summary information, start AO, and wait for the device object to stop running.

```
AO
start(AO);
while strcmp(AO.Running, 'On')
end
```

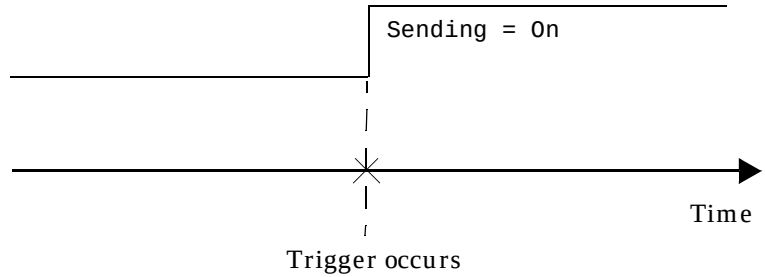
Termination: Delete AO.

daqdoc5_3.m

```
delete(AO)
```

Configuring Triggers

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 to the D/A converter.



Properties that allow you to configure an analog output trigger are shown below.

Table 5-7: Analog Output Trigger Properties

Property	Description
`TriggerAction`	M-file that is executed after the trigger is occurs
`TriggerTime`	Array of observed absolute trigger times
`TriggerType`	Source of the trigger

Except for `TriggerAction`, these trigger-related properties are discussed in the following section. `TriggerAction` is discussed in "Configuring Events and Actions" on page 5-30.

Trigger Types

Trigger types specify how data output is initiated and is specified with the `TriggerType` property. You can think of the trigger type as the source of the trigger.

The supported trigger types for analog output objects are given in "Defining a Trigger" on page 5-9.

The default TriggerType value is Immediate. An immediate trigger automatically occurs just after the start command is issued. If TriggerType is Manual, the trigger occurs just after you issue the trigger command. An example using a manual trigger is given on page 5-14. For some hardware, additional trigger types are available. Refer to Chapter 7, "Property Reference" for these device-specific values.

Trigger Times

You can keep track of the number of triggers issued and the absolute times they occurred with the TriggerTime property. To find out how many triggers occurred, you can use MATLAB's length command. For the example on page 5-14

```
NumTrigs = length(AO.TriggerTime)
NumTrigs =
     1
```

To find out the trigger times, you must convert the elements of the TriggerTime array from the datenum format to a string.

```
times = AO.TriggerTime;
time1_str = datestr(times(1), 13)
time1_str =
17:17:58
```

Configuring Events and Actions

Data acquisition tasks are based on *events*. An event occurs at a particular time after a condition is met and may result in one or more actions. The events associated with analog output objects are:

- **Data-Output Event**

  A predefined amount of data has been output.
- **Timer Event**

  A predefined amount of time passed.
- **Start or Stop Event**

  A `start` or `stop` command was issued.
- **Trigger Event**

  A trigger occurred.
- **Runtime Error Event**

  A runtime error occurred. Runtime errors include timeouts and hardware errors incurred while converting data. These errors are distinguished from configuration errors.
- **Out-of-Data Event**

  Data was missed by the engine.

Recording and retrieving event information is described below.

Recording and Retrieving Event Information

For some of the events listed above, certain information is automatically stored in the `EventLog` property. `EventLog` is a structure that contains two fields: `Type` and `Data`. The `Type` field contains the event type. The event types that are logged for analog output objects are start, stop, trigger, runtime error, and out-of-data events. Data-output events and timer events are not stored in `EventLog`.

The `Data` field contains the event-specific subfields shown below.

Table 5-8: EventLog Subfields

Events	Data Subfield	Description
All events	TimeStamp	The absolute time the event occurred.
	SampleStamp	The number of samples acquired when the event occurred. The sample count begins when the start function is issued.
Trigger	Trigger	The trigger number
Out-of-data	Channel	The channels(s) that experienced an out-of-data condition

The example below illustrates how you can retrieve event information.

Example: Retrieving Event Information

Suppose you want to examine the events logged for the example on page 5-26. You can do this by accessing the EventLog property.

```
events = AO.EventLog
```

By examining the contents of the Type field, you can list the events that were recorded while the data was output.

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

Note: Unless a runtime error occurs, all data acquisition sessions must have a start event, a trigger event, and a stop event.

The Data field of the trigger event recorded the absolute time the trigger occurred, the number of samples output from when the start event occurred and the trigger occurred, and the trigger number.

```
trigdata = events(2).Data
trigdata =
      TimeStamp: 7.3001e+005
    SampleStamp: 0
        Trigger: 1
```

Action Properties and Functions

When an event occurs, you can execute a related function known as an *action function*. Action functions are M-files that can perform essentially any task during your data acquisition session. For example, you can use action functions to process data, display data, or display a message. All events have an associated action property.

You can specify the action function to be executed when an event occurs by including the name of an M-file as the value for the associated action property. You can also specify additional parameters to be passed as input arguments to the action function. The action function and additional parameters must be specified as elements of a cell array.

```
set(obj, 'Property, {'actionfcn', arg1, arg2 ,..., argn});
```

where:

- `Property` is the action property that is automatically called when the associated event occurs.
- `actionfcn` is the name of the M-file to be executed when the event occurs. `actionfcn` can be an M-file that you create or the `start`, `stop`, or `trigger` functions included with the toolbox.
- `arg1`, `arg2` ,..., `argn` are the optional input arguments that are passed to `actionfcn`. These arguments can be any data type since they are elements of a cell array.

Action functions require at least two arguments. The first argument must be a device object. The second argument is a variable that captures the event information contained by `EventLog`. This event information pertains only to the event that caused the action function to be executed. The action function prototype is shown below.

```
function actionfcn(obj, events)
```

If additional parameters are passed to the action function, then these arguments must be included after the two required arguments.

```
function actionfcn(obj, events, arg1, arg2 ,..., argn)
```

Note: If additional input parameters are passed to an action function, you are not required to use `varargin`. Instead, you can explicitly include the parameters in the function prototype as shown above.

Example action functions are given later in this section. The analog input properties related to action functions are shown below.

Table 5-9: Action-Related Properties

Property	Description
`OutOfDataAction`	Specifies the M-file to execute when data output has stopped
`RuntimeErrorAction`	Specifies the M-file to execute when a runtime error occurs
`SamplesOutputAction`	Specifies the M-file to execute when the number of samples specified by `SamplesOutputActionCount` is output
`SamplesOutputActionCount`	Sample-based frequency with which `SamplesOutputAction` is called
`StartAction`	Specifies the M-file to execute just prior to the device and engine start
`StopAction`	Specifies the M-file to execute just after the device and engine stop
`TimerAction`	Specifies the M-file to execute when the time specified by `TimerPeriod` occurs
`TimerPeriod`	Time-based frequency with which `TimerAction` is called
`TriggerAction`	Specifies the M-file to execute just after a trigger occurs

The relationship between events and action properties is described below for each event type.

Data-Output Event

A data-output event is generated immediately after a predefined number of samples are output.

This event executes the M-file specified for the SamplesOutputAction property every time the number of samples specified by SamplesOutputActionCount is output.

Timer Event

A time-based event is generated immediately after a predefined period of time has passed.

This event executes the M-file specified for the TimerAction property every time the number of seconds specified for the TimerPeriod property passes. Time is measured relative to when the hardware device starts running.

Start and Stop Events

Start and stop events are described together since these two events occur for all data acquisition sessions. A start event is generated immediately after the start command is issued, while the stop event is generated immediately after the stop command is issued.

The start event executes the M-file specified for the StartAction property prior to starting the hardware and engine. Once this M-file has completed execution, the Running property is set to On.

The stop event executes the M-file specified for the StopAction property. Under most circumstances, the M-file is not guaranteed to complete execution until sometime after the hardware and engine stop and the Running property is set to Off.

Trigger Event

A trigger event is generated immediately after a trigger is issued.

This event executes the M-file specified for the TriggerAction property. Under most circumstances, the M-file is not guaranteed to complete execution until sometime after the trigger occurs and Sending is set to On by the engine.

Runtime Error Event

A runtime error event is generated immediately after a runtime error occurs. Runtime errors include hardware errors and timeouts. Runtime errors do not include configuration errors.

This event executes the M-file specified for the RuntimeErrorAction property.

Out-of-Data Event

An out-of-data event is generated immediately after an out-of-data condition is detected for any channel group member. An out-of-data condition occurs when the data stream has been interrupted before being completely output to the D/A hardware.

This event executes the M-file specified for the `OutOfDataAction` property.

Examples: Using Action Properties and Functions

Examples showing how to create action functions and configure action-related properties are given below for a data-output action and a stop action.

Specifying an Action Function for an Event

This example illustrates how you can generate data-output events. As shown below, the M-file `daqdoc5_4disp` is executed every time the specified number of samples are output.

`daqdoc5_4.m`

Initialization: Create the analog output object `AO` for a sound card. The installed adaptors and hardware ID's are found with `daqhwinfo`.

```
AO = analogoutput('winsound');
%AO = analogoutput('nidaq', 1);
```

Configuration: Add two channels to `AO`, set the trigger to repeat four times, specify `daqug4_4disp` as the M-file to execute when 8000 samples are output, generate data to be queued, and queue the data with one call to `putdata`.

```
chans = addchannel(AO, 1:2);
%chans = addchannel(AO, 0:1);
set(AO, 'SampleRate', 8000);
ActualRate = get(AO, 'SampleRate');
set(AO, 'RepeatOutput', 4);
set(AO, 'SamplesOutputActionCount', 8000);
freq = get(AO, 'SamplesOutputActionCount');
set(AO, 'SamplesOutputAction', 'daqdoc5_4disp');
data = sin(linspace(0, 2*pi, 3*freq))';
time = (length(data)/AO.SampleRate)*(AO.RepeatOutput + 1);
putdata(AO, [data data]);
```

Execution: Display summary information, start AO, and pause to update command line display from daqdoc5_4disp.

```
AO
start(AO);
pause(0.1)
```

Termination: Delete AO.

daqdoc5_4.m

```
delete(AO)
```

The daqdoc5_4disp function is given below.

```
function daqdoc5_4disp(obj, event)
samp = obj.SamplesOutput;
disp([num2str(samp), ' samples have been output'])
```

Displaying Event Information with an Action Function

This example illustrates how the action function prototype allows you to easily display event information. The code below outputs five seconds of queued data to one sound card channel.

daqdoc5_5.m

Initialization: Create the analog output object AO for a sound card. The installed adaptors and hardware ID's are found with daqhwinfo.

```
AO = analogoutput('winsound');
%AO = analogoutput('nidaq', 1);
```

Configuration: Add one channel to AO, specify daqdoc5_5disp as the M-file 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);
chan = addchannel(AO, 1);
%chan = addchannel(AO, 0);
ActualRate = get(AO, 'SampleRate');
set(AO, 'StartAction', 'daqdoc5_5disp');
set(AO, 'TriggerAction', 'daqdoc5_5disp');
set(AO, 'StopAction', 'daqdoc5_5disp');
data = sin(linspace(0, 2*pi, ActualRate));
data = [data data data];
time = (length(data)/AO.SampleRate)
putdata(AO, data');
```

Execution: Display summary information, start AO, and pause to update command line display from daqdoc5_5disp.

```
AO
start(AO);
pause(time+0.5)
```

Termination: delete AO.

```
delete(AO)
```

daqdoc5_5.m

The daqdoc5_5disp function is shown below.

```
function daqdoc5_5disp(obj, event)
EventType = event.Type;
EventData = event.Data;
EventDataTime = EventData.TimeStamp;
EventDataSample = EventData.SampleStamp;
disp([EventType,' event occurred at ',datestr(EventDataTime,13)])
```

Engineering Unit Conversion

The Data Acquisition Toolbox provides you with a method for scaling output data such that it is not clipped when sent to the D/A converter. In most cases, data that represents specific engineering units (Pascals, Newtons, etc.) should be scaled. When data is output to a D/A converter, you must be aware of these two issues:

- The range of the data you are outputting to the D/A converter.
- The valid range(s) that your D/A hardware can be set to. For many devices, the D/A hardware range is specified by the gain and polarity.

You can set engineering unit (EU) conversions for each channel of data. Therefore, the properties related to EU conversions can be set on a per-channel basis. These properties are given below.

Table 5-10: Engineering Units Properties

Property	Description
`OutputRange`	Range of D/A converter
`Units`	Engineering units name
`UnitsRange`	Scaling factor for data in MATLAB workspace

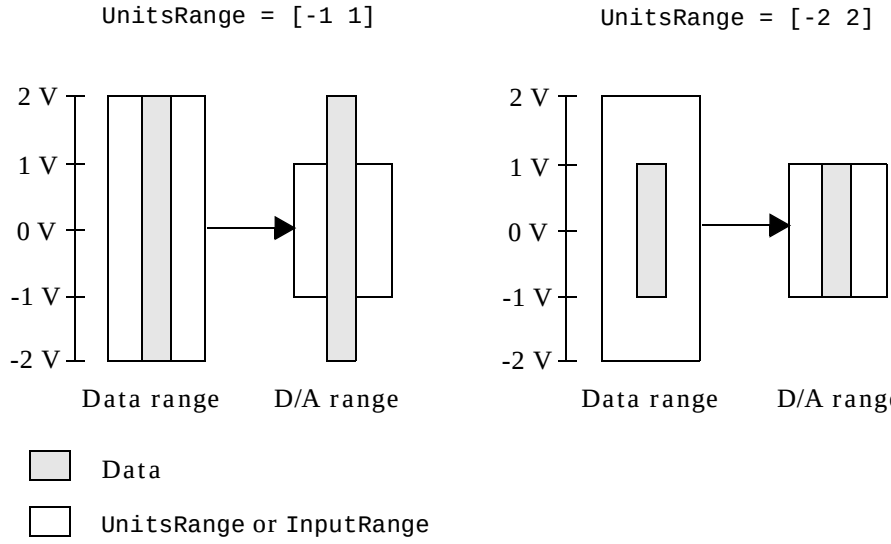
Note: Only linear engineering unit conversions are supported. You can perform nonlinear conversions by creating the appropriate M-file function.

For many devices, the `OutputRange` property is expressed in terms of gain and polarity. If data is outside the `OutputRange` values, it is clipped to the values set for `UnitsRange`. An example using `UnitsRange` is given in the following example.

Example: Performing a Linear Conversion

Suppose you are outputting data to one analog output channel. The output data ranges between -2 and 2 volts and the D/A converter has a maximum range of -1 volt to 1 volt. If the data is not scaled, it will be clipped since its range exceeds

that of the D/A hardware. To solve this problem, you must use the `UnitsRange` property. As shown in the figure below, setting `UnitsRange` to `[-2 2]` scales the output data such that a value of -2 maps to -1 at the hardware level and a value of 2 maps to 1 at the hardware level.



The code below illustrates how to configure the appropriate properties.

daqdoc5_6.m

Initialization: create the analog output object `AO` for a sound card. The available adaptors are found with `daqhwinfo`.

```
AO = analogoutput('winsound');
%AO = analogoutput('nidaq', 1);
```

Configuration: Add one channel to AO, set the trigger to repeat two times, generate data to be queued, set the UnitsRange property, and queue the data with one call to putdata.

```
chan = addchannel(AO, 1);
%chan = addchannel(AO, 0);
set(AO, 'SampleRate', 8000);
ActualRate = get(AO, 'SampleRate');
set(AO, 'RepeatOutput', 2);
data = 2*sin(0:(1/ActualRate):2*pi)';
set(chan, 'UnitsRange', [-2 2]);
putdata(AO, data)
```

Execution: Start AO and wait for the device object to stop running.

```
start(AO);
while strcmp(AO.Running, 'On')
end
```

Termination: delete AO.

daqdoc5_6.m

```
delete(AO)
```

6

Function Reference

The Function Reference Page Format

Each Data Acquisition Toolbox function reference page contains some or all of this information:

- The function name
- The purpose of the function
- The function syntax

  All valid input argument and output argument combinations are shown. In some cases, an ellipsis (. . .) is used for the input arguments. This means that all preceding input argument combinations are valid for the specified output argument(s).
- A description of each argument
- A description of the function
- An example of usage

  Some function examples are constructed explicitly for sound cards or National Instruments boards. However, for most of these device-specific examples, the functionality illustrated applies to all supported devices. If device-specific properties or values are used, or if the illustrated behavior depends on the device, then this is explained in the example.
- Related functions or properties

Data Acquisition Toolbox Functions

This section describes the Data Acquisition Toolbox M-file functions that you can use directly. These functions and the device objects they are associated with are organized into the six groups shown below.

Creating Device Objects		AI	AO	DIO
`analoginput`	Create an analog input object.	✓		
`analogoutput`	Create an analog output object.		✓	
`digitalio`	Create a digital I/O object.			✓

Configuring Device Objects		AI	AO	DIO
`addchannel`	Add hardware channels to an analog input or analog output object.	✓	✓	
`addline`	Add hardware lines to a digital I/O object.			✓
`get`	Return property values.	✓	✓	✓
`set`	Configure or display property values.	✓	✓	✓
`setverify`	Set and return the specified property value.	✓	✓	✓

Executing the Object		AI	AO	DIO
`start`	Start the execution of a device object.	✓	✓	✓
`stop`	Stop the execution of a device object.	✓	✓	✓
`trigger`	Manually execute a trigger.	✓	✓	✓

Working with Data		AI	AO	DIO
flushdata	Remove data from the data acquisition engine.	✓	✓	
getdata	Return data, time, and event information to the MATLAB workspace for an analog input.	✓		✓
getsample	Return one sample for each analog input channel group member or digital I/O line group member.	✓		✓
peekdata	Immediately return data to the MATLAB workspace for an analog input object.	✓		
putdata	Queues data in the engine for eventual output to the D/A hardware.		✓	✓
putsample	Send one sample for each analog output channel group member or digital I/O line group member.		✓	✓

Getting Information and Help		AI	AO	DIO
daqhelp	Display help for device objects, constructors, functions, and properties.	✓	✓	✓
daqhwinfo	Display data acquisition hardware information.	✓	✓	✓
nidaq	Display information about National Instruments adaptor	✓	✓	✓
propinfo	Return property information for device objects, channels, or lines.	✓	✓	✓
winsound	Display information for sound card adaptor	✓	✓	✓

General Purpose		AI	AO	DIO
clear	Clear device objects from the MATLAB workspace.	✓	✓	✓
daqaction	Display event information for specified event	✓	✓	
daqfind	Return device objects, channels, or lines from the data acquisition engine to the workspace.	✓	✓	✓
daqread	Read a Data Acquisition Toolbox (.daq) file.	✓		
daqmem	Allocate or display memory for one or more device objects.	✓	✓	✓
daqregister	Register or unregister a Data Acquisition Toolbox adaptor.	✓	✓	✓
daqreset	Remove data acquisition objects and .dll files from memory.	✓	✓	✓
delete	Delete device objects, channels, or lines.	✓	✓	✓
disp	Display device object, channel, and line information.	✓	✓	✓
ischannel	Check for channels.	✓	✓	
isline	Check for lines			✓
isvalid	Check for valid device objects, channels, or lines.	✓	✓	✓
length	Determine length of data acquisition objects.	✓	✓	✓
load	Load device objects into the MATLAB workspace	✓	✓	✓
makenames	Generate a list of channel or line names.	✓	✓	✓
obj2code	Convert device objects, channels, or lines to MATLAB code and save to disk.	✓	✓	✓
save	Save device objects to a MAT-file.	✓	✓	✓
size	Determine size of data acquisition objects.	✓	✓	✓

A number of other MATLAB M-file helper functions are provided with this toolbox to support the functions listed above. These helper functions are not documented since they are not intended for direct use.

Purpose

Add hardware channels to an analog input or analog output object.

Syntax

```
chans = addchannel(obj, hwch);
chans = addchannel(obj, hwch, index);
chans = addchannel(obj, hwch, names);
chans = addchannel(obj, hwch, index, names);
```

Arguments

`obj`	An existing analog input or analog output object.
`hwch`	The hardware channels added to the object. Any MATLAB vector syntax can be used.
`index` (optional)	The MATLAB indices to associate with the hardware channels. Any MATLAB vector syntax can be used provided the vector elements are monotonically increasing.
`names` (optional)	A cell array containing the names of the channels.
`chans`	A vector of channels with the same length as `hwch`.

Description

`addchannel` adds hardware channels to an existing analog input or analog output object. The added channels constitute a channel group. The values you supply for `hwch` depend on the hardware accessed. For National Instruments hardware, channels are "zero-based" (begin at zero). For Hewlett-Packard hardware and sound cards, channels are "one-based" (begin at one). If in doubt, consult your hardware manual.

A hardware channel can be assigned to multiple device objects, or multiple times to the same device object. For sound cards however, hardware channels cannot be assigned multiple times to the same device object. Furthermore, sound cards can be configured in one of only two ways: mono mode or stereo mode. For mono mode, `hwch` must be `1`, while for stereo mode, `hwch` must be specified as `[1 2]`.

For every hardware channel contained by a device object, there is an associated MATLAB index that is used to reference the channel. The index assignments follow these rules:

- If index is not specified, and no hardware channels are assigned to the object, then the assigned indices start at one and increase monotonically. If hardware channels have already been assigned to the object, then the assigned indices 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 hwch and index are not equal, then an error is returned and no channels are added to the device object.

In either case, the resulting indices will begin at one and increase monotonically up to the total size of the channel group.

Hardware channels can be assigned descriptive names. Choosing a unique descriptive name may be a useful way to identify and reference channels. For a single call to addchannel, you can:

- Specify one channel name which will apply 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 less than 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 begin with a character and contain only characters, numbers, and underscores. If you are naming a large number of channels, then the makenames function may be useful. If a channel is not assigned a descriptive name, then it must be referenced by index.

Examples

Sound Card

Suppose you create the analog input object AI1 for a sound card. Sound cards have 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);
```

and the ChannelName property is assigned the name Mono. You can now configure the sound card to operate in stereo mode by adding the second channel

```
addchannel(AI1, 2);
```

and the ChannelName property is assigned the names 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.

```
AI1 = analoginput('winsound');
addchannel(AI1, 1:2);
```

National Instruments

Suppose you created the analog input object AI1 for a National Instruments board, and add the first four physical hardware channels (channels 0-3).

```
AI1 = analoginput('nidaq', 1)
addchannel(AI1, 0:3);
```

These channels are automatically assigned the indices 1-4. You can associate descriptive names with channels by passing a cell array of names.

```
addchannel(AI1, 0:3, {'chan1', 'chan2', 'chan3', 'chan4'});
```

The cell array of channel names can be created manually as shown above or with the makenames function.

You can 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 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 since there is a gap in the indices (index 5 has no associated hardware channel).

See Also

Functions

delete, makenames

Purpose

Add hardware lines to a digital I/O object.

Syntax

```
lines = addline(obj, hwl);
lines = addline(obj, hwl, index);
lines = addline(obj, hwl, index, port);
lines = addline(obj, hwl, port);
lines = addline(obj, hwl, index, port, names);
lines = addline(obj, hwl, index, names);
lines = addline(obj, hwl, port, names);
```

Arguments

`obj`	An existing digital I/O object.
`hwl`	The hardware lines added to the object. Any MATLAB vector syntax can be used.
`index` (optional)	The MATLAB indices to associate with the hardware channels. Any MATLAB vector syntax can be used provided the vector elements are monotonically increasing.
`port` (optional)	The port number. If not specified, then lines are added first from port 1, then from port 2, and so on.
`names` (optional)	A cell array containing the names of the lines.
`lines`	A vector of lines with the same length as `hwch`.

Description

`addline` adds hardware lines to an existing digital I/O device. The added lines constitute a line group. The values you supply for `hwl` depend on the hardware accessed. For National Instruments hardware, lines are “zero-based” (begin at zero). For Hewlett-Packard hardware, lines are “one-based” (begin at one). If in doubt, consult your hardware manual.

With every hardware line there is an associated MATLAB index that is used to reference the line. The index assignments follow these rules:

- If `index` is not specified, and no hardware lines are assigned to the object, then the assigned indices start at one and increase monotonically. If

hardware lines have already been assigned to the object, then the assigned indices start at the next highest 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 `hwch` and `index` are not equal, then an error is returned and no lines are added to the device object.

In either case, the resulting indices must begin at one and increase monotonically up to the total size of the line group. If the line group contains gaps or indices without lines assigned, then an error is returned.

Hardware lines can be assigned descriptive names. Choosing a unique descriptive name may be a useful way to identify and reference lines. If the lengths of `names` and `hwl` are not the same, then only those lines having a corresponding `names` entry are named. Line names must begin with a character and contain only characters, numbers, and underscores. If you are naming a large number of lines, then the `makenames` function may be useful. If a line is not assigned a descriptive name, then it must be referenced by index.

Examples

National Instruments

Suppose you created a digital I/O object called `DIO1` for a National Instruments board, and add the first four physical hardware lines (lines 0-3) from port 1.

```
DIO1 = digitalio('nidaq', 1)
addline(DIO1, 0:3);
```

These lines are automatically assigned the indices 1-4. You can associate descriptive names with these lines by passing a cell array of names.

```
addline(DIO1, 0:3, {'line1', 'line2', 'line3', 'line4'});
```

The cell array of line names can be created manually or with the `makenames` function.

You can add lines 0-3 from port 2 to the existing line group.

```
addline(DIO1, 0:3, 2);
```

The new lines are automatically assigned the indices 5-8. Suppose instead you add lines 0-3 from port 2 to the line group and assign them the indices 1-4.

```
addline(DIO1, 0:3, 1:4, 2);
```

The new lines are assigned the indices 1-4, and the previously defined lines are reindexed as indices 5-8. However, if you assigned lines 0-3 from port 2 to indices 6-9, then when the device object was started, an error would be returned since there is a gap in the indices (index 5 has no associated hardware line).

See Also

Functions

`delete`, `makenames`

analoginput

Purpose

Create an analog input object.

Syntax

```
AI = analoginput('adaptor')
AI = analoginput('adaptor', ID)
```

Arguments

`adaptor`	Name of the hardware driver adaptor
`ID`	Device identifier
`AI`	The analog input object

Description

`analoginput` creates an analog input object. This device object exists both in the data acquisition engine and the MATLAB workspace. When the object is created:

- If referenced for the first time, the hardware device driver associated with `adaptor` is loaded and initialized.
- There are no hardware channels associated with it.

`analoginput` accepts one or two input arguments depending on the hardware you are accessing. The input arguments are given below.

Description	Supported Vendors and Devices		
	Sound Card	**National Instruments**	**Hewlett-Packard**
`adaptor`	`winsound`	`nidaq`	`hpvxi`
`ID`	N/A	Device number	Device ID

For National Instruments hardware, `ID` must be specified as a double. For Hewlett-Packard hardware, `ID` can be specified as a double or a string.

If the values supplied to `analoginput` are incomplete or incorrect, or the referenced driver is not found, then an error is returned. To use the analog input object to perform data acquisition tasks, hardware channels must be added with the `addchannel` function.

Example

Sound Card

To create an analog input object for a sound card

```
AI = analoginput('winsound')
```

National Instruments

To create an analog input object for a National Instruments device defined as device number 1

```
AI = analoginput('nidaq', 1)
```

Hewlett-Packard

To create an analog input object for a Hewlett-Packard (HP) VXI device with device ID 8 (slot 8)

```
AI = analoginput('hpvxi', 8)
```

The HP driver allows you to span multiple hardware. To span two HP VXI devices with device ID's 8 and 9

```
AI = analoginput('hpvxi', [8 9])
```

See Also

Functions

`addchannel`, `analogoutput`, `digitalio`

analogoutput

Purpose

Create an analog output object.

Syntax

```
AO = analogoutput('adaptor')
AO = analogoutput('adaptor', ID)
```

Arguments

`adaptor`	Name of the hardware driver adaptor
`ID`	Device identifier
`AO`	The analog output object

Description

`analogoutput` creates an analog output object. This device object exists both in the data acquisition engine and the MATLAB workspace. When the object is created:

- If referenced for the first time, the hardware device driver associated with `adaptor` is loaded and initialized.
- There are no hardware channels associated with it.

`analogoutput` accepts one or two input arguments depending on the hardware you are accessing. The input arguments are given below.

Description	Supported Vendors and Devices		
	sound card	National Instruments	Hewlett-Packard
`adaptor`	`winsound`	`nidaq`	`hpvxi`
`ID`	N/A	Device number	Device ID

For National Instruments hardware, `ID` must be specified as a double. For Hewlett-Packard hardware, `ID` can be specified as a double or a string.

If the values supplied to `analogoutput` are incomplete or incorrect, or the referenced driver is not found, then an error is returned. To use the analog output object to perform data acquisition tasks, channels must be added with the `addchannel` function.

Example

Sound Card

To create an analog output object for a sound card

```
AO = analogoutput('winsound')
```

National Instruments

To create an analog output object for a National Instruments device defined as device number 1

```
AO = analogoutput('nidaq', 1)
```

Hewlett-Packard

To create an analog output object for a Hewlett-Packard (HP) VXI device with device ID 8 (slot 8)

```
AO = analogoutput('hpvxi', 8)
```

The HP driver allows you to span multiple hardware. To span two HP VXI devices with device ID's 8 and 9

```
AO = analogoutput('hpvxi', [8 9])
```

See Also

Functions

`addchannel`, `analoginput`, `digitalio`

clear

Purpose

Clear device objects from the MATLAB workspace.

Syntax

```
clear obj
```

Description

`clear obj` removes the device object including all associated channels or lines from the MATLAB workspace but not from the data acquisition engine. Therefore, if multiple references to an object exist in the workspace, removing one reference will not invalidate the remaining references. Cleared objects can be restored to the MATLAB workspace with `daqfind`.

Note: To remove device objects, channels, or lines from the workspace and the data acquisition engine, `delete` should be used.

See Also

Functions

`daqfind`, `delete`

Purpose

Display event information for the specified event.

Syntax

```
daqaction(obj, event)
daqaction(obj, event, arg1, arg2, ..., argn)
```

Description

`daqaction(obj, event)` is an example action function that displays the type of the event, the time of the event and the number of samples acquired at the time the event takes place. Action functions must be passed two arguments. `obj` is a device object and `event` captures the event information contained by the `EventLog` property.

When any of the action properties are set to `daqaction`, the information described above is displayed to the MATLAB command window.

Example

This example calls `daqaction` when a trigger event, runtime error event, or stop event occurs.

```
ai = analoginput('winsound');
addchannel(ai, 1);
set(ai, 'TriggerRepeat', 3);
set(ai, 'TriggerAction', 'daqaction');
set(ai, 'RuntimeErrorAction', 'daqaction');
set(ai, 'StopAction', 'daqaction');
start(ai);
delete(ai);
```

See Also

Properties

`EventLog`

daqfind

Purpose

Return device objects, channels, or lines from the data acquisition engine to the workspace.

Syntax

```
daqobjcell = daqfind
daqobjcell = daqfind('P1', V1, 'P2', V2,...)
daqobjcell = daqfind(obj, 'P1', V1, 'P2', V2,...)
```

Description

`daqfind` locates and returns Data Acquisition Toolbox device objects, channels, or lines. The items returned can be selected based on property values. This function is particularly useful if:

- A device object is cleared from the MATLAB workspace, and needs to be retrieved from the data acquisition engine.
- You need to locate a device object, channel, or line having a particular property value.

`daqobjcell = daqfind` returns any device objects that exist in the data acquisition engine. The objects are returned as a cell array to `daqobjcell`.

`daqobjcell = daqfind('P1', V1, 'P2', V2,...)` returns any device objects, channels, or lines that exist in the data acquisition engine having the specified property values.

`daqobjcell = daqfind(obj, 'P1', V1, 'P2', V2,...)` returns the device object, or any channels or lines contained by `obj` that have the specified property values.

Example

Sound Card

`daqfind` allows you to assign a MATLAB variable to a cleared device object, channel, or line.

```
ai = analoginput('winsound')
ch = addchannel(ai, 1:2)
set(ch, {'ChannelName'}, {'Joe'; 'Jack'})
clear ai
aicell = daqfind
ai_new = aicell{1}
```

You can find the channel that is assigned the descriptive name `Jack`.

```
out = daqfind(ai_new, 'ChannelName', 'Jack')
chan = out{1}
```

`daqfind` searches `ai`'s channels for the specified property value and returns the result to `out`.

See Also

Functions

`clear`

daqhelp

Purpose

Display help for device objects, constructors, functions, and properties.

Syntax

```
daqhelp
daqhelp('name')
daqhelp(obj)
daqhelp(obj, 'name')
```

Description

`daqhelp` displays a complete listing of Data Acquisition Toolbox functions with a brief description of each function.

`daqhelp('name')` displays device object, constructor, adaptor, function, or property help.

If `name` is the name of a device object 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 command-line help for the device object's constructor is also displayed. In this case, `name` is not specified as a string. If `name` is the name of a device object constructor, and the ".m" extension is included, the command-line constructor help is displayed. If `name` is the name of an adaptor, the command-line adaptor help is displayed. If `name` is the name of a function or property, the command-line function or property help is displayed.

Object-specific function information can be displayed by specifying `name` as object/function. For example to display the command-line help for an analog input object's `getdata` function, `name` would be `analoginput/getdata`.

Object-specific property information can be displayed by specifying `name` as `obj.property`. For example to display the command-line help for an analog input object's `SampleRate` property, `name` would be `analoginput.SampleRate`.

`daqhelp(obj)` displays a complete listing of functions and properties for `obj`, along with the command-line help for the `obj`'s constructor.

`daqhelp(obj, 'name')` displays the help for `name` for the specified device object. `name` can be a property or function name.

Examples

The commands listed below are some of the ways `daqhelp` can be used to obtain information about device objects, constructors, adaptors, functions, and properties.

```
daqhelp('analogoutput')
daqhelp('analogoutput.m')
daqhelp('winsound')
daqhelp set
daqhelp analoginput/peekdata
daqhelp analoginput.TriggerDelayUnits
```

The commands listed below are some of the ways `daqhelp` can be used to obtain information about functions and properties for a given device object.

```
ai = analoginput('winsound');
daqhelp(ai, 'TriggerTime');
daqhelp(ai, 'getsample');
daqhelp(ai, 'set');
```

See Also

Functions

`propinfo`

daqhwinfo

Purpose

Display data acquisition hardware information.

Syntax

```
hwinfo = daqhwinfo
hwinfo = daqhwinfo('adaptor')
hwinfo = daqhwinfo(obj)
hwinfo = daqhwinfo(obj,'Field')
```

Description

`hwinfo = daqhwinfo` returns general hardware-related information as a structure to `hwinfo`. The information is contained in the fields shown below.

Field	Description
`ToolboxName`	Toolbox name
`ToolboxVersion`	Toolbox version
`MatlabVersion`	MATLAB version
`InstalledAdaptors`	Installed adaptors. For example, `winsound`, `nidaq`

`hwinfo = daqhwinfo('adaptor')` returns hardware-related information for `adaptor` as a structure to `hwinfo`. The information is contained in the fields shown below.

Field	Description
`AdaptorDllName`	Adaptor DLL name. For example, `mwwinsound.dll`
`AdaptorDllVersion`	Adaptor DLL version
`AdaptorName`	Adaptor name. For example, `winsound`
`BoardNames`	Board names. For example, `Sound Blaster Record`

Field	Description
InstalledBoardIds	Installed board hardware identifier.
ObjectConstructorName	Object constructor name. For example subsystem('winsound')

hwinfo = daqhwinfo(obj) returns hardware-related information for the device object obj as a structure to hwinfo. The information is contained in the fields shown below

hwinfo = daqhwinfo(obj, 'Field') returns the specified hardware-related information for the device object obj as a structure to hwinfo. Field can be a single field or a cell array of fields. The valid fields are shown below.

Field	Description
AutoCalibrate	Possible values are On or Off
Bits	Number of bits of resolution
ConversionOffset	Offset used by the engine when converting from binary values to volts (usually 0 except for 8-bit sound cards or 16-bit sound cards configured for 8 bit conversions)
Coupling	Possible values are DC or AC Coupled
DeviceName	Device name
DifferentialChannels	Number of differential channels (AI-only)
DriverName	Driver name
Gains	Gains (AI-only)
ID	Hardware identifier
InputRanges	Input ranges (AI-only)
MaxSampleRate	Maximum sample rate
MinSampleRate	Minimum sample rate

Field	Description
NativeDataType	Native data type
OutputRanges	Output ranges in volts (AO-only)
Polarity	Can be Unipolar or Bipolar
SampleType	Can be SimultaneousSample or Scanning
SingleEndedChannels	Number of single-ended channels (AI-only)
SubsystemType	Possible values are AnalogInput, AnalogOutput, or DigitalIO
TotalChannels	Total number of channels
VendorDriverDescription	Vendor driver description
VendorDriverVersion	Vendor driver version

Example

To display all the installed adaptors

```
hwinfo = daqhwinfo;
hwinfo.InstalledAdaptors
ans =
    'winsound'
    'nidaq'
```

To display the subsystems for all installed winsound devices

```
hwinfo = daqhwinfo('winsound');
hwinfo.Subsystems
ans =
    'AnalogInput AnalogOutput'
```

To display the input ranges for an analog input object created for a sound card

```
ai = analoginput('winsound');
hwinfo = daqhwinfo(ai);
hwinfo.InputRanges
ans =
    '[-1 1]'
```

Purpose

Allocate or display memory for one or more device objects.

Syntax

```
mem = daqmem
mem = daqmem(obj)
mem = daqmem(obj, value)
mem = daqmem(value)
```

Description

`daqmem` returns a structure array containing four fields: `System`, `Object`, `BytesUsed`, and `BytesAvailable`. `System` returns the memory available on your system and the memory used by your system, `Object` lists all existing device objects, `BytesUsed` lists the memory used by each device object, and `BytesAvailable` lists the maximum memory allocated for each device object.

`daqmem(obj)` returns the memory used by the specified device object.

`daqmem(obj, value)` sets the maximum memory that can be allocated for the specified device object.

`daqmem(value)` sets the maximum memory that can be allocated for all device objects. If multiple device objects are being used at the same time, the engine automatically assigns the appropriate memory to each device object.

Example

Sound Card

See Also

Properties

`BufferingConfig`, `BufferingMode`

daqread

Purpose

Read a Data Acquisition Toolbox (.daq) file.

Syntax

```
data = daqread('file');
data = daqread('file','P1, V1, 'P2', V2,...);
[data, time] = daqread(...);
[data, time, abstime] = daqread(...);
[data, time, abstime, events] = daqread(...);
[data, time, abstime, events, daqinfo] = daqread(...);
daqinfo = daqread('file','info');
```

Description

`data = daqread('file')` reads all the data from `file` and returns an *m*-by-*n* data matrix, `data`, where *m* is the number of samples and *n* is the number of channels. If data from multiple triggers is read, each trigger is separated by a NaN and *m* is increased by the number of triggers.

`data = daqread('file','P1', V1, 'P2', V2,...)` reads the specified data from `file` and returns an *m*-by-*n* matrix to `data`, where *m* is the number of samples and *n* is the number of channels. If data from multiple triggers is read, each trigger is separated by a NaN and *m* is increased by the number of triggers. The amount of data returned and the format of that data is specified with property name/property value pairs. The available properties and values are given below.

Property	Data Type	Description
`Samples`	MATLAB vector	Specifies the sample range.
`Time`	MATLAB vector	Specifies the time range.
`Triggers`	MATLAB vector	Specifies the trigger range.
`Channels`	MATLAB vector or cell array	Specifies the channel range. Channel names can be specified as a cell array.

Property	Data Type	Description
DataFormat	{double} | native	Specifies the data format as doubles or native.
TimeFormat	{vector} | matrix	Specifies the time format as a vector or a matrix.

The Samples, Time, and Triggers properties are mutually exclusive. The Time property must be specified as a datenum value. This allows event times to be used as the range.

[data, time] = daqread(...) returns sample-time pairs.

[data, time, abstime] = daqread(...) returns sample-time pairs and the absolute time of the first trigger. abstime is returned as a datenum value and can be converted to a string using datestr.

[data, time, abstime, events] = daqread(...) returns sample-time pairs, the absolute time of the first trigger, and a log of events. abstime is returned as a datenum value and can be converted to a string using datestr. events contains the appropriate event structure based on the samples, triggers or time specified. The entire event log is returned only if samples, time, and triggers are not specified.

[data, time, abstime, events, daqinfo] = daqread(...) returns sample-time pairs, the absolute time, the event log, and the structure daqinfo, which contains two fields: ObjInfo and HwInfo. ObjInfo is a structure containing property name/property value pairs and HwInfo is a structure containing hardware information. The events structure is identical to the information returned by daqinfo.ObjInfo.EventLog. The entire event log is returned only if samples, time, and triggers are not specified.

daqinfo = daqread('file','info') returns the structure daqinfo, which contains two fields: ObjInfo and HwInfo. ObjInfo is a structure containing property/value pairs and HwInfo is a structure containing hardware information. The entire event log is returned to daqinfo.ObjInfo.EventLog.

Note: Data Acquisition Toolbox (.daq) files are created by specifying a value for the `LogFileName` property and setting `LoggingMode` to `Disk` or `Disk&Memory`.

Example

National Instruments

Suppose you configure the analog input object `ai` for a National Instruments board as shown below. The object acquires one second of data for four channels, and saves the data to the output file `data.daq`.

```
ai = analoginput('nidaq','1');
chans = addchannel(ai, 0:3);
set(ai,'SampleRate', 1000)
ActualRate = get(ai,'SampleRate')
set(ai,'SamplesPerTrigger', ActualRate)
set(ai,'LoggingMode','Disk&Memory')
set(ai,'LogFileName','data.daq')
start(ai)
```

After the data has been collected and saved to a disk file, you can retrieve the data and other acquisition-related information using `daqread`. To read all the sample-time pairs from the file `data.daq`

```
[data, time] = daqread('data.daq');
```

To read samples 500 to 1000 for all channels from `data.daq`

```
data = daqread('data.daq', 'Samples', [500 1000]);
```

To obtain the channel property information from `data.daq`

```
daqinfo = daqread('data.daq', 'info');
chaninfo = daqinfo.ObjInfo.Channel;
```

To obtain a list of event types and event data contained by `data.daq`

```
daqinfo = daqread('data.daq', 'info');
events = daqinfo.ObjInfo.EventLog;
event_type = {events.Type};
event_data = {events.Data};
```

To return the absolute time of the first logged event (the start event) and convert the time from a `datenum` value to a string

```
event1_time = event_data{1}.Time;
event1_time_str = datestr(event1_time);
```

See Also

Properties

`LogFileName`, `LoggingMode`, `LogToDiskMode`

daqregister

Purpose

Register or unregister a Data Acquisition Toolbox adaptor.

Syntax

```
daqregister('adaptor')
daqregister('adaptor','unload')
```

Description

`daqregister` registers or unregisters toolbox adaptors. An 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. Normally, registration occurs automatically when the toolbox is installed.

`daqregister('adaptor')` registers `adaptor`.

`daqregister('adaptor','unload')` unregisters `adaptor`.

Examples

Sound Card

```
daqregister('winsound')
```

National Instruments

```
daqregister('nidaq')
```

Hewlett-Packard

```
daqregister('hpvxi')
```

Purpose

Remove device objects and .dll files from memory.

Syntax

```
daqreset
```

Description

daqreset removes all device objects existing in the engine, and unloads all .dll files loaded by the engine.

daqreset should be used if you want to return MATLAB to the known initial state of having no device objects and .dll's loaded in memory. When MATLAB is returned to this state, the data acquisition hardware is reset.

See Also

Functions

clear, delete

delete

Purpose

Delete device objects, channels, or lines.

Syntax

```
delete(obj)
delete(obj.Channel(index))
delete(obj.Line(index))
```

Description

`delete` removes device objects, channels, or lines from the data acquisition engine and the MATLAB workspace. If multiple references to a device object exist in the workspace, removing one reference invalidates the remaining references. These remaining references should be cleared from the workspace. `delete` should be used at the end of a data acquisition session. If `delete` is issued and the `Running` property is set to `On`, the device object, channels, or lines will not be deleted and an error will be returned.

`delete(obj)` removes the device object including all associated channels or lines. If `obj` is the last object accessing the driver, then the driver and associated adaptor are closed and unloaded.

`delete(obj.Channel(index))` removes the channels specified by `index` from `obj`. As a result, the remaining channels may be reordered.

`delete(obj.Line(index))` removes the lines specified by `index` and from `obj`. As a result, the remaining lines may be reordered.

Examples

Sound Cards

Suppose you create the analog input object `AI1` for a sound card, and configure it to operate in stereo mode.

```
AI1 = analoginput('winsound')
addchannel(AI1, 1:2)
```

You can now configure the sound card for mono mode by deleting hardware channel 2.

```
delete(AI1.Channel(2))
```

If hardware channel 1 is deleted instead, then an error is returned.

National Instruments

Suppose you create the analog input object `ai` for a National Instruments board, add hardware channels 0-7 to it, and make a copy of channels 0 and 1.

```
ai = analoginput('nidaq', 1)
addchannel(ai, 0:7)
ch = ai.Channel(1:2)
```

To delete channels 0 and 1

```
delete(ch)
```

These channels are deleted from the data acquisition engine and the workspace, and are no longer accessible associated with `ai`. The remaining channels are reindexed such that the indices begin at 1 and increase monotonically to 14.

To delete the analog input object

```
delete(ai)
```

See Also

Functions

`clear`, `daqreset`

digitalio

Purpose

Create a digital I/O object.

Syntax

```
DIO = digitalio('adaptor', ID);
```

Arguments

`adaptor`	Name of the hardware driver adaptor
`ID`	Device identifier
`DIO`	The digital I/O object

Description

`digitalio` creates a digital I/O object. This device object exists both in the data acquisition engine and the MATLAB workspace. When the object is created:

- If referenced for the first time, the hardware device driver associated with `adaptor` is loaded and initialized.
- There are no hardware lines associated with it.

`digitalio` accepts two input arguments. The input arguments are given below.

Description	Supported Vendors and Devices	
	National Instruments	Hewlett-Packard
`adaptor`	`nidaq`	`hpvxi`
`ID`	Device number	Device ID

For National Instruments hardware, `ID` must be specified as a double. For Hewlett-Packard hardware, `ID` can be specified as a double or a string.

If the values supplied to `digitalio` are incomplete or incorrect, or the referenced driver is not found, then an error is returned. To use the digital I/O object to perform data acquisition tasks, lines must be added with the `addline` function.

Example

National Instruments

To create a digital I/O object for a National Instruments device defined as device number 1

```
DIO = digitalio('nidaq', 1);
```

Hewlett-Packard

To create a digital I/O object for a Hewlett-Packard (HP) VXI device with device ID 8 (slot 8)

```
DIO = digitalio('hpvxi', 8);
```

The HP driver allows you to span multiple hardware. To span two HP VXI devices with device ID's 8 and 9

```
AI = digitalio('hpvxi', [8 9]);
```

See Also

Functions

`addline`, `analoginput`, `analogoutput`

disp

Purpose

Display device object, channel, and line information.

Syntax

```
obj
disp(obj)
obj.Channel(index)
disp(obj.Channel(index))
obj.Line(index)
disp(obj.Line(index))
```

Description

`disp(obj)` displays base information about the specified device object, and any channels or lines contained by the device object. Typing `obj` at the command line produces the same base information.

`disp(obj.Channel(index))` displays base information about the specified channels contained by `obj`. Typing `obj.Channel(index)` at the command line produces the same base information.

`disp(obj.Line(index))` displays base information about the specified lines contained by `obj`. Typing `obj.Line(index)` at the command line produces the same base information.

Note: You can invoke `disp` by typing the device object name at the MATLAB command line, or by excluding the semicolon when creating an object, adding channel or lines, or setting property values.

Example

Sound Card

Suppose the analog input object `AI` is created for a sound card and configured as shown below:

```
AI = analoginput('winsound');
chans = addchannel(AI, 1:2);
set(AI,'SampleRate', 8000)
set(AI,'SamplesPerTrigger', 8000)
```

Typing the object name (AI) at the command line displays the device object and channel information shown below.

```
Display Summary of Analog Input (AI) Object Using 'Sound Blaster Record'.
  Acquisition Parameters:  8000 samples per second on each channel.
                           8000 samples per trigger on each channel.
                           1 sec. of data to be logged upon START.
                           Log data to 'Memory' on trigger.
      Trigger Parameters:  1 'Immediate' trigger(s) on START.
           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       'Left'        1           [-1 1]       [-1 1]        [-1 1]       'Volts'
   2       'Right'       2           [-1 1]       [-1 1]        [-1 1]       'Volts'
```

flushdata

Purpose

Remove data from the data acquisition engine.

Syntax

```
flushdata(obj)
```

Description

`flushdata(obj)` removes data from the data acquisition engine and resets the `SamplesAvailable` property to zero.

Example

National Instruments

Suppose you create the analog input object `ai` for a National Instruments board and add hardware channels 0-7 to it.

```
ai = analoginput('nidaq', 1)
addchannel(ai, 0:7)
```

A one second acquisition is configured where `SampleRate` is set to 2000 Hz and `SamplesPerTrigger` is set to 2000 samples. The device object is then started.

```
set(ai,'SampleRate', 2000)
ActualRate = get(ai, 'SampleRate');
set(ai,'SampleRate', ActualRate)
set(ai, 'SamplesPerTrigger', 2000);
start(ai)
```

2000 samples will be acquired for each channel group member. To extract 100 samples from the data acquisition engine for each channel

```
data = getdata(ai, 100);
```

To remove the remaining 1900 samples from the data acquisition engine, the `flushdata` command can be used.

```
flushdata(ai)
```

See Also

Functions

`getdata`

Properties

`SamplesAvailable`

Purpose

Return property values.

Syntax

```
out = get(obj)
out = get(obj.Channel(index))
out = get(obj,'Property')
out = get(obj.Channel(index),'Property')
```

Description

`out = get(obj)` returns a structure to `out` where each field name is the name of a common or device-specific property of `obj`. Each field of `out` contains the value of that property.

`out = get(obj.Channel(index))` returns a structure to `out` where each field name is the name of a channel property of `obj`. Each field of `out` contains the value of that property.

`out = get(obj,'Property')` returns the value of the specified property for `obj`. If `Property` 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.

`out = get(obj.Channel(index),'Property')` returns the value of the specified property for the specified channels contained by `obj`. If `index` is greater than 1, `out` will be an *m*-by-1 cell array of property values where *m* is equal to the length of `obj.Channel(index)`.

Example

Suppose you 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 `get` can be used to return property values.

```
chan = get(ai,'Channel')
out = get(ai, {'SampleRate','TriggerDelayUnits'})
out = get(ai)
get(chan, 'Units')
get(chan(1), {'HwChannel';'ChannelName'})
```

See Also

`set`

getdata

Purpose

Return data, time, and event information to the MATLAB workspace for an analog input or digital I/O object.

Syntax

```
data = getdata(obj);
data = getdata(obj, samples);
data = getdata(obj, type);
data = getdata(obj, samples, type);
[data, time] = getdata(...);
[data, time, abstime] = getdata(...);
[data, time, abstime, events] = getdata(...);
```

Arguments

`obj`	An existing analog input object.
`samples` (optional)	The number of samples to return. If `samples` is not specified, then the number of samples to return is given by the `SamplesPerTrigger` property.
`type` (optional)	Specifies the format of the returned data. If `type` is specified as `'native'`, then data is returned in the native data format of the device. If `type` is not specified, samples are returned as doubles.
`data`	An *m*-by-*n* array containing acquired data where *m* is the number of samples and *n* is the number of channels or lines.
`time` (optional)	An *m*-by-*1* array containing the time values for all *m* samples acquired. `time` = 0 corresponds to the time the first sample logged by the data acquisition engine. Time is measured continuously until the acquisition is stopped. Absolute times are available through the `TriggerTime` property.

`abstime` (optional)	The absolute time of the trigger.
`events` (optional)	A structure containing a list of events that have occurred up to the time of the `getdata` call. The possible events that can be returned are identical to those stored by the `EventLog` property.

Description

`getdata` is a blocking function that returns execution control to the MATLAB workspace when the requested number of samples are extracted from the data acquisition engine for each channel group member. Once the requested data is available, the `SamplesAvailable` property is 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 an amount of time given by the time it takes the engine to fill one data block plus the time specified by the `Timeout` property, then an error is returned.

`data = getdata(obj)` returns the number of samples specified by the `SamplesPerTrigger` property for each channel group member.

`data = getdata(obj, samples)` returns the specified number of data samples for each channel group member.

`data = getdata(obj, 'native')` returns data in the native format of the device for each channel group member. By default, data is returned as doubles.

`[data, time] = getdata(...)` returns data as sample-time pairs.

`[data, time, abstime] = getdata(...)` returns data as sample-time pairs and returns the absolute time of the trigger. The absolute time returned is identical to the first element of the `TriggerTime` vector. `abstime` is returned as a `datenum` value and can be converted to a string using `datestr`. Refer to "Calculating Time" on page 4-28 for more information about absolute and relative time.

`[data, time, abstime, events] = getdata(...)` returns data as sample-time pairs, returns the absolute time of the trigger, and returns a structure containing a list of events that occurred during the time span of the `getdata` call. The possible events that can be returned are identical to those stored by the `EventLog` property. Refer to Chapter 7, "Property Reference" for more information about `EventLog`.

Note: You can issue ^C (**Control-C**) while getdata is blocking. This will not stop the acquisition but will return control to MATLAB.

If getdata spans multiple triggers, a NaN is inserted into data and time between triggered sample sets thus increasing the length of data and time.

It is possible that data can be missed in the unlikely case that the engine cannot keep pace with the hardware device. You can monitor whether data is missed during the acquisition by specifying the name of an M-file for the DataMissedAction property.

Example

National Instruments

Suppose you create the analog input object ai for a National Instruments board and add hardware channels 0-3 to it. Additionally, a one second acquisition is configured with SampleRate set to 1000 and SamplesPerTrigger set to 1000. The object is then started.

```
ai = analoginput('nidaq', 1)
addchannel(ai, 0:3);
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.

```
[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 datenum value. and events is returned as a 3-by-1 structure array.

The three events stored correspond to the start event, trigger event, and stop event. For example, to return specific event information about the stop event, you must access the Type and Data subfields of events.

```
type = events(3).Type;
data = events(3).Data;
```

See Also

Functions

`flushdata`, `getsample`, `peekdata`, `putsample`

Properties

`EventLog`, `SamplesAvailable`, `SamplesPerTrigger`, `Timeout`

getsample

Purpose

Return one sample for each analog input channel group member or digital I/O line group member.

Syntax

```
samples = getsample(obj);
```

Description

`samples = getsample(obj)` immediately returns a row vector containing one sample for each channel group member contained by `obj`. The samples returned are not removed from the data acquisition engine.

Unlike `getdata`, `getsample` does not depend on data existing in the data acquisition engine. The function can be executed at any time after channels have been added to `obj`, and `Running` does not have to be `On`.

Using `getsample` is a good way to test your analog input configuration before issuing `start`.

Note: `getsample` is not supported for sound cards.

Example

National Instruments

Suppose you create the analog input object `ai` and add 8 channels to it.

```
ai = analoginput('nidaq', 1)
ch = addchannel(ai, 0:7)
```

The following command returns one sample for each channel.

```
getsample(ai)
```

See Also

Functions

`getdata`, `peekdata`

Purpose

Check for channels.

Syntax

```
ischannel(obj.Channel(index))
```

Description

`ischannel(obj.Channel(index))` returns a logical "1" if `obj.Channel(index)` is a channel. Otherwise, a logical "0" is returned. `ischannel` does not determine if channels are valid (associated with hardware). To check for valid channels, use the `isvalid` function.

Note: Typically, you use `ischannel` directly only when you are creating your own M-files.

Example

Suppose you create the function `func` for use with the Data Acquisition Toolbox. If `func` 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.

```
function func(obj.Channel)
% Determine if a channel was passed.
if ~ischannel(obj.Channel)
   error('The argument passed is not a channel.');
end
```

You can examine the Data Acquisition Toolbox M-files for examples using `ischannel`.

See Also

Functions

`delete`, `isvalid`

isline

Purpose

Check for lines.

Syntax

```
isline(obj.Line(index))
```

Description

`isline(obj.Line(index))` returns a logical "1" if `obj.Line(index)` is a line. Otherwise, a logical "0" is returned. `isline` does not determine if lines are valid (associated with hardware). To check for valid lines, use the `isvalid` function.

Note: Typically, you use `isline` directly only when you are creating your own M-files.

Example

Suppose you create the function `func` for use with the Data Acquisition Toolbox. If `func` 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 func(obj.Line)
% Determine if a line was passed.
if ~isline(obj.Line)
   error('The argument passed is not a line.');
end
```

You can examine the Data Acquisition Toolbox M-files for examples using `isline`.

See Also

Functions

`delete`, `isvalid`

Purpose

Check for valid device objects, channels, or lines.

Syntax

```
isvalid(obj)
isvalid(obj.Channel(index))
isvalid(obj.Line(index))
```

Description

`isvalid(obj)` returns a "1" if `obj` is a valid device object and a "0" if `obj` is not a valid device object. An invalid object exists in the workspace but does not exist in the engine. Therefore, invalid objects are no longer associated with any hardware and should be cleared from the workspace.

`isvalid(obj.Channel(index))` returns a logical array containing a "1" where the channels associated with `obj` are valid, and a "0" where they are not. Invalid channels are no longer associated with any hardware and should be cleared from the workspace.

`isvalid(obj.Line(index))` returns a logical array containing a "1" where the lines associated with `obj` are valid, and a "0" where they are not. Invalid lines are no longer associated with any hardware and should be cleared from the workspace.

Note: Typically, you use `isvalid` directly only when you are creating your own M-files.

Examples

Suppose you create the analog input object `ai` for a National Instruments board and add 8 channels to it.

```
ai = analoginput('nidaq', 1);
ch = addchannel(ai, 0:7);
```

To verify the device object is valid

```
isvalid(ai)
ans =
     1
```

To verify the channels are valid

```
isvalid(ch)
ans =
     1     1     1     1     1     1     1     1
```

If you delete a channel, then `isvalid` returns a logical “0” in the appropriate location

```
delete(ai.Channel(3))
isvalid(ch)
ans =
     1     1     0     1     1     1     1     1
```

Typically, you will use `isvalid` directly only when you are creating your own M-files. Suppose you create the function `func` for use with the Data Acquisition Toolbox. If `func` is passed the previously defined device object `ai` as an input argument

```
func(ai);
```

the first thing you should do in the function is check if `ai` is a valid device object.

```
function func(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 using `isvalid`.

See Also

Functions

`delete`, `ischannel`, `isline`

Purpose

Return the length of a data acquisition device object, channel group, or line group.

Syntax

```
length(obj)
length(obj.Channel)
length(obj.Line)
```

Description

`length(obj)` returns the length of data acquisition device object `obj`. It is equivalent to `maxsize(obj)`.

`length(obj.Channel)` returns the length of the channel group `obj.Channel`.

`length(obj.Line)` returns the length of the line group `obj.Line`.

Example

Suppose you create the analog input object `ai` for a National Instruments board and add 8 channels to it.

```
ai = analoginput('nidaq', 1);
ch = addchannel(ai, 0:7);
```

To find the length of the device object

```
length(ai)
ans =
     1
```

To find the length of the channel group

```
length(ch)
ans =
     1
```

See Also

Functions

`size`

load

Purpose

Load device objects into the MATLAB workspace.

Syntax

```
load file
load file, obj1, obj2,...
struct = load('file', 'obj1', 'obj2',...)
```

Description

`load file` returns all variables from the MAT-file `file` into the MATLAB workspace.

`load file, obj1, obj2,...` returns the specified device objects from the MAT-file `file` into the MATLAB workspace.

`struct = load('file', 'obj1', 'obj2',...)` returns the specified device objects from the MAT-file `file` as a structure to `struct` instead of directly loading them into the workspace. The field names in `struct` match the names of the device objects that were retrieved. If no objects are specified, then all variables existing in the MAT-file are loaded.

Note: `load` is not used to read in previously acquired data. You should use `daqread` for this purpose.

See Also

Functions

`daqread`, `save`

Purpose

Generate a list of channel or line names.

Syntax

```
names = makenames('prefix', index);
```

Arguments

`prefix`	The string to append to the front of `index`.
`index`	Numbers appended to the end of `prefix`. Any MATLAB vector syntax can be used to specify `index`. The numbers must be positive.
`names`	An *n*-by-1 cell array of channel names where *n* is the length of `index`.

Description

`makenames` generates a list of channel or line names. This list is to be passed as an input argument to the `addchannel` or `addline` function. The names are constructed by concatenating `prefix` and an `index` value. `prefix` must begin with a character and contain only characters and numbers. The number of hardware channels specified in `addchannel` or hardware lines specified in `addline` should agree with the number of indices specified in `makenames`.

Although `makenames` does not provide much flexibility in the descriptive names you can assign to channels or lines, it does provide you with a quick way to name a large number of channels or lines.

Example

National Instruments

Suppose you create the analog input object `AI` for a National Instruments board and add eight input channels. You can use the `makenames` function to define descriptive names for each channel.

```
AI = analoginput('nidaq', 1);
names = makenames('chan', 1:8);
```

`names` is an 8-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`

nidaq

Purpose

Display information about the National Instruments adaptor.

Syntax

```
help nidaq
```

Description

nidaq is the adaptor which allows National Instruments' NI-DAQ drivers and E-series boards to be used with the Data Acquisition Toolbox.

The device objects supported for National Instruments hardware include analog input, analog output, and digital i/o objects.

The supported device-specific properties for National Instruments hardware include `DriveAISenseToGround`, `NumMuxBoards`, and `TransferMode`. For more information about these properties, refer to Appendix A.

Example

To create an analog input object `AI` for a National Instruments board

```
AI = analoginput('nidaq', 1);
```

To list the device-specific properties for `AI`

```
set(AI)
```

See Also

Functions

`daqhelp`

Properties

`analoginput`, `analogoutput`, `digitalio`

Purpose

Convert device object, channels, or lines to MATLAB code and save to disk.

Syntax

```
obj2code(obj, 'file');
obj2code(obj, 'file', syntax);
obj2code(obj, 'file', 'all');
obj2code(obj, 'file', syntax, 'all');
```

Arguments

`obj`	An existing device object
`file`	The file that the MATLAB code is written to. The full pathname of `file` can be specified. If an extension is not specified for `file`, the `.m` extension is used.
`syntax` (optional)	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'` (optional)	If `'all'` is not specified, only properties that are not set to their default values are written to `file`. If `'all'` is specified, all properties are written to `file`.

Description

`obj2code(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`.

`obj2code(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`.

`obj2code(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, `'named'` uses named referencing (if defined).

`obj2code(obj, 'file', syntax, 'all')` 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, and 'named' uses named referencing (if defined). 'all' specifies that all properties are written to file.

Note: If UserData is not empty or if any of the action properties are set to a cell array of values, 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).

Example

Suppose you create the analog input object ai for a sound card, add two channels, and set values for several properties.

```
ai = analoginput('winsound');
addchannel(ai, 1:2);
set(ai, 'Tag', 'myai', 'TriggerRepeat', 4);
set(ai, 'StartAction', {'test', 2, magic(10)});
```

The following command writes MATLAB code to the files myai.m and myai.mat

```
obj2code(ai, 'myai.m', 'dot');
```

myai.m contains code which recreates the analog input code shown above using subscripted assignment for all properties that have their default values changed. Since StartAction is set to a cell array of values, this property appears in myai.m as

```
ai.StartAction = startaction
```

and is saved in myai.mat as

```
startaction = {'test', 2, magic(10)});
```

To load ai and assign the device object to a new variable ai_new

```
ai_new = myai
```

The associated MAT-file, myai.mat, is automatically loaded.

Purpose

Immediately return data to the MATLAB workspace for an analog input object.

Syntax

```
data = peekdata(obj, samples)
```

Arguments

`obj`	An existing analog input object.
`samples`	The number of samples to return.
`data`	An *m*-by-*n* matrix where *m* is the number of samples and *n* is the number of channels.

Description

`peekdata` is a nonblocking function that allows you to preview your data by immediately returning the requested number of samples to MATLAB. Since `peekdata` does not block execution control, data may be missed or repeated. If the number of samples requested is greater than the number of samples available in the current block, then whatever samples are available are returned along with a warning message stating that the requested samples are not available.

`peekdata` takes a "snapshot" of the most recent data and does not remove samples from the data acquisition engine. Therefore, the `SamplesAvailable` property value is not affected by the number of samples returned by `peekdata`. Unlike `getdata`, `peekdata` can be used without `Logging` being `On`.

`data = peekdata(obj, samples)` returns the latest number of `samples` to `data`.

Note: `peekdata` can be used only after the `start` command is issued.

Example

National Instruments

Suppose you create the analog input object `ai` for a National Instruments board, add eight input channels, and configure `ai` for a one second acquisition.

```
ai = analoginput('nidaq', 1)
addchannel(ai, 0:7)
set(ai, 'SampleRate', 1000);
set(ai, 'SamplesPerTrigger', 1000);
```

After issuing the start command, you can preview the data.

```
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. The data is not removed from the data acquisition engine.

See Also

Functions

getdata, getsample

Properties

SamplesAvailable

Purpose

Return property information for device objects, channels, or lines.

Syntax

```
info = propinfo(obj);
info = propinfo(obj, 'Property');
```

Description

`info = propinfo(obj)` returns information for all device object properties to `info`. The information is returned as a structure containing the fields shown below.

Property	Description
`Type`	The property data type (e.g., double, string)
`Constraint`	Constraints on property values (e.g., `None`, `Bounded`)
`ConstraintValue`	Property value constraint (e.g., range of valid values, elements of a list of valid values)
`DefaultValue`	The property default value
`ReadOnly`	If a property is read-only, a "1" is returned. Otherwise a "0" is returned.
`ReadOnlyRunning`	If a property cannot be set while running, a "1" is returned. Otherwise a "0" is returned.
`DeviceSpecific`	If the property is device-specific, then a "1" is returned. Otherwise a "0" is returned.

`info = propinfo(obj, 'Property')` returns information for the specified property to `info`.

Example

Sound Cards

Suppose you 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 following command displays the default value for the `SampleRate` property.

```
info1 = propinfo(ai);
info1.SampleRate.DefaultValue
```

The following command displays all the `propinfo` information for the `InputRange` property.

```
info2 = propinfo(ai.Channel);
info2.InputRange
```

See Also

Functions

`daqhelp`

Purpose

Queue data in the engine for eventual output to the D/A hardware.

Syntax

```
putdata(obj, data);
```

Description

`putdata(obj, data)` queues the data specified by `data` in the engine for eventual output to the D/A hardware. `data` must consist of a column of data for each channel group member. If any output data is not within the range of the `UnitsRange` property, then the data is clipped to the maximum or minimum value given by `UnitsRange`. An error is returned if a NaN is included in the data stream. The data type of `data` can be the native data type of the hardware.

`putdata` can be used to queue data in memory before the `start` command is issued, or it can be used to directly output data after `start` has been issued. In either case, no data is output until the trigger occurs. If `putdata` is called before `start`, then the specified data is queued to memory until:

- `MaxSamplesQueued` is reached. If this value is exceeded, an error is returned.
- The limitations of your hardware or computer are reached.

If the value of the `RepeatOutput` property is greater than 0, then all data queued before `start` is issued will be requeued (repeated) until the `RepeatOutput` value is reached. If `start` is issued wothout any data being queued, then an error is returned.

If `putdata` is called after start is issued, then the `RepeatOutput` property cannot be used. If `MaxSamplesQueued` is exceeded, `putdata` becomes a blocking function until there is enough space in the queue to add the additional data. If the time to queue the specified data exceeds the time it takes the engine to fill one data block plus the time specified by the `Timeout` property, then an error is returned.

As soon as a trigger occurs, samples can be output. The `SamplesOutput` property keeps a running count of the total number of samples per channel that have been output. Additionally, the `SamplesAvailable` property tells you how many samples are ready to be output from the engine per channel. When data is output, `SamplesAvailable` is reduced by the number of samples sent to the hardware.

Note: You will be able to issue ^C (**Control-C**) while putdata is blocking. This stops the data from being added to the queue. It does not stop data from being output.

Example

National Instruments

Suppose you create the analog output object ao for a National Instruments board and add two output channels to it.

```
ao = analogoutput('nidaq',1);
ch = addchannel(ao, 0:1);
```

Suppose the MATLAB variable data contains two columns of double precision numbers representing 1000 samples collected from two channels of a function generator. After the start command is issued, the following putdata call outputs all the data to the analog output device.

```
start(ao)
putdata(ao, data);
```

See Also

Functions

putsample

Properties

MaxSamplesQueued, RepeatOutput, Timeout

Purpose

Send one sample for each analog output channel group member or digital I/O line group member.

Syntax

```
putsample(obj, source)
```

Description

`putsample(obj, source)` immediately outputs the data from `source`. `source` must be a row vector containing one sample for each channel contained by `obj`.

Unlike `putdata`, `putsample` does not depend on the `Running` property being `On`. `putsample` can be executed at any time after channels have been added to `obj`.

Using `putsample` is a good way to test your analog output configuration before issuing `start`.

Note: `putsample` is not supported for sound cards.

Example

National Instruments

Suppose you create the analog output object `ao` for a National Instruments board and add two hardware channels to it.

```
ao = analogoutput('nidaq',1)
ch = addchannel(ao, 0:1)
```

If the MATLAB variable `data` is a row vector containing two data values, then following command outputs the first value of `data` to channel one, and the second value of `data` to channel two.

```
putsample(ao, data)
```

See Also

Functions

`putdata`

save

Purpose

Save device objects to a MAT-file.

Syntax

```
save file
save file, obj1, obj2,...
```

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 the MAT-file specified by `file`. If an extension is not specified for `file`, then a .MAT extension is used.

`save` can be used in the functional form as well as the command form shown above. When using the functional form, you must specify the file name and device objects as strings.

See Also

Functions

`load`

Purpose

Configure or display property values.

Syntax

```
set(obj)
props = set(obj)
set(obj, 'Property')
props = set(obj, 'Property')
set(obj, 'P1', V1, 'P2', V2,...)
set(obj, PN, PV)
set(obj, S)
```

Description

`set(obj)` displays all settable properties and their possible values.

`props = set(obj)` returns all settable properties and their possible values for `obj` 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, 'Property')` returns the valid values for `Property`.

`props = set(obj, 'Property')` returns the valid values for `Property` 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, 'P1', V1, 'P2', V2,...)` sets multiple property values with a single statement. Note that you can use structures, 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.

Examples

Suppose you 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 settable properties and their valid values

```
set(ai)
```

To set the value for the `SampleRate` property to 10000

```
set(ai, 'SampleRate', 10000)
```

There are two ways to set the value for the `SampleRate` and `InputType` properties

```
set(ai, 'SampleRate', 10000, 'TriggerType', 'Manual')
set(ai, {'SampleRate','TriggerType'}, {10000, 'Manual'})
```

You can also set different channel property values for multiple channels.

```
ch = ai.Channel(1:2)
set(ch, {'UnitsRange','ChannelName'}, {[-1 1] 'Name1'; ...
[-2 2] 'Name2'})
```

See Also

Functions

`get`, `setverify`

Purpose

Set and return the specified property value.

Syntax

```
Actual = setverify(obj, 'Property', Value)
Actual = setverify(obj.Channel(index), 'Property', Value)
```

Description

`Actual = setverify(obj, 'Property', Value)` sets `Property` to `Value` for `obj` and returns the actual property value to `Actual`.

`Actual = setverify(obj.Channel(index), 'Property', Value)` sets `Property` to `Value` for the channels specified by `index` and returns the actual property value to `Actual`.

`setverify` is equivalent to the commands

```
set(obj, 'Property', Value)
Actual = get(obj, 'Property')
```

and is particularly useful when setting the `SampleRate`, `InputRange`, and `OutputRange` properties since these properties can only be set to 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 the closest supported value is automatically selected. If you try to set `SampleRate` to a value outside the range of supported values, then an error is returned and the current `SampleRate` value is unchanged.

Although using `setverify` is not required for setting property values, it does provide a convenient way to verify the actual property value set by the data acquisition engine.

Example

Sound Card

Suppose you create the analog input object `ai` for a sound card and add one channel to it.

```
ai = analoginput('winsound')
ch = addchannel(ai, 1)
```

To set the input range to -1 to 1 volts

```
Actual = setverify(ai.Channel(1), 'InputRange', [-1 1])
```

If the input range was set to -2 to 2 volts, then depending on which sound card you are using, an error may be returned since most sound cards only support an input range of -1 to 1 volts.

National Instruments

Suppose you 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`.

```
ai = analoginput('nidaq', 1)
ch = addchannel(ai, 0:7)
setverify(ai, 'SampleRate', 10000)
```

Suppose you use `setverify` to set the input range for all channels contained by `ai` to -8 to 8 volts.

```
Actual = setverify(ai.Channel, 'InputRange', [-8 8])
```

you will see that `InputRange` was actually set to -10 to 10 volts, which is the closest valid value.

See Also

Functions

`get`, `propinfo`, `set`

Properties

`InputRange`, `OutputRange`, `SampleRate`

Purpose

Return the size of a data acquisition 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)
```

Description

`d = size(obj)` returns the two-element row vector `d = [m, n]` containing the number of rows and columns in the data acquisition device object `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 = size(obj.Channel)` returns the two-element row vector `d = [m, n]` containing the number of rows and columns in the channel group `obj.Channel`.

`[m1, m2, m3, ..., mn] = size(obj.Channel)` returns the length of the first `n` dimensions of the channel group `obj.Channel` to separate output variables. For example, `[m, n] = size(obj.Channel)` returns the number of rows to `m` and the number of columns to `n`.

`m = size(obj.Channel, dim)` returns the length of the dimension specified by the scalar `dim`. For example, `size(obj.Channel, 1)` returns the number of rows.

`d = size(obj.Line)` returns the two-element row vector `d = [m, n]` containing the number of rows and columns in the line group `obj.Line`.

`[m1, m2, m3, ..., mn] = size(obj.Line)` returns the length of the first `n` dimensions of the line group `obj.Line` to separate output variables. For

example, `[m, n] = size(obj.Line)` returns the number of rows to `m` and the number of columns to `n`.

`m = size(obj.Line, dim)` returns the length of the dimension specified by the scalar `dim`. For example, `size(obj.Line, 1)` returns the number of rows.

Example

Suppose you create the analog input object `ai` for a National Instruments board and add 8 channels to it.

```
ai = analoginput('nidaq', 1);
ch = addchannel(ai, 0:7);
```

To find the size of the device object

```
size(ai)
ans =

     1     1
```

To find the size of the channel group

```
size(ch)
ans =

     8     1
```

See Also

Functions

`length`

Purpose

Start the execution of a device object.

Syntax

```
start(obj)
```

Description

`start(obj)` initiates the execution of a device object. When `start` is issued:

1. The M-file specified for `StartAction` is executed.
2. `Running` is set to `On`.

If `start` is called and no channels or lines are defined for the device object, an error is returned.

Although the device object is executing, data is not necessarily logged for analog input objects or sent for analog output objects. Data logging or sending depends on the `TriggerType` property. If the `TriggerType` value is `Immediate`, data logging or sending occurs immediately. If the `TriggerType` value is `Manual`, then data logging or sending occurs when the `trigger` function is issued. Otherwise, data logging or sending occurs when the specified `TriggerCondition` is met. The `start` action is recorded in the `EventLog` property.

A device object can stop executing under one of these conditions:

- A stop command is issued.
- For analog input objects, `SamplesAcquired` equals the product of the `SamplesPerTrigger` and `TriggerRepeat` properties, or for analog output objects, all samples were sent out.
- The `Timeout` value is reached.
- An error occurs.

See Also

Functions

`stop`, `trigger`

Properties

`EventLog`, `Timeout`, `TriggerType`

stop

Purpose

Stop the execution of a device object.

Syntax

```
stop(obj)
```

Description

`stop(obj)` terminates the execution of a device object. When `stop` is issued:

1. `Running` is set to `Off`.
2. `Logging` or `Sending` is set to `Off`.
3. The M-file specified for `StopAction` is executed.

A device object can also stop executing under one of these conditions:

- For analog input objects, `SamplesAcquired` equals the product of the `SamplesPerTrigger` and `TriggerRepeat` properties, or for analog output objects, all samples were sent out.
- The `Timeout` value is reached.
- An error occurs.

For analog input objects, `stop` must be called when `TriggerRepeat` or `SamplesPerTrigger` is set to `Inf`. The `stop` action is recorded in the `EventLog` property.

See Also

Functions

`start`, `trigger`

Properties

`EventLog`, `Timeout`

Purpose

Manually execute a trigger.

Syntax

```
trigger(obj);
```

Description

`trigger(obj)` manually executes a trigger. After `trigger` is issued:

- The absolute time of the trigger event is recorded in the `TriggerTime` property.
- `Logging` or `Sending` is set to `On`.
- The M-file specified by `TriggerAction` is executed.

`trigger` can only be invoked if `TriggerType` is set to `Manual`, `Running` is `On`, and `Logging` is `Off`. The `trigger` action is recorded in the device object's `EventLog` property.

See Also

Functions

`start`, `stop`

Properties

`TriggerType`

winsound

Purpose

Display information about the sound card adaptor.

Syntax

```
help winsound
```

Description

`winsound` is the adaptor which allows sound cards and Windows multimedia services to be used with the Data Acquisition Toolbox.

The device objects supported for sound cards include analog input and analog output objects.

The supported device-specific property for sound cards is `BitsPerSample`. For more information about this property, refer to Appendix A.

Example

To create an analog input object `AI` for a sound card

```
AI = analoginput('winsound');
```

To list the device-specific properties for `AI`

```
set(AI)
```

See Also

Functions

`daqhelp`

Properties

`analoginput`, `analogoutput`

7

Property Reference

The Property Description Format

Each Data Acquisition Toolbox property description contains some or all of this information:

- The property name
- A description of the property
- The property characteristics, including:
  - Device Objects – which device objects the property is associated with

    The device objects supported by the Data Acquisition Toolbox include analog input (AI), analog output (AO), and digital I/O (DIO) objects.
  - Class – whether it is a common property, or a channel or line property

    Common properties apply to all channels or lines contained by the device object.

    Channel (line) properties can be set on a per-channel (per-line) basis.
  - Access – whether the property is read/write or read-only

    Read/write property values can be returned with the `get` command, and configured with the `set` command.

    Read-only property values can be returned with the `get` command, but cannot be configured with the `set` command.
  - Data Type – the property data type
  - Modify while Running – whether a property value can be set while the device object is running
- Valid property values including the default value

  When property values are given by an enumerated list, the default value is usually indicated by "{ }". If a default value is not indicated, then it is determined by the hardware. Default values for some properties are calculated by the data acquisition engine, while others are determined by the hardware driver.

  If there are device-specific enumerated values, they are listed separately.
- An example using the property
- Related properties and functions

  Functions appear in lower case, while properties appear in mixed case.

Analog Input Properties

Analog input 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. Starting on page 7-12, these properties are listed alphabetically.

Note: Depending on the hardware device you are using, additional properties and values may be present. Device-specific values associated with base properties are given in this chapter, while device-specific properties are given in Appendix A.

Common Properties

Common properties apply to all channels contained by the analog input object. The common analog input properties are grouped into seven functional categories as shown below.

Analog Input Basic Setup Properties	
`Channel`	Contains all the channels associated with the device object as created with `addchannel`
`SamplesPerTrigger`	Specifies the number of samples to acquire per channel for each trigger issued
`SampleRate`	Specifies the samples per second of the analog input hardware device
`TriggerType`	Specifies the type of trigger to be issued

Analog Input Logging Properties	
LogFileName	Specifies the name of the disk file that data and events are written to
LoggingMode	Specifies where data is logged
LogToDiskMode	Specifies if data and events are saved in one disk file or multiple disk files

Analog Input Trigger Properties	
TriggerChannel	Contains the hardware channels serving as trigger sources
TriggerCondition	Specifies the condition to be satisfied before a trigger is issued
TriggerConditionValue	Specifies the value of a trigger condition
TriggerDelay	Specifies the delay value for data logging
TriggerDelayUnits	Specifies the trigger delay as samples or seconds
TriggerRepeat	Specifies the number of times to repeat the trigger
TriggerTime	Indicates the observed absolute times when triggers occurred

Analog Input Status Properties	
Logging	Indicates whether data is logged to memory and/or a disk file
Running	Indicates whether the device object and data acquisition engine are running
SamplesAcquired	Indicates the number of samples acquired per channel
SamplesAvailable	Indicates the number of samples available per channel in the data acquisition engine

Analog Input Hardware Configuration Properties	
ChannelSkew	Specifies the time, in seconds, between consecutive scanned hardware channels
ChannelSkewMode	Specifies the hardware-specific channel skew
ClockSource	Specifies which clock is used to govern the acquisition rate
InputType	Specifies the hardware channel configuration, e.g., single-ended, differential, or AC-Coupled

Analog Input Action Properties	
DataMissedAction	Specifies the M-file to be executed when data is missed
InputOverRangeAction	Specifies the M-file to be executed when an input channel exceeds its valid range
RuntimeErrorAction	Specifies the M-file be executed when a runtime error occurs
SamplesAcquiredAction	Specifies the M-file to be executed every time the number of samples specified by SamplesAcquiredActionCount are acquired
SamplesAcquiredActionCount	Specifies the number of samples to acquire for each channel group member before the M-file specified by SamplesAcquiredAction is executed
StartAction	Specifies the M-file to be executed just before the hardware device and data acquisition engine start
StopAction	Specifies the M-file to be executed just after the hardware device and data acquisition engine stop
TimerAction	Specifies the M-file to be executed after the time specified by TimerPeriod has passed

Analog Input Action Properties	
TimerPeriod	Specifies the time, in seconds, between calls to TimerAction
TriggerAction	Specifies the M-file to be executed just after a trigger occurs

Analog Input General Purpose Properties	
BufferingConfig	Specifies the per-channel allocated memory
BufferingMode	Specifies if memory is automatically by the engine or manually
EventLog	Contains information related to certain events that occur for a device object
Name	Contains the name of the device object
Tag	Stores a device object label
Timeout	Specifies the additional time to wait for a data block to fill
Type	Indicates the device object type
UserData	Stores data that you want to associate with the device object

Channel Properties

Channel properties can be set on a per-channel basis. The analog input channel properties are given below.

Analog Input Channel Properties	
ChannelName	Specifies a descriptive channel name
HwChannel	Specifies the hardware channel ID
Index	Indicates the MATLAB index number of a channel
InputRange	Specifies the range of data input to the analog input hardware device

Analog Input Channel Properties	
`Parent`	Contains the parent (device) object of the channel
`SensorRange`	Specifies the range of data you expect from your sensor
`Units`	Specifies the engineering units
`UnitsRange`	Specifies the engineering units range of the data input into MATLAB

Analog Output Properties

Analog output 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. Starting on page 7-12, these properties are listed alphabetically.

Note: Depending on the hardware device you are using, additional properties and values may be present. Device-specific values associated with base properties are given in this chapter, while device-specific properties are given in Appendix A.

Common Properties

Common properties apply to all channels contained by the analog output object. The common analog output properties are grouped into six functional categories as shown below.

Analog Output Basic Setup Properties	
`Channel`	Contains all the channels associated with the device object as created with `addchannel`.
`SampleRate`	Specifies the sampling rate of the analog output device
`TriggerType`	Specifies the type of trigger to be issued

Analog Output Trigger Properties	
`TriggerTime`	Indicates the observed times when triggers occurred

Analog Output Status Properties	
Running	Indicates whether the hardware device and data acquisition engine are running
SamplesAvailable	Indicates the number of samples available per channel in the data acquisition engine
SamplesOutput	Indicates the number of samples per channel that have been sent to the hardware device
Sending	Indicates data is being output to the hardware device

Analog Output Hardware Configuration Properties	
OutOfDataMode	Specifies the last value to write to the analog output device when data is no longer being sent.
ClockSource	Specifies which clock is used to govern the acquisition rate

Analog Output Action Properties	
OutOFDataAction	Specifies the M-file to be executed when data is no longer sent to the analog output device
RuntimeErrorAction	Specifies the M-file to be executed when a runtime error has occurred
SamplesOutputAction	Specifies the M-file to be executed every time the number of samples specified by SamplesOutputActionCount are sent
SamplesOutputAction Count	Specifies the number of samples to output before the function specified by SamplesOutputAction is executed
StartAction	Specifies the M-file to be executed just before the hardware device and data acquisition engine start

Analog Output Action Properties	
StopAction	Specifies the M-file to be executed just after the hardware device and data acquisition engine stop
TimerAction	Specifies the M-file to be executed after the time specified by TimerPeriod has passed
TimerPeriod	Specifies the time, in seconds, between TimerAction callbacks
TriggerAction	Specifies the M-file to be executed just after a trigger occurs

Analog Output General Purpose Properties	
BufferingConfig	Specifies the per-channel allocated memory
BufferingMode	Specifies if memory is automatically by the engine or manually
EventLog	Contains information related to certain events
MaxSamplesQueued	Specifies the maximum number of samples allowed in the analog output queue
Name	Contains the name of the device object
RepeatOutput	Specifies the number of times to send queued data to the analog output device
Tag	Stores a device object label
Timeout	Specifies the additional time to wait for a data block to be output
Type	Indicates the device object type
UserData	Stores data that you want to associate with the device object

Channel Properties

Channel properties can be set on a per-channel basis. The analog output channel properties are given below.

Analog Output General Properties	
`ChannelName`	Specifies a descriptive channel name
`DefaultChannelValue`	Specifies the value to write to the analog output hardware device when data has stopped being sent
`HwChannel`	Specifies the hardware channel ID
`Index`	Indicates the MATLAB index number of a channel
`OutputRange`	Specifies the range of the D/A hardware
`Parent`	Contains the parent (device) object of the channel
`Units`	Specifies the engineering units
`UnitsRange`	Specifies the engineering units range of the data

BufferingConfig

`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 (block size) × (number of blocks) × (number of channels) × (native data type).

Memory can be allocated either automatically by the engine or manually depending on the value of `BufferingMode`. 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 needs of the engine. 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 requested exceeds the available memory, an out-of-memory error is returned.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read/write
Data Type	Vector of doubles
Modify while Running	No

Values

The default value is determined by the engine.

See Also

Properties

`BufferingMode, SampleRate, SamplesPerTrigger`

Functions

`daqmem`

BufferingMode

`BufferingMode` specifies how memory is allocated. If `BufferingMode` is configured to its default value `Auto`, then the data acquisition engine automatically allocates the required memory. If `BufferingMode` is set to `Manual`, then you must manually allocate memory with the `BufferingConfig` property.

For most data acquisition applications, memory should be automatically allocated by the engine.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read/write
Data Type	Enumerated list
Modify while Running	No

Values

`{Auto}`	Memory is allocated by the data acquisition engine
`Manual`	Memory is allocated manually

See Also

Properties

`BufferingConfig`

Functions

`daqmem`

Channel

`Channel` is a vector of all the hardware channels contained by an analog input (AI) or analog output (AO) object.

A newly created AI or AO object does not contain any channels. Therefore, `Channel` is initially an empty vector. The size of `Channel` increases as channels are added using the `addchannel` function, and decreases as channels are deleted using the `delete` function.

`Channel` may be used to reference one or more channels by index. You must use `Channel` when assigning or returning channel property values.

If you are using scanning hardware, then the scan order is indicated by the `HwChannel` property. To change the scan order, you must specify a permutation of the hardware indices with `Channel`.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read/write
Data Type	Vector of channels
Modify while Running	No

Values

The default value is an empty column vector.

Example

Suppose you create an analog input object for a National Instruments card and add three hardware channels.

```
ai = analoginput('nidaq',1)
addchannel(ai, 0:2)
```

To set a property value for a specific channel, you must reference the channel by its index using the `Channel` property.

```
chans = ai.Channel(1);
set(chans, 'InputRange', [-5 5]);
```

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 must specify the appropriate permutation of the hardware indices.

```
ai.Channel([1 2 3]) = ai.Channel([2 1 3])
```

See Also

Properties

`HwChannel`

Functions

`addchannel`, `delete`

ChannelName

`ChannelName` specifies a descriptive name for an analog input or analog output channel. Channel names are not required to be unique.

If a channel name is defined, then you can reference that channel by its name. If channel name is not defined, then it must be referenced by its index. You can use the `makenames` function to define descriptive names. This function is particularly useful when defining names for many channels.

Characteristics

Device Objects	AI, AO
Class	Channel
Access	Read/write
Data Type	String
Modify while Running	No

Values

The default value is an empty string.

Example

Suppose you create an analog input object for a sound card and add two hardware channels.

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

You can assign a descriptive name to the first channel contained by ai.

```
Chan1 = ai.Channel(1)
set(Chan1, 'ChannelName', 'Joe')
```

You can now reference this channel by name instead of index.

```
set(ai.Joe,'InputRange', [-1 1]);
```

See Also

Functions

makenames

ChannelSkew

ChannelSkew specifies the time, in seconds, between consecutive scanned hardware channels. This property is settable only when ChannelSkewMode is set to Manual. If ChannelSkewMode is set to Minimum or Equisample, then ChannelSkew contains the appropriate read-only value.

ChannelSkew is settable only for scanning hardware and not for simultaneous sample and hold (SSH) hardware. For SSH hardware, the only valid ChannelSkew value is 0. SSH hardware includes Hewlett-Packard devices and sound cards.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read/write (depends on ChannelSkewMode value)
Data Type	Double
Modify while Running	No

Values

The default value is given by the hardware driver.

See Also

Properties

ChannelSkewMode

ChannelSkewMode

ChannelSkewMode specifies how the channel skew is determined. The channel skew is defined as the time between consecutive scanned hardware channels. If ChannelSkewMode is set to Minimum, then the minimum channel skew supported by the hardware is used. If ChannelSkewMode is set to Equisample, the channel skew is given by (sampling rate × number of channels)$^{-1}$. If ChannelSkewMode is set to Manual, then the channel skew is set by the ChannelSkew property.

Since ChannelSkew is settable only for scanning hardware and not for simultaneous sample and hold (SSH) hardware, ChannelSkewMode can only be set to None for SSH hardware. In this case, ChannelSkew is 0. SSH hardware includes Hewlett-Packard devices and sound cards.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read/write
Data Type	Enumerated list
Modify while Running	No

Values

None	This is the only supported value for SSH hardware.
Minimum	The channel skew is set to the minimum supported value. This is the default for scanning hardware.
Equisample	The channel skew is set to (sampling rate × number of channels)$^{-1}$.
Manual	The channel skew is set to the value specified by ChannelSkew.

The default value is determined from the hardware driver.

See Also

Properties

`ChannelSkew`, `SampleRate`

ClockSource

`ClockSource` specifies the clock used to govern the acquisition rate.

Characteristics

Device Objects	AI, AO
Class	N/A
Access	Read/write
Data Type	Enumerated list
Modify while Running	No

Values

`Internal`	The internal PC clock is used.

National Instruments Analog Input Values

`ExternalSampleCtrl`	Externally control the channel clock
`ExternalScanCtrl`	Externally control the scan clock
`ExternalSampleAndScanCtrl`	Externally control the channel and scan clocks

National Instruments Analog Output Values

`{Internal}`	The internal clock
`Delay`	The delay clock
`DelayPrescalar1`	The delay clock prescalar 1
`DelayPrescalar2`	The delay clock prescalar 2

DataMissedAction

`DataMissedAction` specifies the M-file function to be executed when data has been missed. Data can be missed if the data acquisition engine cannot keep pace with the hardware device. If `DataMissedAction` is called, the default action is to stop the acquisition.

Characteristics

Device Objects	AI
Class	Common
Access	Read/write
Data Type	String
Modify while Running	Yes

Values

The default value is `stop`.

DefaultChannelValue

`DefaultChannelValue` specifies a vector of values to write to the analog output hardware device when data has stopped being output. If one value is specified, then this value is written for all defined output channels. If supported by the hardware device, you can specify a different value for each defined output channel.

`DefaultChannelValue` is used only when `OutOfDataMode` is set to `DefaultValue`.

Characteristics

Device Objects	AO
Class	Channel
Access	Read/write

Data Type	Vector of doubles
Modify while Running	No

Values

The default value is a vector of zeros.

See Also

Properties

`OutOfDataMode`

EventLog

`Eventlog` is a structure array that stores certain event information in the `Type` and `Data` fields.

The `Type` field stores the event type. For analog input objects, the logged event types are start, stop, trigger, runtime error, input over-range, and data missed events. For analog output objects, the logged event types are start, stop, trigger, runtime error, and out-of-data events. Timer events, data-available events (AI), and data-output events (AO) are not logged.

The `Data` field stores event-specific information associated with the event type in several subfields. For all stored events, `Data` contains the `TimeStamp` and `SampleStamp` subfields. `TimeStamp` contains the absolute time the event occurred and `SampleStamp` contains the number of samples processed by the data acquisition engine at the time the event occurred. Other event-specific subfields are included in `Data`. For a description of these subfields, refer to "Configuring Events and Actions" on page 4-48.

`EventLog` can store a maximum of 1024 events.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read-only

Data Type	Structure array
Modify while Running	N/A

Values The default value is an empty structure array

Example Suppose you perform the following one-second acquisition for a National Instruments board.

```
ai = analoginput('nidaq', 1);
chans = addchannel(ai, 0:3);
set(ai,'SampleRate', 1000);
ActualRate = get(ai,'SampleRate')
duration = 1; % One second acquisition
set(ai,'SamplesPerTrigger', ActualRate*duration)
start(ai)
data=getdata(ai);
```

You can examine the logged event types

```
events = ai.EventLog
events.Type
```

and the data associated with the events

```
events.Data
```

HwChannel

`HwChannel` contains the hardware channel IDs. For National Instruments hardware, channel ID's are zero-based. For Hewlett-Packard hardware and sound cards, channel ID's are one-based.

The elements of `HwChannel` are defined when hardware channels are associated with a device object with the `addchannel` function. If you are using scanning hardware, then `HwChannel` determines the scanning order. Namely, the first `HwChannel` element is sampled first, the second `HwChannel` element is sampled second, and so on.

Each hardware channel has an associated MATLAB index. To assign or return property values for individual channels, or to rearrange the scan order, you must address the hardware channels by their index with the `Channel` property.

A hardware channel can be assigned to multiple device objects, or multiple times to the same device object. For sound cards however, hardware channels cannot be assigned multiple times to the same device object.

Characteristics

Device Objects	AI, AO
Class	Channel
Access	Read/write
Data Type	Array of doubles
Modify while Running	No

Values

The values are specified when channels are added to the device object with the `addchannel` function.

Example

Suppose you create the analog input object `ai` for a National Instruments board and add the first four hardware channels

```
ai = analoginput('nidaq', 1)
addchannel(ai, 0:3)
```

These channels are automatically assigned the MATLAB indices 1-4, and the scan order is given by `HwChannel`. Namely, channel 1 is sampled first, channel 2 is sampled second, and so on.

See Also

Properties

`Channel`, `ChannelName`, `Index`

Index

`Index` contains the MATLAB index of a hardware channel.

Index provides you with a way of accessing the channel index programmatically. Without this property, you would use the Data Acquisition Toolbox display method to get the channel index, and then incorporate the information into your function or script.

Characteristics

Device Objects	AI, AO
Class	Channel
Access	Read-only
Data Type	Double
Modify while Running	N/A

Values

The values are automatically defined when channels are added to the device object with the addchannel function.

Example

Suppose you create an analog input object ai for a sound card and add two hardware channels.

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

The corresponding MATLAB indices are 1 and 2, respectively. These indices can be accessed with Index.

```
Index1 = ai.Channel(1).Index
Index2 = ai.Channel(2).Index
```

See Also

Properties

Channel, ChannelName, HwCHannel

InputOverRangeAction

InputOverRangeAction specifies the M-file function to be executed when data sampled by an analog input channel group member exceeds its valid range. The valid range for a channel is given by the InputRange property.

Over-range checking is enabled for all channel group members only when a value is specified for `InputOverRangeAction` and the analog input object is running (`Running` is `On`). If a value is not specified for `InputOverRangeAction`, then over-range detection will not occur.

When an over-range condition is detected, a typical action to take is to determine which channel(s) went over-range. You can determine which channels experienced an over-range condition by looking at the event passed to the action function.

Characteristics

Device Objects	AI
Class	Common
Access	Read/write
Data Type	String
Modify while Running	Yes

Values

The default value is an empty string.

Example

When a channel group member experiences an over-range condition, you may want to display a message. In the example code below, `overrangedisp` is an M-file script that displays a simple message to the MATLAB environment.

```
InputOverRangeAction = 'overrangedisp'
```

The `overrangedisp` function is shown below.

```
function overrangedisp(obj, events)
% Display over-range message
chans = events.Data.ChannelIndex;
disp(['For the object ',obj.Name])
disp(['These channel(s) went over-range: ',num2str(chans)])
```

See Also

Properties

`EventLog`, `InputRange`

InputRange

`InputRange` specifies the range of the analog input hardware device. For many devices, this range is expressed in terms of the gain and polarity. Therefore, if you set `InputRange` to a particular value, then the gain and/or polarity are automatically modified accordingly.

Hardware devices have a finite number of `InputRange` values that can be set. If an `InputRange` value is specified but does not match a valid value, then the closest supported value is automatically selected by the engine.

Since `InputRange` can be set to a value that differs from the one specified, you should return the actual input range for each channel.

```
ActualRange = get(AI.Channel(index), 'InputRange');
```

Alternatively, you can use the `setverify` function. This function sets the input range value and then returns the actual value that is set.

```
ActualRange = setverify(AI.Channel(index), 'InputType', Value)
```

Characteristics

Device Objects	AI
Class	Channel
Access	Read/write
Data Type	Vector of doubles
Modify while Running	Yes

Values

The default value is supplied by the hardware driver.

See Also

Functions

`setverify`

Properties

`SensorRange`, `UnitsRange`

InputType

`InputType` specifies the analog input hardware channel configuration. For sound cards, `InputType` can only be `AC-Coupled`. For National Instruments devices, `InputType` can be `SingleEnded`, `Differential`, or `NonReferencedSingleEnded`.

Depending on the hardware you are using, changing the `InputType` property value may change the number of channels available for the analog input object. Furthermore, if the `InputType` value is changed while the analog input object is running, then all channels are deleted and a warning is issued if the number of available channels is reduced.

Note: You should use the differential mode whenever possible since this mode helps to reduce noise.

Characteristics

Device Objects	AI
Class	Common
Access	Read/write
Data Type	Enumerated list
Modify while Running	No

Values

Sound Card Values

`AC-Coupled`	The input is coupled so that constant (DC) signal levels are suppressed.

National Instruments Values

`SingleEnded`	Channels are configured as single-ended

`{Differential}`	Channels are configured as differential
`NonReferencedSingleEnded`	This channel configuration is used when the input signal has its own ground reference. The input signal's ground reference is connected to AISENSE, which is tied to the negative input of the instrumentation amplifier.

LogFileName

`LogFileName` specifies the name of the disk file that data and events are written to. The log file name can be any string. If an extension is not specified, then `.daq` is used.

To log data and events, you must specify a value for `LogFileName`. Data and events are logged to a disk file only for analog input objects. For other device objects, data and events cannot be written to a disk file. For all device objects, information can be retrieved for certain events with the `EventLog` property.

Data and events are written out to the specified file for all channels contained by the analog input object. Just before the data is logged, a header is appended to the start of the file. The header contains all the properties of the analog input object, the date, and the absolute time when the data acquisition started.

Characteristics

Device Objects	AI
Class	Common
Access	Read/write
Data Type	String
Modify while Running	Yes, but not settable while logging to disk.

Values

The default value is an empty string.

See Also

Properties

EventLog, Logging, LogToDisk, LogToDiskMode

Logging

Logging indicates whether data is logged. Data can be logged to memory and/or a disk file.

Logging must be On for acquired data to be saved to the MATLAB workspace. However, if you only want to check the validity of acquired data without saving it, you can use the peekdata or getsample functions and Logging can be Off. Logging is automatically set to On when a trigger occurs. Logging is set to Off when the requested samples are acquired, an error occurs, or a stop command is issued.

Events are logged to the EventLog property regardless of the state of the Logging property.

Characteristics

Device Objects	AI
Class	Common
Access	Read-only
Data Type	Boolean
Modify while Running	N/A

Values

On	Acquired data is logged to memory and/or a disk file.
{Off}	Acquired data is not logged.

See Also

Properties

LogFileName, LoggingMode, LogToDiskMode, Running

LoggingMode

`LoggingMode` specifies where data is logged. Data can be logged to memory and/or to a disk file. If logging to memory, then acquired data is temporarily stored in the data acquisition engine. If logging to disk, a value for `LogFileName` must be specified, and acquired data and events are streamed to the specified log file.

Data stored in memory can be returned to the MATLAB workspace with the `getdata` function. Data stored to a disk file can be returned to the MATLAB workspace with the `daqread` function.

Characteristics

Device Objects	AI
Class	Common
Access	Read/write
Data Type	Enumerated list
Modify while Running	No

Values

`Disk`	Acquired data is logged to disk.
`{Memory}`	Acquired data is logged to memory.
`Disk&Memory`	Acquired data is logged to disk and memory.

See Also

Properties

`LogFileName`, `Logging`, `LogToDiskMode`

Functions

`daqread`, `getdata`

LogToDiskMode

LogToDiskMode specifies whether data and events are saved to one disk file or multiple disk files. If LogToDiskMode is set to Overwrite, then the output 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 the disk file name follows these rules:

- If no number is specified as part of LogFileName, then two digits are appended to the end of the file name starting at "01". For example, if LogFileName is specified as data.daq, then data01.daq is used as the output file.
- If a number is specified as part of LogFileName, then the number will be incremented by one for the next output file.

If an error occurs during data logging, such as the disk fills up, an error message is returned and data logging is stopped.

Separate analog input objects must be logged to separate files. Data stored to a disk file can be returned to the MATLAB workspace with the daqread function.

Characteristics

Device Objects	AI
Class	Common
Access	Read/write
Data Type	Enumerated list
Modify while Running	No

Values

{Overwrite}	The log file is overwritten.
Index	Multiple log files are written, each with an indexed file name.

See Also

Properties

LogFileName, Logging, LoggingMode

Functions

daqread

MaxSamplesQueued

MaxSamplesQueued indicates the maximum number of samples allowed in the analog output queue.

The value of MaxSamplesQueued can affect the behavior of putdata. For example, if queued data exceeds the value of MaxSamplesQueued, putdata becomes a blocking function until there is enough space in the queue to add the additional data.

Characteristics

Device Objects	AO
Class	Common
Access	Read-only
Data Type	Double
Modify while Running	N/A

Values

The value is calculated by the data acquisition engine.

See Also

Functions

putdata

Name

Name specifies the name of the device object.

After the device object is created, Name is automatically assigned a descriptive name that is produced by concatenating the name of the adaptor, the device ID, and the device object type.

The value of Name can be changed at any time.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read/write
Data Type	String
Modify while Running	Yes

Values

The value is defined after the device object is created.

Example

Suppose you create an analog input object for the sound card:

```
AI = analoginput('winsound')
```

The `Name` property is automatically assigned the value

```
winsound0-AI
```

OutOfDataAction

`OutOfDataAction` specifies the M-file function to be executed when data is no longer being sent to the analog output device.

Characteristics

Device Objects	AO
Class	Common
Access	Read/write
Data Type	String
Modify while Running	Yes

Values

The default value is an empty string.

OutOfDataMode

`OutOFDataMode` specifies the value to send to the analog output hardware device when an out-of-data condition occurs. An out-of-data condition indicates that data is no longer being sent to the analog output device.

Characteristics

Device Objects	AO
Class	Common
Access	Read/write
Data Type	Enumerated list
Modify while Running	No

Values

`{Hold}`	Hold the last value output.
`Stop`	Stop the device. The value written to the D/A channels is device-dependent. For some devices, the D/A value drifts to zero, while for other devices, the last value is held until it is reset.
`DefaultValue`	The value specified by the `DefaultChannelValue` property is sent to the D/A converter.

See Also

Properties

`DefaultChannelValue`

OutputRange

`OutputRange` specifies the range of the analog output hardware device. For many devices, this range is expressed in terms of gain and polarity. Therefore, if you set `OutputRange` to a particular value, then gain and/or polarity are automatically modified accordingly.

Hardware devices have a finite number of `OutputRange` values that can be set. If an `OutputRange` value is specified but does not match a valid value, then the closest supported value is automatically selected by the engine.

Since `OutputRange` can be set to a value that differs from the one specified, you should return the actual output range for each channel.

```
ActualRange = get(AO.Channel(index), 'OutputRange');
```

Alternatively, you can use the `setverify` function. This function sets the output range value and then returns the actual value set.

```
ActualRange = setverify(AO.Channel(index), 'OutputRange', Value)
```

Characteristics

Device Objects	AO
Class	Channel
Access	Read/write
Data Type	Vector of doubles
Modify while Running	Yes

Values

The default value is determined by the hardware driver.

See Also

Functions

`setverify`

Properties

`UnitsRange`

Parent

`Parent` contains the parent (device) object of the channel.

Characteristics

Device Objects	AI, AO
Class	Channel

Access	Read-only
Data Type	Device object
Modify while Running	N/A

Values The value is defined when channels are added to the device object.

RepeatOutput

`RepeatOutput` specifies the number of additional times queued data is to be sent to the analog output device.

All data queued before the `start` function is issued will be requeued until the `RepeatOutput` value is reached. While the data is being output, you will not be able to add additional data to the queue.

Characteristics

Device Objects	AO
Class	Common
Access	Read/write
Data Type	Double
Modify while Running	No

Values The default value is zero.

Running

`Running` indicates whether the device object and data acquisition engine are running.

When `Running` is `On`, data can be acquired from an analog input device or sent to an analog output device. When `Running` is `Off`, you can still acquire one

sample for each channel contained by an analog input device using the `getsample` function, or send one sample for each channel contained by an analog output device using the `putsample` function.

`Running` is automatically set to `On` once the `start` command is issued. `Running` is automatically set to `Off` once the `stop` command is issued, the specified data is acquired or sent, or a runtime error occurs.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read-only
Data Type	Enumerated list
Modify while Running	N/A

Values

`{Off}`	The hardware device and data acquisition engine are not running.
`On`	The hardware device and data acquisition engine are running.

See Also

Properties

`Logging`

RuntimeErrorAction

`RuntimeErrorAction` specifies the M-file function to be executed when a runtime error occurs. Runtime errors can come from the hardware device or the data acquisition engine. A timeout is considered a runtime error.

Characteristics

Device Objects	AI, AO
Class	Common

Access	Read/write
Data Type	String
Modify while Running	Yes

Values The default value is an empty string.

See Also **Properties**
`Timeout`

SampleRate

`SampleRate` specifies the per-channel rate (in samples/second) that the analog input or analog output hardware can digitize data.

Data acquisition devices have a finite (though often large) number of valid sampling rates. To find out the range of sample rates supported by your board, you can use the `propinfo` function.

If you specify a sampling rate that does not match one of the valid device values, then the data acquisition engine automatically selects the closest supported sampling rate that is within 1% of the requested rate. If there is no valid sampling rate within this tolerance, the next highest value is selected. If a higher value is not available, then an error is returned.

Since `SampleRate` can be set to a value that differs from the one specified with the `set` function, you should return the actual sampling rate.

```
ActualRate = get(AI, 'SampleRate');
```

Alternatively, you can use the `setverify` function. This function sets the `SampleRate` value and then returns the actual value that is set.

```
ActualRate = setverify(AI, 'SampleRate', Value)
```

Note: Whenever the SampleRate value is changed, the BufferingConfig values are recalculated by the engine.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read/write
Data Type	Double
Modify while Running	No

Values

The default value is obtained from the hardware driver.

Example

Suppose you create an analog input object AI for a sound card and add two channels to it.

```
AI = analoginput('winsound')
addchannel(AI, 1:2)
```

You can find out the range of valid sample rates with the propinfo function.

```
propinfo(AI,'SampleRate')
```

You can use the set command to set the sample rate to 8 kHz for each channel.

```
set(AI, 'SampleRate', 8000);
```

To find out what the actual sample rate is set to:

```
ActualRate = get(AI, 'SampleRate');
```

Alternatively, you can use the setverify function. This function sets the sample rate value and then returns the actual value that is set.

```
setverify(AI, 'SampleRate', 8000)
```

See Also

Properties

BufferingConfig

Functions

`propinfo`, `setverify`

SamplesAcquired

`SamplesAcquired` indicates the number of samples acquired per channel.

`SamplesAcquired` is continuously updated to reflect the current number of samples acquired. It is reset to zero only after a `start` command is issued.

Characteristics

Device Objects	AI
Class	Common
Access	Read-only
Data Type	Double
Modify while Running	N/A

Values

The default value is zero.

See Also

Properties

`SamplesAcquiredAction`, `SamplesAvailable`

SamplesAcquiredAction

`SamplesAcquiredAction` specifies the M-file function to be executed every time the number of samples specified by `SamplesAcquiredActionCount` are acquired for each channel group member. The actual frequency with which `SamplesAcquiredAction` is called is based largely on this value and the sample rate.

If data is to be processed as it is being acquired, it must be extracted from the data acquisition engine with the `getdata` function during execution of the

SamplesAcquiredAction M-file. getdata ensures that all data is processed and that no data is missed. Therefore, acquisition performance may be affected.

Characteristics

Device Objects	AI
Class	Common
Access	Read/write
Data Type	String
Modify while Running	Yes

Values

The default value is an empty string.

See Also

Properties

SamplesAcquiredActionCount

SamplesAcquiredActionCount

SamplesAcquiredActionCount specifies the number of samples to acquire for each channel group member before the M-file function specified by SamplesAcquiredAction is executed.

Characteristics

Device Objects	AI
Class	Common
Access	Read/write
Data Type	Double
Modify while Running	No

Values

The default value is 1024.

See Also

Properties

SamplesAcquired, SamplesPerTrigger

SamplesAvailable

SamplesAvailable indicates the number of samples available per channel in the data acquisition engine.

For analog input objects, SamplesAvailable indicates the number of samples that can be removed from the engine for each channel group member with getdata. For analog output objects, SamplesAvailable indicates the number of samples that can be sent to each channel group member with putdata.

After a getdata or putdata function call is issued, SamplesAvailable is reduced by the number of samples extracted from the data acquisition engine.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read-only
Data Type	Double
Modify while Running	N/A

Values

The default value is zero.

Example

It is sometime useful to poll SamplesAvailable.

```
SamplesAvailable = get(AI, 'SamplesAvailable')
if (SamplesAvailable > 100)
getdata(AI, SamplesAvailable)
end
```

See Also

Properties

SamplesAcquired, SamplesOutput

SamplesOutput

`SamplesOutput` indicates the number of samples that have been sent to the analog output hardware device for each channel group member.

Characteristics

Device Objects	AO
Class	Common
Access	Read-only
Data Type	Double
Modify while Running	N/A

Values

The default value is zero.

See Also

Properties

`SamplesAvailable`

SamplesOutputAction

`SamplesOutputAction` specifies the M-file function to be executed every time the number of samples specified by `SamplesOutputActionCount` are sent.

Characteristics

Device Objects	AO
Class	Common
Access	Read/write
Data Type	String
Modify while Running	Yes

Values

The default value is an empty string.

See Also **Properties**

`SamplesOutputActionCount`

SamplesOutputActionCount

`SamplesOutputActionCount` specifies the number of samples to output for each channel group member before the M-file function specified by `SamplesOutputAction` is executed.

Characteristics

Device Objects	AO
Class	Common
Access	Read/write
Data Type	Double
Modify while Running	No

Values The default value is 1024.

See Also **Properties**

`SamplesOutputAction`

SamplesPerTrigger

`SamplesPerTrigger` specifies the number of samples to acquire for each channel group member for each analog input trigger that occurs.

If `SamplesPerTrigger` is set to `Inf`, then the analog input object continually samples 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

Device Objects	AI
Class	Common
Access	Read/write
Data Type	Double
Modify while Running	No

Values

The default value is set by the engine such that one second of data is acquired.

See Also

Properties

`SampleRate`

Sending

`Sending` indicates data is being output to the D/A hardware device.

`Sending` is automatically set to `On` when a trigger occurs. `Sending` is set of `Off` when the requested samples are sent, an error occurs, or a `stop` command is issued.

Characteristics

Device Objects	AO
Class	Common
Access	Read-only
Data Type	Enumerated list
Modify while Running	N/A

Values

`{Off}`	Data is not being sent to the D/A hardware device.
`On`	Data is being sent to the D/A hardware device.

SensorRange

SensorRange specifies the range of data you expect from your sensor. If SensorRange is greater than InputRange, a warning is issued.

Characteristics

Device Objects	AI
Class	Channel
Access	Read/write
Data Type	Vector of doubles
Modify while Running	Yes

Values

The default value is defined when channels are added to the device object.

See Also

Properties

InputRange, UnitsRange

StartAction

StartAction specifies the M-file function to be executed just before the hardware device and data acquisition engine start. That is, StartAction is called just after the start function is issued. The hardware device will not start until the StartAction function has finished executing.

When the function specified by StartAction has finished executing, Running is set to On.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read/write

Data Type	String
Modify while Running	Yes

Values The default value is an empty string.

See Also **Properties**
`StopAction`

StopAction

`StopAction` specifies the M-file function to be executed just after the hardware device and data acquisition engine stop. A stop condition occurs when the requested data is acquired (AI) or sent (AO), a run-time error occurs, or the `stop` command is issued.

When the function specified by `StopAction` has finished executing, `Running` is set to `Off`.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read/write
Data Type	String
Modify while Running	Yes

Values The default value is an empty string.

See Also **Properties**
`StartAction`

Tag

Tag provides a means to identify device objects with a label. Using the daqfind function and the Tag value, a device object that was cleared from the MATLAB workspace can be identified and retrieved.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read/write
Data Type	String
Modify while Running	Yes

Values

The default value is an empty string.

Example

For example, suppose you want to label the analog input object AI.

```
set(AI, 'Tag', 'Sound')
```

If AI is cleared from the workspace, then daqfind and the Tag value can be used to identify and retrieve the object.

```
clear AI
AIcell = daqfind('Tag', 'Sound')
AI = AIcell{1}
```

See Also

Functions

daqfind

Timeout

Timeout is an additional time to wait (in seconds) for getdata or putdata to return. The Timeout value is added to the time required by the engine to fill or send one data block. If the data block is not filled or sent after waiting the required time, then a timeout occurs. Control is returned to MATLAB and you

will be able to extract or send the remaining data from the engine. Possible timeout conditions include:

- You are triggering on a voltage level and that level never occurs
- You have an externally clocked acquisition and the external clock signal never occurs
- The toolbox loses its hardware connection

A timeout is one of the conditions for stopping an acquisition. When a timeout occurs, the function specified by `RuntimeErrorAction` is called.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read/write
Data Type	Double
Modify while Running	No

Values

The default value is 1 second.

See Also

Properties

`RunTimeErrorAction`

TimerAction

`TimerAction` specifies the M-file function to be executed whenever the time specified by `TimerPeriod` has passed. This time-based call is typically used to display data. Only one `TimerAction` M-file function can be queued at any one time. Therefore, data can be missed.

Time is measured relative to when the hardware device starts running.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read/write
Data Type	String
Modify while Running	Yes

Values

The default value is an empty string.

See Also

Properties

`TimerPeriod`

TimerPeriod

`TimerPeriod` specifies the time, in seconds, that M-file function specified for `TimerAction` is called.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read/write
Data Type	Double
Modify while Running	No

Values

The default value is 0.1 seconds.

See Also

Properties

`TimerAction`

TriggerAction

`TriggerAction` specifies the M-file function to be executed when a trigger occurs. `TriggerAction` is called just after `Logging` is set to `On`.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read/write
Data Type	String
Modify while Running	Yes

Values

The default value is an empty string.

TriggerChannel

`TriggerChannel` is a vector containing the analog input channels serving as trigger sources. For all supported hardware, `TriggerChannel` is used only when `TriggerType` is `Software`. For National Instruments hardware, if `TriggerType` is `HwAnalogChannel`, `TriggerChannel` must be the first element of the channel group.

Characteristics

Device Objects	AI
Class	Common
Access	Read/write
Data Type	Vector of channels
Modify while Running	No

Values

The default value is an empty vector.

See Also

Properties

TriggerCondition, TriggerConditionValue, TriggerType

TriggerCondition

TriggerCondition specifies the condition which must be satisfied before a trigger is issued. 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 a value to be specified for TriggerConditionValue. Depending on the hardware you are using, additional trigger conditions may be available.

Characteristics

Device Objects	AI
Class	Common
Access	Read/write
Data Type	Enumerated list
Modify while Running	No

Values

The trigger condition given below can be used when TriggerType is Immediate or Manual or HwDigital (National Instruments only).

{None}	No trigger condition is required.

The trigger conditions given below can be used 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.

`Leaving`	The trigger occurs when the signal leaves the specified range of values.
`Entering`	The trigger occurs when the signal enters the specified range of values.

National Instruments Values

The trigger condition given below can be used when `TriggerType` is `HwDigital`.

`{None}`	No trigger condition is required.

The trigger conditions given below can be used when `TriggerType` is `HwAnalogChannel` or `HwAnalogPin`.

`{AboveHighLevel}`	The trigger occurs when the signal is above the specified value.
`BelowLowLevel`	The trigger occurs when the signal is below the specified value.
`InsideRegion`	The trigger occurs when the signal is inside the specified region.
`LowHysteresis`	The trigger occurs when the signal is less than the specified low value with hysteresis given by the specified high value.
`HighHysteresis`	The trigger occurs when the signal is greater than the specified high value with hysteresis given by the specified low value.

See Also

Properties

`TriggerChannel`, `TriggerConditionValue`, `TriggerType`

TriggerConditionValue

`TriggerConditionValue` specifies the value of a trigger condition. The trigger condition is given by `TriggerCondition`.

`TriggerConditionValue` is ignored when `TriggerCondition` is `None`.

Characteristics

Device Objects	AI
Class	Common
Access	Read/write
Data Type	Row vector of doubles
Modify while Running	No

Values

The default value is zero.

See Also

Properties

`TriggerCondition`, `TriggerType`

TriggerDelay

`TriggerDelay` specifies the delay value for data logging.

Both pretriggers and postriggers can be defined. Pretriggers are specified with a negative `TriggerDelay` value and postriggers are specified with a positive `TriggerDelay` value. Triggers can be delayed in units of time or samples with the `TriggerDelayUnits` property. Pretriggers are not defined when `TriggerType` is `Immediate` or `HWDigital` (National Instruments only).

Pretrigger samples are included as part of the total samples acquired per trigger as specified by the `SamplesPerTrigger` property. For example, if `SamplesPerTrigger` is set to 1000, `TriggerDelay` = -100, and TriggerDelayUnits is set to `Samples`, then 100 samples are logged before the trigger event and 900 samples are logged after the trigger event.

If sample-time pairs are returned to the workspace with the `getdata` function, then pretriggered samples are identified with negative time values.

Characteristics

Device Objects	AI
Class	Common
Access	Read/write
Data Type	Double
Modify while Running	No

Values

The default value is zero.

See Also

Properties

`SamplesPerTrigger`, `TriggerDelayUnits`

TriggerDelayUnits

`TriggerDelayUnits` specifies the trigger delay as either clocked samples or seconds.

If `TriggerDelayUnits` is specified as seconds, then data logging is delayed by the specified time for each channel contained by the device object. If `TriggerDelayUnits` is specified as samples, then data logging is delayed by the specified number of samples for each channel contained by the device object.

Characteristics

Device Objects	AI
Class	Common
Access	Read/write
Data Type	Enumerated list
Modify while Running	No

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`

TriggerRepeat

`TriggerRepeat` specifies the number of additional times the trigger executes.

Triggers can be configured to occur once (one-shot acquisition) or multiple times. If `TriggerRepeat` is set to its default value of 0, then the trigger executes only once when the trigger condition is met. If `TriggerRepeat` is set to a positive integer value, then the trigger repeats the specified number of times when the trigger condition is met. If `TriggerRepeat` is set to `inf`, then the trigger repeats every time the trigger condition is met until an explicit `stop` function is issued or an error occurs.

You can quickly evaluate how many triggers have been issued 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

Device Objects	AI
Class	Common
Access	Read/write
Data Type	Double
Modify while Running	No

Values

The default value is 0.

See Also **Properties**

`TriggerCondition`, `TriggerConditionValue`

TriggerTime

`TriggerTime` is an absolute time vector that indicates the observed times when triggers occurred. The size of the `TriggerTime` vector grows with each trigger. Therefore, you will know how many triggers were issued by examining the size of the `TriggerTime` vector.

If `TriggerCondition` is set to `Manual`, then `TriggerTime` records the time that the `trigger` function is issued. Otherwise, `TriggerTime` records the time when `Logging` or `Sending` is set to `On`.

`TriggerTime` values are stored in the `datenum` format. This format can be converted to a string with the `datestr` function.

`TriggerTime` can store a maximum of 1024 trigger times.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read-only
Data Type	Column vector of doubles
Modify while Running	N/A

Values The default value is an empty vector.

See Also **Properties**

`TriggerCondition`

TriggerType

`TriggerType` specifies the type of trigger to be issued. All triggers have the action of setting `Logging` (AI) or `Sending` (AO) to `On`. Some trigger types require trigger conditions. Trigger conditions are specified with the `TriggerCondition` property.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read/write
Data Type	Enumerated list
Modify while Running	Yes

Values

`{Immediate}`	The trigger occurs immediately after `start` is issued. `Logging` is set to `On` once the `start` function is issued. Pretrigger data cannot be captured.
`Manual`	The trigger occurs immediately after the `trigger` function is issued. `Logging` is set to `On` once the `trigger` function is issued.
`Software`	The trigger occurs when the associated trigger condition is satisfied. Trigger conditions are given by the `TriggerCondition` property. (AI only).

National Instruments Values

`HwDigital`	The trigger source is an external TTL signal (AI and AO). Pretrigger data cannot be captured.
`HwAnalogChannel`	The trigger source is an external analog signal (AI only).
`HwAnalogPin`	The trigger source is an external analog signal (AI only).

See Also **Properties**

`Logging`, `TriggerCondition`

Type

`Type` indicates the device object type. The device object types supported by the Data Acquisition Toolbox are analog input and analog output. Once a device object is created, the value of `Type` is automatically defined.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read-only
Data Type	String
Modify while Running	N/A

Values

The value can be `Analog Input` or `Analog Output` and is defined after the device object is created.

Units

`Units` is a string that specifies the engineering units.

Characteristics

Device Objects	AI, AO
Class	Channel
Access	Read/write
Data Type	String
Modify while Running	No

Values The default value is `Volts`.

See Also **Properties**
`UnitsRange`

UnitsRange

`UnitsRange` specifies the engineering units range of the data stored in MATLAB.

Characteristics

Device Objects	AI, AO
Class	Channel
Access	Read/write
Data Type	Vector of doubles
Modify while Running	Yes

Values The default value is defined when channels are added to the device object.

See Also **Properties**
`InputRange`, `OutputRange`, `SensorRange`, `Units`

UserData

`UserData` stores data that you want to associate with the device object.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read/write

Data Type	Any type
Modify while Running	Yes

Values

The default value is an empty vector.

Example

Suppose you create an analog input object `ai` and want to access filter coefficients during an acquisition. A structure can be created to store these coefficients, which can then be stored in `UserData`.

```
coeff.a = 1.0
coeff.b = -1.25
ai.UserData = coeff;
```

To return the second coefficient

```
x = ai.UserData;
x.b
ans =
   -1.2500
```

A

Device-Specific Properties

Overview

This chapter presents the device-specific properties. For detailed information on the device-specific properties listed here, refer to the relevant documents provided by the hardware vendor.

Device-specific enumerated values for common properties are listed in Chapter 7, “Property Reference”.

Sound Card Properties

The sound card properties for analog input and analog output objects are given below. Included are the property name, property description, and whether the property is associated with analog input (AI) or analog output (AO) objects.

Device-specific properties can be displayed by issuing a `set(object)` command, and are listed after the common properties.

Property	Description	Device Objects
`BitsPerSample`	Specifies the number of bits the sound card uses to represent each sample.	AI, AO

BitsPerSample

`BitsPerSample` specifies the number of bits the sound card uses to represent each sample.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read/write
Data Type	Double
Modify while Running	Yes

Values

`8`	Represent data with 8 bits
`{16}`	Represent data with 16 bits

National Instruments Properties

The National Instruments properties for analog input and analog output objects are given below. Included are the property name, property description, and whether the property is associated with analog input (AI) or analog output (AO) objects.

Note: The device-specific property values listed below are not necessarily available for all devices. Refer to your hardware documentation for a complete description of device-specific property values.

Device-specific properties can be displayed by issuing a `set(object)` command, and are listed after the common properties.

Property	Description	Device Objects
`DriveAISenseToGround`	Specifies whether to drive AISENSE to onboard ground or not.	AI
`NumMuxBoards`	Specifies the number of external multiplexer devices connected.	AI
`TransferMode`	Specifies how data is transferred from the data acquisition device to computer memory.	AI, AO

DriveAISenseToGround

`DriveAISenseToGround` specifies whether to drive AISENSE to onboard ground or not.

Characteristics

Device Objects	AI
Class	Common

Access	Read/write
Data Type	Boolean
Modify while Running	Yes

Values

`{Off}`	Do not drive AISENSE to ground
`On`	Drive AISENSE to ground

NumMuxBoards

`NumMuxBoards` specifies the number of external multiplexer devices connected.

Characteristics

Device Objects	AI
Class	Common
Access	Read/write
Data Type	Double
Modify while Running	Yes

Values

`{0},1,2, or 4`	The number of AMUX-64T multiplexer devices connected

TransferMode

`TransferMode` specifies how data is transferred from the data acquisition device to computer memory. This property is automatically set by the driver.

Note: Depending on your resources, controlling data transfer via interrupts can significantly degrade system performance.

Characteristics

Device Objects	AI, AO
Class	Common
Access	Read/write
Data Type	Enumerated list
Modify while Running	Yes

Analog Input Values

`Interrupts`	Transfer data using interrupts
`SingleDMA`	Transfer data using a single DMA channel
`DualDMA`	Transfer data using dual DMA channels

Analog Output Values

`Interrupts`	Transfer data using interrupts
`SingleDMA`	Transfer data using a single DMA channel

B

Testing Your Hardware

Testing Your Sound Card

Before using the Data Acquisition Toolbox, you should make sure your sound card works properly. You can check your sound card by recording data and then playing back the recorded data. Recording data uses the sound card's analog input (A/D) subsystem, while playing back data uses the sound card's analog output (D/A) subsystem. Successful completion of these two tasks indicates your sound card works properly, and you will be able to access all the Data Acquisition Toolbox functionality.

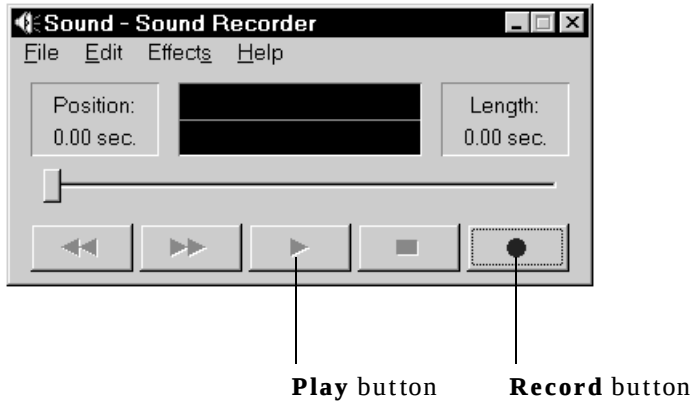
The data to be recorded can come from two sources:

- A microphone
- A CD player

In either case, you will record and play back data using Microsoft's Sound Recorder. To invoke this program:

Start>**Program**>**Accessories**>**Multimedia**>**Sound Recorder**

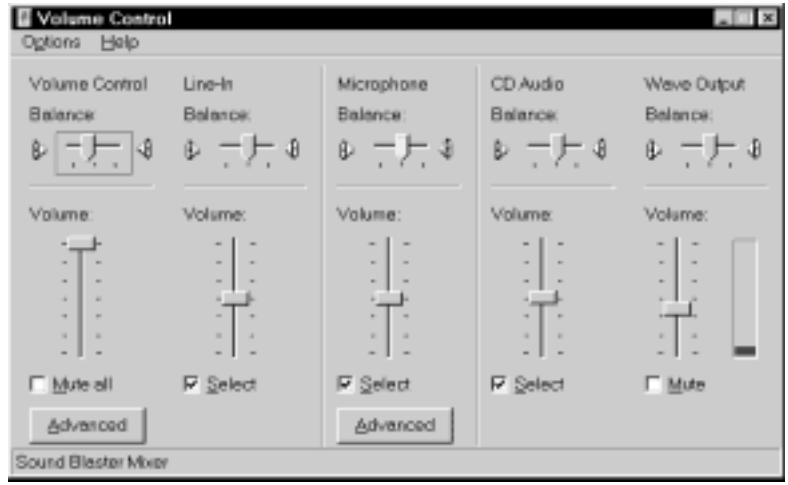
The figure below shows how to record and play back data.



You must also make sure that your microphone or CD player is enabled for recording and playback. To check this, invoke the Volume Control

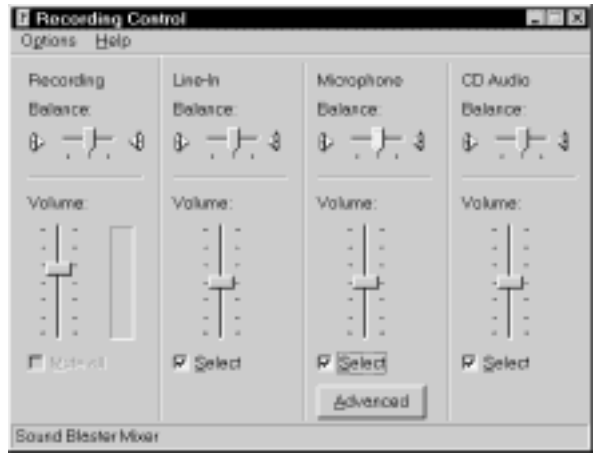
Start>**Program**>**Accessories**>**Multimedia**>**Volume Control**

The Volume Control panel is shown below.



The input devices are enabled for playback by checking the **Select** checkbox, and WAV files can be played back by leaving the **Mute** checkbox unchecked. If the microphone, CD, or Wave Output entries do not appear in the Volume Control panel, you must modify the playback properties by selecting **Options>Properties** from the Volume Control menu. The Properties dialog box is shown below. Select the appropriate device checkbox to enable playback.

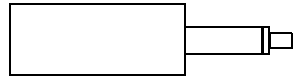
To check if the CD and microphone are enabled for recording, select the Recording radio button in the Properties dialog box, and then select the appropriate device checkbox to enable recording. The Recording Control panel is shown below.



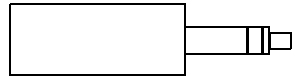
Microphone and Sound Card Types

Your microphone will be one of two possible types: powered or unpowered. Powered microphones can be used only with Sound Blaster or Sound Blaster compatible microphone inputs. Unpowered microphones can be used with any sound card microphone input. Many laptops must use unpowered microphones since they do not have Sound Blaster compatible sound cards.

As shown below, these two microphone types can be easily identified by their jacks.



Unpowered microphone jack



Powered microphone jack

You can find out which sound card brand you have installed through the Control Panel:

Start>Control Panel>Multimedia

and then select the Devices tab and the Audio Devices entry.

Testing with a Microphone

Testing your sound card with a microphone follows 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 Sound Recorder's **Record** button and then speak into the microphone. While recording, the green line in the Sound Recorder should not be flat-lining. If everything has worked, then the A/D subsystem on your sound card is working 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 back, the green line in the Sound Recorder should not be flat-lining. If everything has worked, then the D/A subsystem on your sound card is working properly.

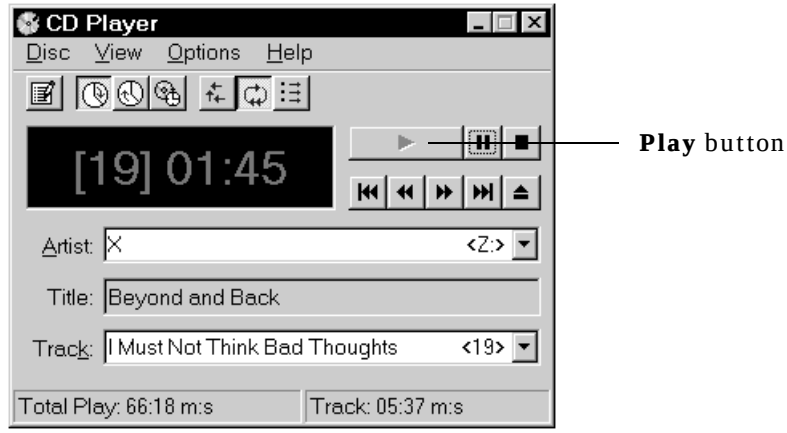
Testing with a CD Player

Testing your sound card with a CD player follows these steps

1. Check that your CD is physically connected to your sound card.
   - Open your computer and find 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 use the microphone to record data.
2. Put an audio CD into your CD player. The Microsoft CD Player should automatically be invoked and begin playing the CD. If this doesn't occur, then you must invoke this program yourself:

Start>**Programs**>**Accessories**>**Multimedia**>**CD Player**

The figure below shows how to play a CD with the CD Player program.



3 While the CD is playing, record audio data by selecting the Sound Recorder's **Record** button. While recording, the green line in the Sound Recorder should not be flat-lining. If everything has worked, then the A/D subsystem on your sound card is working properly. Note that the CD player performs a D/A conversion on the 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 back, the green line in the Sound Recorder should not be flat-lining. If everything has worked, then the D/A subsystem on your sound card is working properly.

Running in Full Duplex Mode

If your sound card supports full duplex mode, then you can simultaneously access your sound card's D/A and A/D converter. You must have the appropriate full duplex driver installed to use this functionality.

Simultaneous A/D and D/A conversions must be performed at the same sample rate.

If you try to run in full duplex mode, and your sound card does not support this mode or you don't have the correct driver installed, then the following error is returned.

```
?? Error using ==> daqdevice/start
Device 'Winsound' already in use.
```

Index

F

G

H