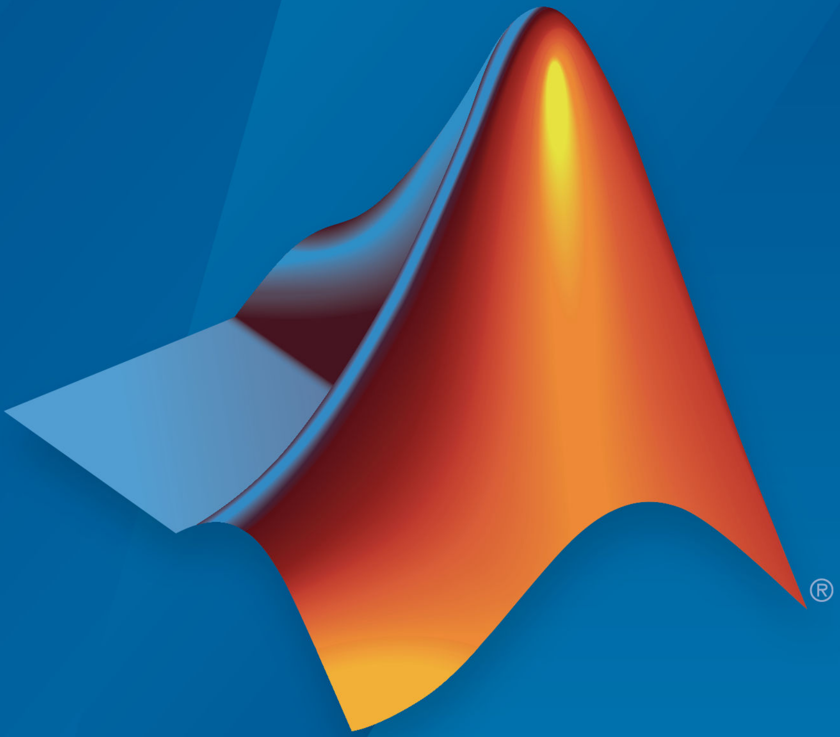
Data Acquisition Toolbox™

User's Guide



MATLAB® & SIMULINK®

R2019**b**

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Data Acquisition Toolbox™ User's Guide

Revision History

May 1999	First printing	New for Version 1
November 2000	Second printing	Revised for Version 2 (Release 12)
June 2001	Third printing	Revised for Version 2.1 (Release 12.1)
July 2002	Online only	Revised for Version 2.2 (Release 13)
June 2004	Online only	Revised for Version 2.5 (Release 14)
October 2004	Online only	Revised for Version 2.5.1 (Release 14SP1)
March 2005	Online only	Revised for Version 2.6 (Release 14SP2)
September 2005	Online only	Revised for Version 2.7 (Release 14SP3)
October 2005	Reprint	Version 2.1 (Notice updated)
November 2005	Online only	Revised for Version 2.8 (Release 14SP3+)
March 2006	Fourth printing	Revised for Version 2.8.1 (Release 2006a)
September 2006	Online only	Revised for Version 2.9 (Release 2006b)
March 2007	Online only	Revised for Version 2.10 (Release 2007a)
May 2007	Fifth printing	Minor revision for Version 2.10
September 2007	Online only	Revised for Version 2.11 (Release 2007b)
March 2008	Online only	Revised for Version 2.12 (Release 2008a)
October 2008	Online only	Revised for Version 2.13 (Release 2008b)
March 2009	Online only	Revised for Version 2.14 (Release 2009a)
September 2009	Online only	Revised for Version 2.15 (Release 2009b)
March 2010	Online only	Revised for Version 2.16 (Release 2010a)
September 2010	Online only	Revised for Version 2.17 (Release 2010b)
April 2011	Online only	Revised for Version 2.18 (Release 2011a)
September 2011	Online only	Revised for Version 3.0 (Release 2011b)
March 2012	Online only	Revised for Version 3.1 (Release 2012a)
September 2012	Online only	Revised for Version 3.2 (Release 2012b)
March 2013	Online only	Revised for Version 3.3 (Release 2013a)
September 2013	Online only	Revised for Version 3.4 (Release 2013b)
March 2014	Online only	Revised for Version 3.5 (Release 2014a)
October 2014	Online Only	Revised Version 3.6 (Release 2014b)
March 2015	Online only	Revised for Version 3.7 (R2015a)
September 2015	Online only	Revised for Version 3.8 (Release 2015b)
March 2016	Online only	Revised for Version 3.9 (Release 2016a)
September 2016	Online only	Revised for Version 3.10 (Release 2016b)
March 2017	Online only	Revised for Version 3.11 (Release 2017a)
September 2017	Online only	Revised for Version 3.12 (Release 2017b)
March 2018	Online only	Revised for Version 3.13 (Release 2018a)
September 2018	Online only	Revised for Version 3.14 (Release 2018b)
March 2019	Online only	Revised for Version 4.0 (Release 2019a)
September 2019	Online only	Revised for Version 4.0.1 (Release 2019b)

Contents

1 Introduction to Data Acquisition

7 Session Based Analog Input and Output

12 Waveform Function Generation

13 Triggers and Clocks

14 Session-Based Synchronization

A Troubleshooting Your Hardware

Introduction to Data Acquisition

Data Acquisition Toolbox Product Description

Connect to data acquisition cards, devices, and modules

Data Acquisition Toolbox provides apps and functions for configuring data acquisition hardware, reading data into MATLAB® and Simulink®, and writing data to DAQ analog and digital output channels. The toolbox supports a variety of DAQ hardware, including USB, PCI, PCI Express®, PXI®, and PXI-Express devices, from National Instruments® and other vendors.

The toolbox apps let you interactively configure and run a data acquisition session. You can then generate equivalent MATLAB code to automate your acquisition in future sessions. Toolbox functions give you the flexibility to control the analog input, analog output, counter/timer, and digital I/O subsystems of a DAQ device. You can access device-specific features and synchronize data acquired from multiple devices.

You can analyze data as you acquire it or save it for post-processing. You can also automate tests and make iterative updates to your test setup based on analysis results.

Product Capabilities

In this section...

Understanding Data Acquisition Toolbox

Data Acquisition Toolbox enables you to:

- Configure external hardware devices.
- Read data into MATLAB for immediate analysis.
- Send out data.

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

- A framework for bringing live, measured data into the MATLAB workspace using PC-compatible, plug-in data acquisition hardware
- Support for analog input (AI), analog output (AO), and digital I/O (DIO) subsystems including simultaneous analog I/O conversions
- Support for these popular hardware vendors/devices:
  - National Instruments CompactDAQ chassis using the session-based interface
  - National Instruments boards that use NI-DAQmx software
  - Microsoft® Windows® sound cards
- Event-driven acquisitions

Supported Hardware

The list of hardware supported by Data Acquisition Toolbox can change in each release, because hardware support is frequently added. The MathWorks website is the best place to check for the most up-to-date listing.

To see the full list of hardware that the toolbox supports, visit the supported hardware page at `https://www.mathworks.com/hardware-support/data-acquistion-software.html`.

Anatomy of a Data Acquisition Experiment

In this section...

System Setup

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

Calibration

After the hardware and software are installed and the sensors are connected, the data acquisition hardware should be *calibrated*. Calibration consists of providing a known input to the system and recording the output. For many data acquisition devices, calibration can be easily accomplished with software provided by the vendor.

Trials

After the hardware is set up and calibrated, you can begin to acquire data. You might think that if you completely understand the characteristics of the signal you are measuring, then you should be able to configure your data acquisition system and acquire the data.

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

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

Data Acquisition System

In this section...

Overview

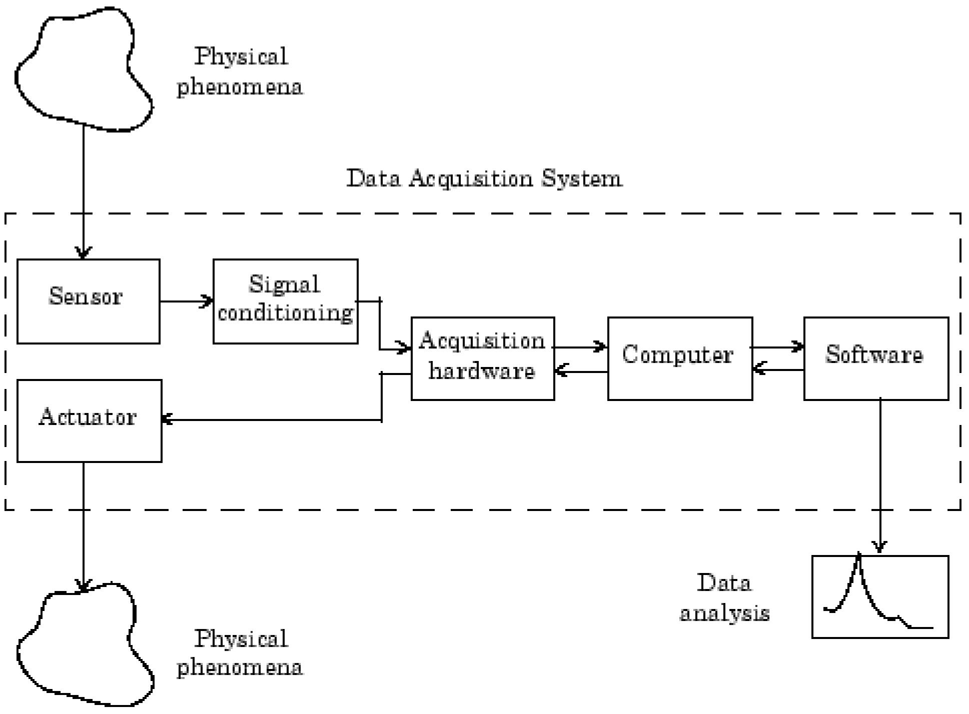
Data Acquisition Toolbox, in conjunction with the MATLAB technical computing environment, gives you the ability to measure and analyze physical phenomena. The purpose of any data acquisition system is to provide you with the tools and resources necessary to do so.

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

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

Components	Description
Signal conditioning hardware	Sensor signals are often incompatible with data acquisition hardware. To overcome this incompatibility, the signal must be conditioned. For example, you might need to condition an input signal by amplifying it or by removing unwanted frequency components. Output signals might need conditioning as well. However, only input signal conditioning is discussed in this topic.
Computer	The computer provides a processor, a system clock, a bus to transfer data, and memory and disk space to store data.
Software	Data acquisition software allows you to exchange information between the computer and the hardware. For example, typical software allows you to configure the sampling rate of your board, and acquire a predefined amount of data.

The following diagram illustrates the data acquisition components, and their relationships to each other.



The figure depicts the two important features of a data acquisition system:

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

  For example, sound level data is acquired from a microphone, amplified, digitized by a sound card, and stored in MATLAB workspace for subsequent analysis of frequency content.
- Data from a computer is converted into an analog signal and output to an actuator.

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

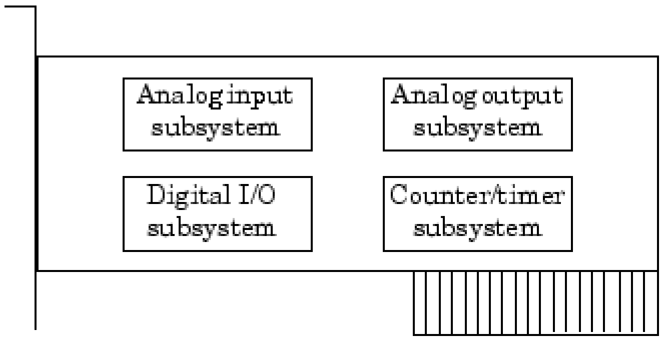
Data Acquisition Hardware

Data acquisition hardware is either internal and installed directly into an expansion slot inside your computer, or external and connected to your computer through an external cable, which is typically a USB cable.

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

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

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



Analog Input Subsystems

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

Analog input subsystems are also referred to as AI subsystems, A/D converters, or ADCs. Analog input subsystems are discussed in detail here.

Analog Output Subsystems

Analog output subsystems convert digital data stored on your computer to a real-world analog signal. These subsystems perform the inverse conversion of analog input subsystems. Typical acquisition boards offer two output channels with 12 bits of resolution, with special hardware available to support multiple channel analog output operations.

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

Digital Input/Output Subsystems

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

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

Counter/Timer Subsystems

Counter/timer (C/T) subsystems are used for event counting, frequency and period measurement, and pulse train generation. Use the session-based interface to work with the counter/timer subsystems.

Sensors

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

Digital sensors produce an output signal that is a digital representation of the input signal, and has discrete values of magnitude measured at discrete times. A digital sensor must output logic levels that are compatible with the digital receiver. Some standard logic levels include transistor-transistor logic (TTL) and emitter-coupled logic (ECL). Examples of digital sensors include switches and position encoders.

Analog sensors produce an output signal that is directly proportional to the input signal, and is continuous in both magnitude and in time. Most physical variables such as temperature, pressure, and acceleration are continuous in nature and are readily measured with an analog sensor. For example, the temperature of an automobile cooling system and the acceleration produced by a child on a swing all vary continuously.

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

Common Analog Sensors

Sensor	Physical Variable
Accelerometer	Acceleration
Microphone	Pressure
Pressure gauge	Pressure
Resistive temperature device (RTD)	Temperature
Strain gauge	Force
Thermocouple	Temperature

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

- The sensor output
- The sensor bandwidth

Note You can use thermocouples and accelerometers without performing linear conversions.

Sensor Output

The output from a sensor can be an analog signal or a digital signal, and the output variable is usually a voltage although some sensors output current.

Current Signals

Current is often used to transmit signals in noisy environments because it is much less affected by environmental noise. The full scale range of the current signal is often either 4-20 mA or 0-20 mA. A 4-20 mA signal has the advantage that even at minimum signal value, there should be a detectable current flowing. The absence of this indicates a wiring problem.

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

Voltage Signals

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

- **Amplitude**

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

The amplitude is related to the sensitivity (resolution) of your hardware. Refer to Accuracy and Precision on page 1-30 for more information about hardware sensitivity.

- **Frequency**

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

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

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

  The frequency is related to the bandwidth of your measurement. Bandwidth is discussed in “Sensor Bandwidth” on page 1-12.

- **Duration**

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

Sensor Bandwidth

In a real-world data acquisition experiment, the physical phenomena you are measuring has expected limits. For example, the temperature of your automobile's cooling system varies continuously between its low limit and high limit. The temperature limits, as well as how rapidly the temperature varies between the limits, depends on several factors including your driving habits, the weather, and the condition of the cooling system. The expected limits might be readily approximated, but there are an infinite number of possible temperatures that you can measure at a given time. As explained in Quantization on page 1-21, these unlimited possibilities are mapped to a finite set of values by your data acquisition hardware.

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

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

Signal Conditioning

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

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

Amplification

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

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

Filtering

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

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

Electrical Isolation

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

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

Multiplexing

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

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

You must take care when using multiplexers so that the switched signal has sufficient time to settle. Refer to Noise on page 1-34 for more information about settling time.

Excitation Source

Some sensors require an excitation source to operate. For example, strain gauges, and resistive temperature devices (RTDs) require external voltage or current excitation.

Signal conditioning modules for these sensors usually provide the necessary excitation. RTD measurements are usually made with a current source that converts the variation in resistance to a measurable voltage.

The Computer

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

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

Data is transferred from the hardware to system memory via dynamic memory access (DMA) or interrupts. DMA is hardware controlled and therefore extremely fast. Interrupts might be slow because of the latency time between when a board requests interrupt servicing and when the computer responds. The maximum acquisition rate is also determined by the computer's bus architecture. Refer to How Are Acquired Samples Clocked? on page 1-24 for more information about DMA and interrupts.

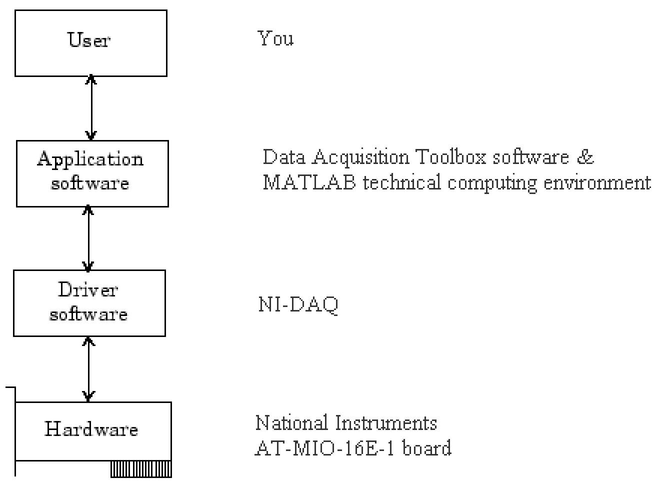
Software

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

There are two kinds of software:

- Driver software
- Application software

For example, suppose you are using Data Acquisition Toolbox software with a National Instruments board and its associated driver. The following diagram shows the relationship between you, the driver software, the application software.


User
You
Application software
Data Acquisition Toolbox software & MATLAB technical computing environment
Driver software
NI-DAQ
Hardware
National Instruments AT-MIO-16E-1 board

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

Driver Software

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

- Transfer data to and from the board
- Control the rate at which data is acquired
- Integrate the data acquisition hardware with computer resources such as processor interrupts, DMA, and memory
- Integrate the data acquisition hardware with signal conditioning hardware
- Access multiple subsystems on a given data acquisition board
- Access multiple data acquisition boards

Application Software

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

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

MATLAB and Data Acquisition Toolbox software provide you with these capabilities, and provide tools that let you perform analysis on the data.

Analog Input Subsystem

In this section...

Function of the Analog Input Subsystem

Note You cannot use the legacy interface on 64-bit MATLAB. See "Session-Based Interface Workflow" on page 5-2 to acquire and generate data on a 64-bit MATLAB.

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

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

Analog signals are continuous in time and in amplitude (within predefined limits). Sampling takes a "snapshot" of the signal at discrete times, while quantization divides the voltage (or current) value into discrete amplitudes. Sampling on page 1-19, quantization on page 1-21, channel configuration on page 1-25, and transferring data on page 1-27 from hardware to system memory are discussed next.

Sampling

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

For most digital converters, sampling is performed by a sample and hold (S/H) circuit. An S/H circuit usually consists of a signal buffer followed by an electronic switch connected to a capacitor. The operation of an S/H circuit follows these steps:

1. At a given sampling instant, the switch connects the buffer and capacitor to an input.
2. The capacitor is charged to the input voltage.
3. The charge is held until the A/D converter digitizes the signal.
4. For multiple channels connected (multiplexed) to one A/D converter, the previous steps are repeated for each input channel.
5. The entire process is repeated for the next sampling instant.

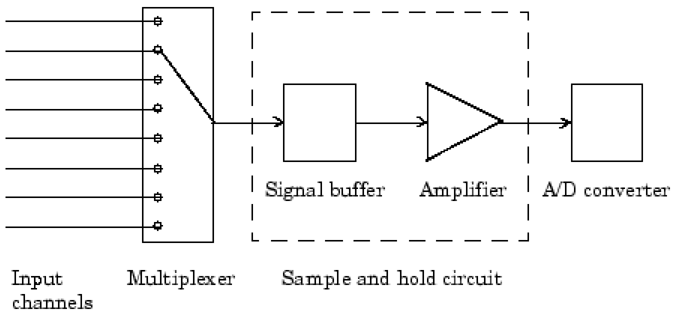
A multiplexer, S/H circuit, and A/D converter are illustrated in the next section.

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

Scanning Hardware

Scanning hardware samples a single input signal, converts that signal to a digital value, and then repeats the process for every input channel used. In other words, each input channel is sampled sequentially. A *scan* occurs when each input in a group is sampled once.

As shown below, most data acquisition devices have one A/D converter that is multiplexed to multiple input channels.

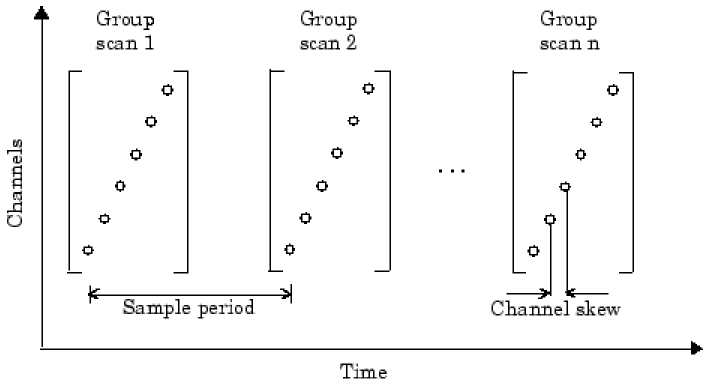


Therefore, if you use multiple channels, those channels cannot be sampled simultaneously and a time gap exists between consecutive sampled channels. This time gap is called the *channel skew*. You can think of the channel skew as the time it takes the analog input subsystem to sample a single channel.

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

$$maximumsamplingrateperchannel = \frac{maximumboardrate}{numberofchannelsscanned}$$

Typically, you can achieve this maximum rate only under ideal conditions. In practice, the sampling rate depends on several characteristics of the analog input subsystem including the settling time and the gain, as well as the channel skew. The sample period and channel skew for a multichannel configuration using scanning hardware is shown below.



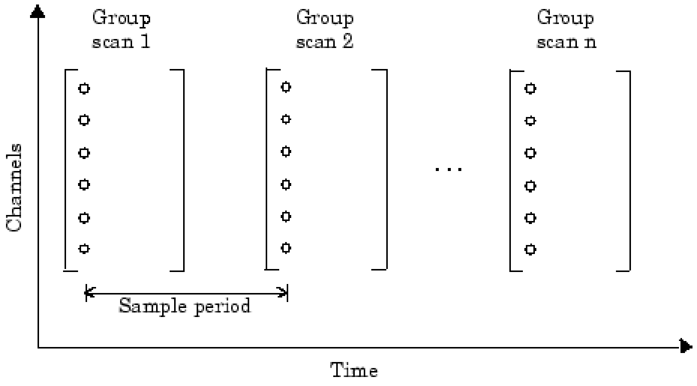
If you cannot tolerate channel skew in your application, you must use hardware that allows simultaneous sampling of all channels. Simultaneous sample and hold hardware is discussed in the next section.

Simultaneous Sample and Hold Hardware

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

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

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

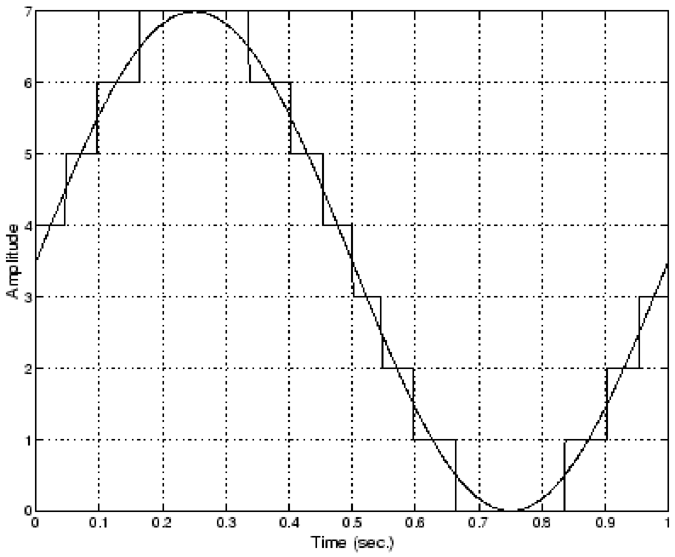


Quantization

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

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

The figure below shows a 1 Hz sine wave quantized by a 3 bit A/D converter.

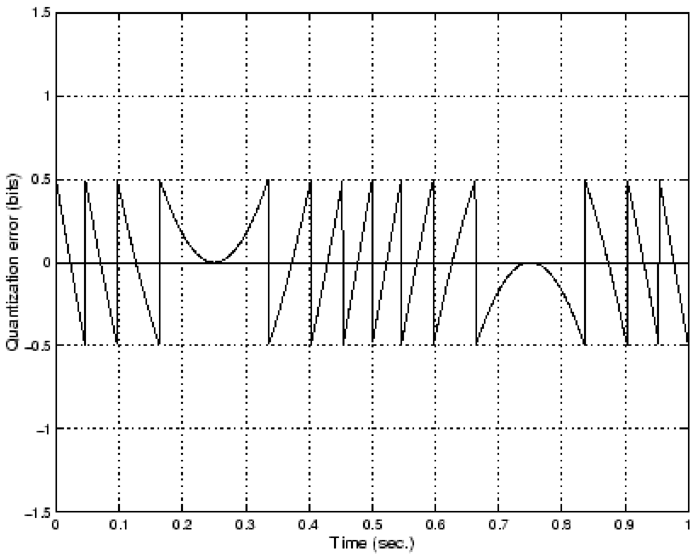


The number of quantized values is given by $2^3 = 8$, the largest representable value is given by $111 = 2^2 + 2^1 + 2^0 = 7.0$, and the smallest representable value is given by $000 = 0.0$.

Quantization Error

There is always some error associated with the quantization of a continuous signal. Ideally, the maximum quantization error is ±0.5 least significant bits (LSBs), and over the full input range, the average quantization error is zero.

As shown below, the quantization error for the previous sine wave is calculated by subtracting the actual signal from the quantized signal.



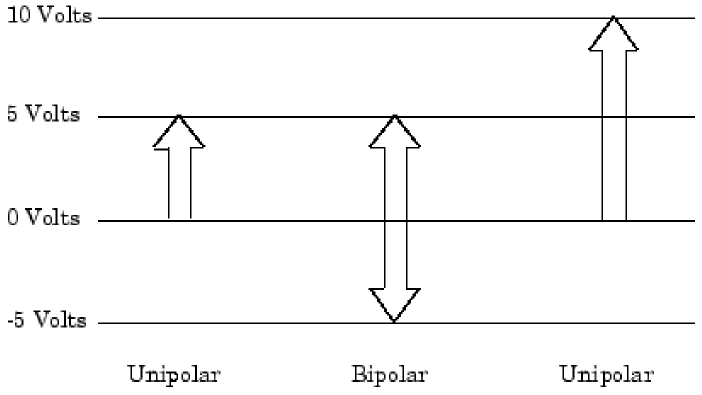
Input Range and Polarity

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

When an input signal exceeds the valid input range of the converter, an *overrange* condition occurs. In this case, most devices saturate to the largest representable value, and the converted data is almost definitely incorrect. The gain setting affects the precision of your measurement — the higher (lower) the gain value, the lower (higher) the precision. Refer to How Are Range, Gain, and Measurement Precision Related? on page 1-33 for more information about how input range, gain, and precision are related to each other.

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

Unipolar and bipolar signals are depicted below. Refer to the figure in “Quantization” on page 1-21 for an example of a unipolar signal.



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

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

How Are Acquired Samples Clocked?

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

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

Onboard Clock

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

Computer Clock

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

External Clock

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

Channel Configuration

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

- Differential
- Single-ended

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

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

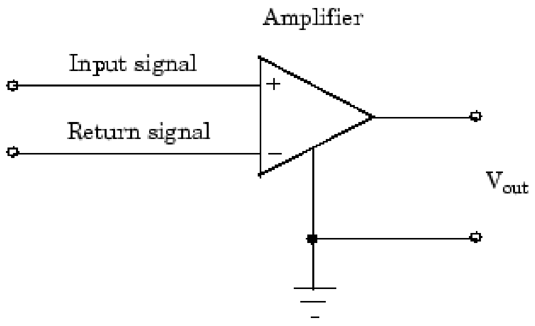
A grounded signal is connected to the building ground. As a result, the input signal and hardware device are connected to a common reference. Examples of grounded signal sources include nonisolated instrument outputs and devices that are connected to the building power system.

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

Differential Inputs

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

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



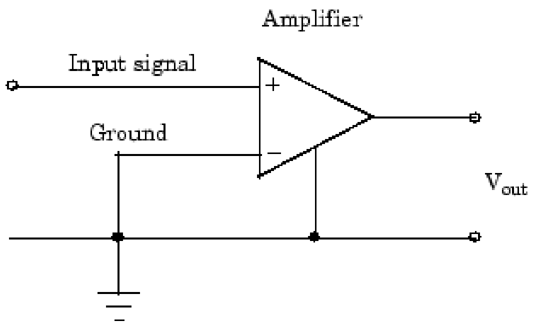
National Instruments recommends that you use differential inputs under any of these conditions:

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

Single-Ended Inputs

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

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



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

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

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

- Referenced single-ended (RSE) connection

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

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

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

Transferring Data from Hardware to System Memory

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

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

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

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

FIFO Buffer

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

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

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

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

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

Interrupts

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

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

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

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

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

DMA

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

1. When data is ready for transfer, the board directs the system DMA controller to put it into in system memory as soon as possible.
2. As soon as the CPU is able (which is usually very quickly), it stops interacting with the data acquisition hardware and the DMA controller moves the data directly into memory.
3. The DMA controller gets ready for the next sample by pointing to the next open memory location.
4. The previous steps are repeated indefinitely, with data going to each open memory location in a continuously circulating buffer. No interaction between the CPU and the board is needed.

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

Making Quality Measurements

In this section...

What Do You Measure?

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

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

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

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

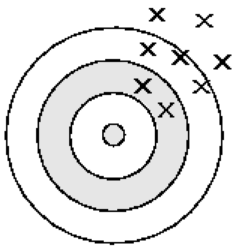
Accuracy and Precision

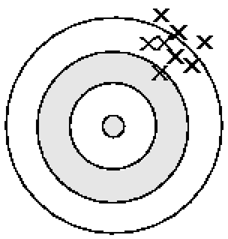
Whenever you acquire measured data, you should make every effort to maximize its accuracy and precision. The quality of your measurement depends on the accuracy and precision of the entire data acquisition system, and can be limited by such factors as board resolution or environmental noise.

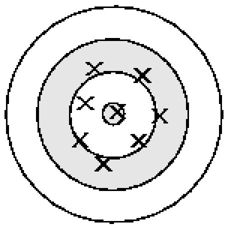
In general terms, the *accuracy* of a measurement determines how close the measurement comes to the true value. Therefore, it indicates the correctness of the result. The *precision* of a measurement reflects how exactly the result is determined without reference to what the result means. The *relative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

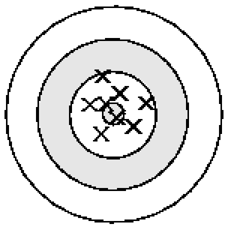
For example, suppose you measure a table top with a meter stick and find its length to be 1.502 meters. This number indicates that the meter stick (and your eyes) can resolve distances down to at least a millimeter. Under most circumstances, this is considered to be a fairly precise measurement with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

Precision and accuracy are illustrated below.


Not precise
Not accurate


Precise
Not accurate


Not precise
Accurate


Precise
Accurate

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

Accuracy

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

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

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

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

Precision

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

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

$$precision = \text{one part in } 2^{numberofbits}$$

The precision in volts is given by:

$$precision = \frac{voltage\ range}{2^{number\ of\ bits}}$$

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

$$precision = \frac{10\ volts}{2^{12}}$$

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

How Are Range, Gain, and Measurement Precision Related?

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

$$actual\ input\ range = \frac{input\ range}{gain}$$

The relationship between gain, actual input range, and precision for a unipolar and bipolar signal having an input range of 10 V is shown below.

Relationship Between Input Range, Gain, and Precision

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

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

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

Noise

Noise is considered to be any measurement that is not part of the phenomena of interest. Noise can be generated within the electrical components of the input amplifier (internal noise), or it can be added to the signal as it travels down the input wires to the amplifier (external noise). Techniques that you can use to reduce the effects of noise are described below.

Removing Internal Noise

Internal noise arises from thermal effects in the amplifier. Amplifiers typically generate a few microvolts of internal noise, which limits the resolution of the signal to this level. The amount of noise added to the signal depends on the bandwidth of the input amplifier.

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

Removing External Noise

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

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

- Configure the input channels in differential mode. Refer to Channel Configuration on page 1-25 for more information about channel configuration.
- Use signal wires that are twisted together rather than separate.

- Keep the signal wires as short as possible.
- Keep the signal wires as far away as possible from environmental electrical activity.

Filtering

Filtering also reduces signal noise. For many data acquisition applications, a low-pass filter is beneficial. As the name suggests, a low-pass filter passes the lower frequency components but attenuates the higher frequency components. The cut-off frequency of the filter must be compatible with the frequencies present in the signal of interest and the sampling rate used for the A/D conversion.

A low-pass filter that's used to prevent higher frequencies from introducing distortion into the digitized signal is known as an antialiasing filter if the cut-off occurs at the Nyquist frequency. That is, the filter removes frequencies greater than one-half the sampling frequency. These filters generally have a sharper cut-off than the normal low-pass filter used to condition a signal. Antialiasing filters are specified according to the sampling rate of the system and there must be one filter per input signal.

Matching the Sensor Range and A/D Converter Range

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

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

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

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

How Fast Should a Signal Be Sampled?

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

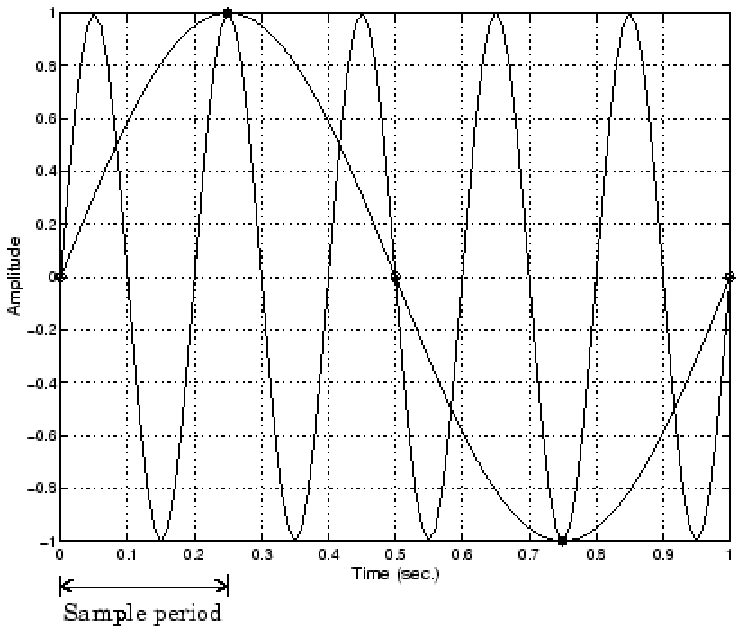
If you sample at a rate that is too slow, then signal aliasing can occur. Aliasing can occur for both rapidly varying signals and slowly varying signals. For example, suppose you are measuring temperature once a minute. If your acquisition system is picking up a 60-Hz hum from an AC power supply, then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

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

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

However, if your input signal is corrupted by noise, then aliasing can still occur.

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



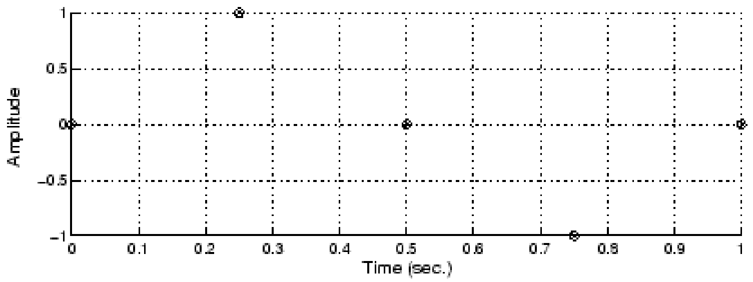
In a real-world data acquisition environment, you might need to condition the signal by filtering out the high frequency components.

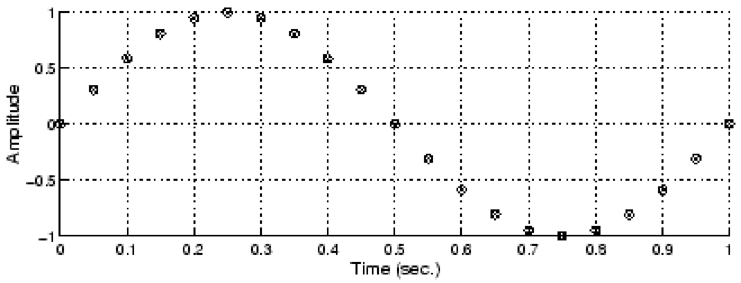
Even though the samples appear to represent a sine wave with a frequency of one-fourth the sampling rate, the actual signal could be any sine wave with a frequency of:

$$(n \pm 0.25) \times (\textit{sampling rate})$$

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

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





As shown in the top figure, the low sampling rate produces a sampled signal that appears to be a triangular waveform. As shown in the bottom figure, a higher fidelity sampled signal is produced when the sampling rate is higher. In the latter case, the sampled signal actually looks like a sine wave.

How Can Aliasing Be Eliminated?

The primary considerations involved in antialiasing are the sampling rate of the A/D converter and the frequencies present in the sampled data. To eliminate aliasing, you must

- Establish the useful bandwidth of the measurement.
- Select a sensor with sufficient bandwidth.
- Select a low-pass antialiasing analog filter that can eliminate all frequencies exceeding this bandwidth.
- Sample the data at a rate at least twice that of the filter's upper cutoff frequency.

Selected Bibliography

[1] *Transducer Interfacing Handbook — A Guide to Analog Signal Conditioning*, edited by Daniel H. Sheingold; Analog Devices Inc., Norwood, MA, 1980.

[2] Bentley, John P., *Principles of Measurement Systems, Second Edition*; Longman Scientific and Technical, Harlow, Essex, UK, 1988.

[3] Bevington, Philip R., *Data Reduction and Error Analysis for the Physical Sciences*; McGraw-Hill, New York, NY, 1969.

[4] *Carr, Joseph J., Sensors*; Prompt Publications, Indianapolis, IN, 1997.

[5] *The Measurement, Instrumentation, and Sensors Handbook*, edited by John G. Webster; CRC Press, Boca Raton, FL, 1999.

[6] *PCI-MIO E Series User Manual, January 1997 Edition*; Part Number 320945B-01, National Instruments, Austin, TX, 1997.

Using Data Acquisition Toolbox Software

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

Installation Information

In this section...

Prerequisites

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

- MATLAB
- Data Acquisition Toolbox
- A supported data acquisition device (see `https://www.mathworks.com/hardware-support/data-acquistion-software.html`)
- Software such as drivers and support libraries, as required by your data acquisition device

Toolbox Installation

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

```
ver
```

at the MATLAB prompt. The MATLAB Command Window lists information about the software versions you are running, including installed add-on products and their version numbers. Check the list to see if Data Acquisition Toolbox product appears. For information about installing the toolbox, see the MATLAB Installation documentation.

If you experience installation difficulties and have Internet access, look for the license manager and installation information at the MathWorks Web site (`https://www.mathworks.com`).

Hardware and Driver Installation

Installation of your hardware device, hardware drivers, and any other device-specific software is described in the documentation provided by your hardware vendor.

Device drivers are available as Support Packages from the Add-Ons menu. See “Install Hardware Support Package for Vendor Support” on page 6-3.

Access Your Hardware

In this section...

Connect to Your Hardware

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

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

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

If you want detailed information about any functions that are used, refer to the list of functions. If you want detailed information about any properties that are used, refer to the list of properties.

Note If you are connecting to a CompactDAQ devices or a counter/timer device, see "Counter and Timer Input and Output".

Acquire Audio Data

If you have a sound card installed, you can run the following example, which acquires 1 second of data an audio input hardware channels, and then plots the acquired data.

You should modify this example to suit your specific application needs.

1. **Create a session object** — Create the session object `s` for a sound card.

   ```
   s = daq.createSession('directsound');
   ```

2. Identify the system devices and their IDs for audio input and output.

   ```
   daq.getDevices

   Data acquisition devices:

   index   Vendor    Device ID                    Description
   ----- ----------- --------- -----------------------------------------------------
   1     directsound Audio0    DirectSound Primary Sound Capture Driver
   2     directsound Audio1    DirectSound HP 4120 Microphone (HP 4120)
   3     directsound Audio2    DirectSound Primary Sound Driver
   4     directsound Audio3    DirectSound Speakers (Realtek High Definition Audio)
   5     directsound Audio4    DirectSound HP 4120 (HP 4120)
   ```

3. **Add channel** — Add an analog input channel to `s`, using the microphone device.

   ```
   addAudioInputChannel(s,'audio1','1','audio');
   ```

   To display a summary of the session, type:

   ```
   s

   Data acquisition session using DirectSound hardware:
      Will run for 1 second (44100 scans) at 44100 scans/second.
      Number of channels: 1
         index Type Device Channel MeasurementType     Range     Name
         ----- ---- ------ ------- --------------- ------------- ----
         1     audi Audio1 1       Audio           -1.0 to +1.0
   ```

4. **Acquire data** — Start the acquisition. When all the data is acquired, it is assigned to `data`.

   ```
   data = startForeground(s);
   plot(data)
   ```

5. **Clean up** — When you no longer need `s`, you should remove it from memory.

   ```
   delete(s)
   clear s
   ```

Generate Audio Data

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

You should modify this example to suit your specific application needs.

1 **Create a session object** — Create the session object `s` for a sound card.

```
s = daq.createSession('directsound');
```

2 **Add channel** — Add an audio output channel to session `s`. This example uses the device ID `audio4` for the speakers from the previous example.

```
addAudioOutputChannel(s,'audio4','1','audio');
```

To display a summary of the session, type:

```
s

Data acquisition session using DirectSound hardware:
   No data queued.  Will run at 44100 scans/second.
   Number of channels: 1
      index Type Device Channel MeasurementType     Range     Name
      ----- ---- ------ ------- --------------- ------------- ----
      1     audo Audio4 1       Audio           -1.0 to +1.0
```

3 **Output data** — Create 1 second of output data, and queue the data for output from the device. You must queue one column of data for each hardware channel.

```
data = sin(linspace(0,2*pi*500,44100)');
queueOutputData(s,data)
```

Start the output. When all the data is output, `s` stops generating.

```
startForeground(s)
```

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

```
delete(s)
clear s
```

Acquire and Generate Digital Data

If you have a supported National Instruments board with at least two digital I/O ports, you can run the following example, which writes and reads digital values.

You should modify this example to suit your specific application needs. Adjust the example if the ports on your device do not support the input/output directions specified here.

1 **Create a session object** — Create the data acquisition session `s` for a National Instruments board with hardware device ID `cDAQ1Mod1`.

```
s = daq.createSession('ni');
```

2 **Add digital input channels** — Add two lines from port 0 to `s`, and configure them for input.

```
addDigitalChannel(s,'cDAQ1Mod1','Port0/Line0:1','InputOnly');
```

3 **Add digital output lines** — Add two lines from port 0 to `s`, and configure them for output.

```
addDigitalChannel(s,'cDAQ1Mod1','Port0/Line2:3','OutputOnly');
```

To display a summary of the session, type:

```
s

Data acquisition session using National Instruments hardware:
   No data queued.  Will run at 1000 scans/second.
   Number of channels: 4
      index Type  Device     Channel   MeasurementType Range Name
      ----- ---- --------- ----------- --------------- ----- ----
      1     dio  cDAQ1Mod1 port0/line0 InputOnly       n/a
      2     dio  cDAQ1Mod1 port0/line1 InputOnly       n/a
      3     dio  cDAQ1Mod1 port0/line2 OutputOnly      n/a
      4     dio  cDAQ1Mod1 port0/line3 OutputOnly      n/a
```

4 **Add clock and trigger** — To synchronize operations, add a clock and trigger connection.

```
addClockConnection(s,'External','cDAQ1/PFI0','ScanClock')
addTriggerConnection(s,'External','cDAQ1/PFI1','StartTrigger')
```

Note Digital line values are usually not transferred at a specific rate. Although some specialized boards support clocked I/O.

5 **Queue output data and start device** — Create an array of output values, and write the values to the digital I/O subsystem. Note that reading and writing digital I/O line values typically does not require that you configure specific property values.

```
queueOutputData(s,round(rand(4000,2)));
gval = startForeground(s);
```

6 **Display input** — To read only the input lines, type:

```
gval
```

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

```
delete(s)
clear s
```

Examine Your Hardware Resources

In this section...

Use the daq.getDevices Function

You can examine the data acquisition hardware resources visible to the toolbox with the `daq.getDevices` function. Hardware resources include installed boards, hardware drivers, and adaptors.

General Toolbox Information

To display general information about the toolbox, enter:

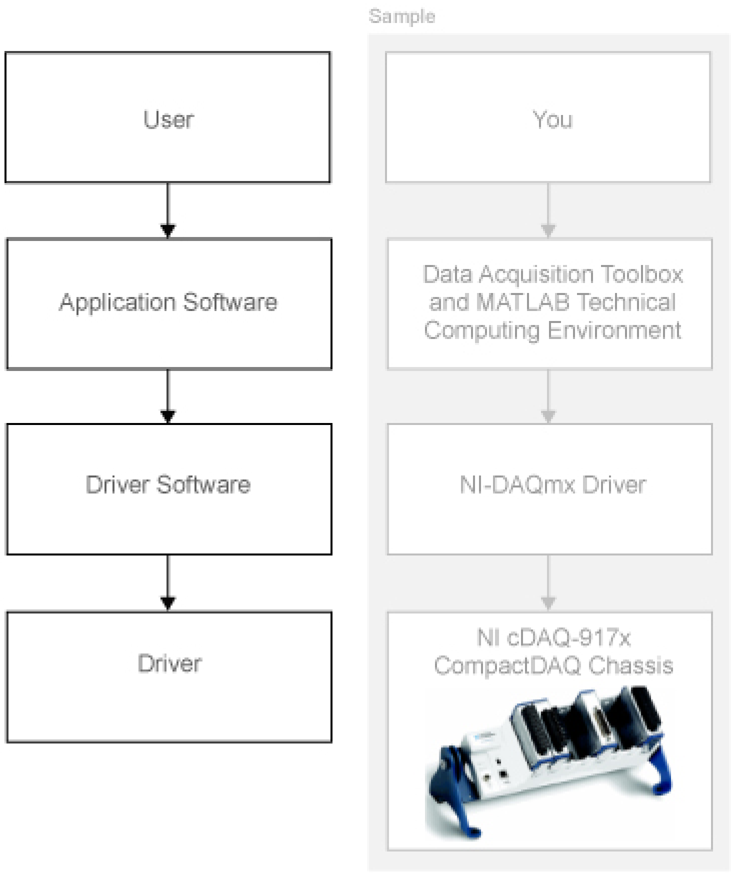
```
daqhelp
```

Introduction to the Session-Based Interface

Data Acquisition Session

The toolbox interface uses a data acquisition session object that allows you to communicate easily with devices from National Instruments, Measurement Computing™, Analog Devices®, Microsoft Windows sound cards, and Digilent®. You create a session using the `daq.createSession` function. A session represents one or more channels that you specify on data acquisition devices. You configure sessions to acquire or generate data at a specific rate, based on the specified number of scans or the duration of the operation.

For an explanation of how this communication works, see Data Acquisition System on page 1-6. The relationship between you, the application software, the driver software, the chassis, and the devices is shown here.


Sample
User
Application Software
Driver Software
Driver
You
Data Acquisition Toolbox and MATLAB Technical Computing Environment
NI-DAQmx Driver
NI cDAQ-917x CompactDAQ Chassis

For more information about creating sessions, see “Create a Session” on page 5-8.

See Also

More About

- “Limitations by Vendor” on page B-2

Getting Help

In this section...

Command-Line Help

To access command-line help for the session-based interface, type:

```
help sessionbasedinterface
```

To access command-line help for a class or method, type:

```
help daq.class_name
help daq.class_name.method_name
```

Online Help

To access online help for the session-based interface via the command line, type:

```
doc daq
```

You can also select **Help** > **Product Help** from the menu bar.

To access online help for a class or method, type:

```
doc daq.class_name
doc daq.class_name.method_name
```

The help browser displays the reference page for the class. You can also select **Help** > **Function Browser** from the menu bar.

Session-Based Interface Examples

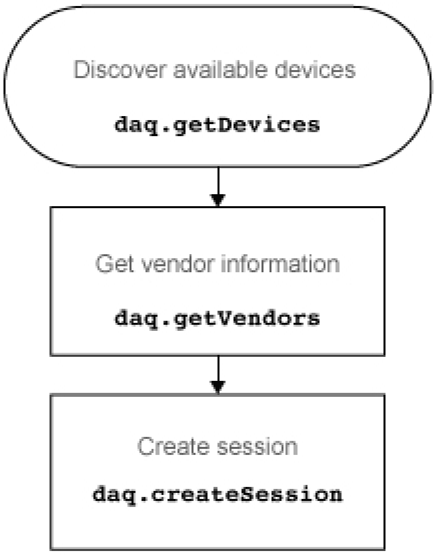
To access the session-based interface examples in the help browser via the command line, type:

```
demo('toolbox','data acquisition')
```

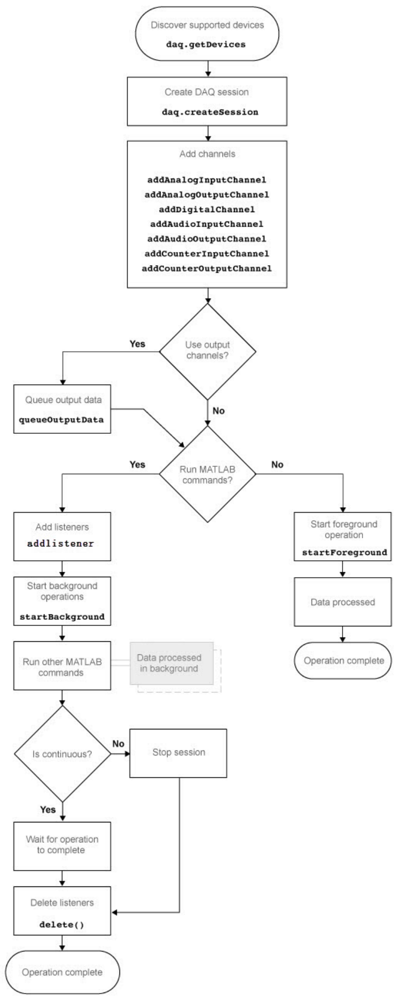
Session-Based Interface Workflows

Session Creation Workflow

This workflow helps you create a data acquisition or generation session.



Once you create a session, you can use this workflow to acquire or generate data.



See Also

Functions

addAnalogInputChannel | addAnalogOutputChannel | addAudioInputChannel | addAudioOutputChannel | addCounterInputChannel | addCounterOutputChannel | addDigitalChannel | addlistener |

`daq.createSession` | `daq.getDevices` | `daq.getVendors` | `queueOutputData` | `startBackground` | `startForeground`

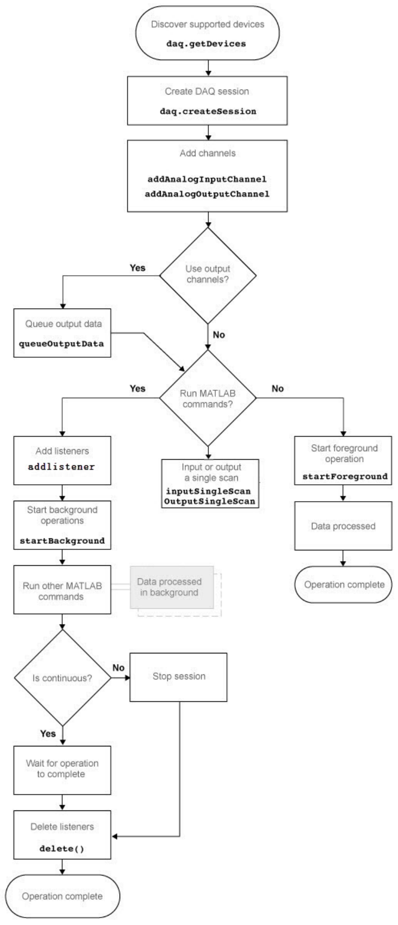
Properties
AutoSyncDSA | DurationInSeconds | EnhancedAliasRejectionEnable | IsContinuous | NumberOfScans | Rate | RateLimit | ScansAcquired | ScansOutputByHardware | ScansQueued

Related Examples

- "Transition Your Code to Session-Based Interface" on page 15-2

Analog Input and Output Workflow

Once you create a session on page 4-2, use this workflow to set up analog channels and acquire and generate data.

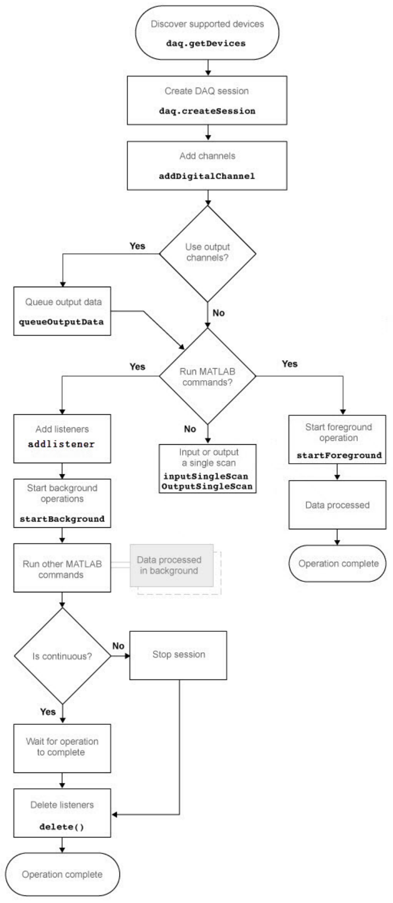


See Also

Functions
`addAnalogInputChannel` | `addAnalogOutputChannel` | `addlistener` | `daq.createSession` | `daq.getDevices` | `inputSingleScan` | `outputSingleScan` | `queueOutputData` | `startBackground` | `startForeground`

Digital Input and Output Workflow

Once you create a session on page 4-2, use this workflow to set up your digital channels and acquire and generate data.

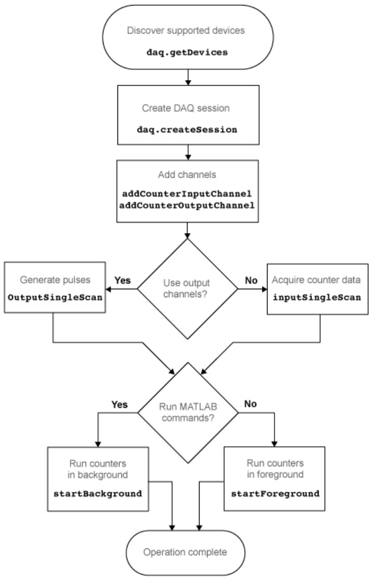


See Also

Functions
`addDigitalChannel` | `addlistener` | `daq.createSession` | `daq.getDevices` | `inputSingleScan` | `outputSingleScan` | `queueOutputData` | `startBackground` | `startForeground`

Counter and Timer Input and Output Workflow

Once you create a session on page 4-2, use this workflow to set up your counter and timer channels and acquire and generate counts.

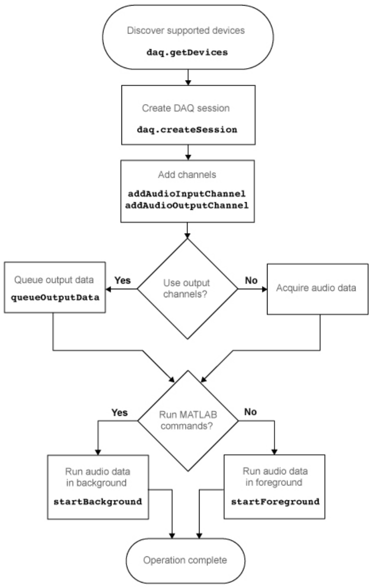


See Also

Functions
`addCounterInputChannel` | `addCounterOutputChannel` | `daq.createSession` | `daq.getDevices` | `inputSingleScan` | `outputSingleScan` | `startBackground` | `startForeground`

Multichannel Audio Input and Output Workflow

Once you create a session on page 4-2, use this workflow to set up your counter and timer channels and acquire and generate multichannel audio.

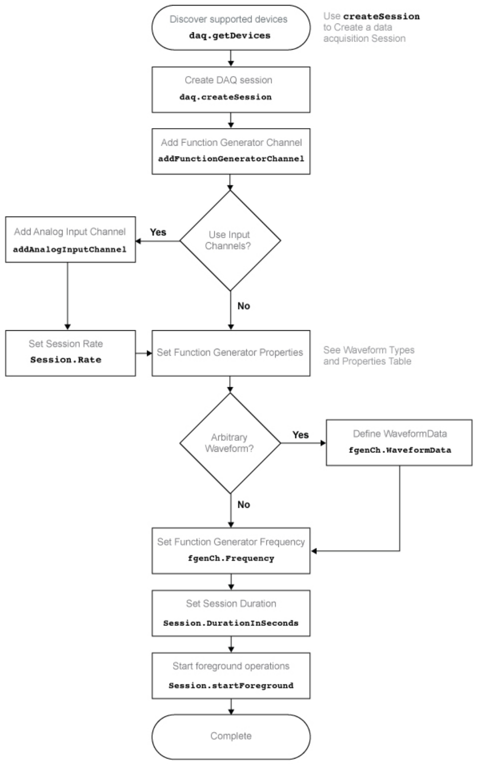


See Also

Functions
`addAudioInputChannel` | `addAudioOutputChannel` | `daq.createSession` | `daq.getDevices` | `queueOutputData` | `startBackground` | `startForeground`

Periodic Waveform Generation Workflow

Once you create a session on page 4-2, use this workflow to create waveform generation channels and acquire waveforms generated on a Digilent Analog Discovery™ device function generation channels.


Discover supported devices
daq.getDevices
Use createSession to Create a data acquisition Session
Create DAQ session
daq.createSession
Add Function Generator Channel
addFunctionGeneratorChannel
Use Input Channels?
Yes
No
Add Analog Input Channel
addAnalogInputChannel
Set Session Rate
Session.Rate
Set Function Generator Properties
See Waveform Types and Properties Table
Arbitrary Waveform?
Yes
No
Define WaveformData
fgenCh.WaveformData
Set Function Generator Frequency
fgenCh.Frequency
Set Session Duration
Session.DurationInSeconds
Start foreground operations
Session.startForeground
Complete

See Also

Functions
`StartForeground` | `addAnalogInputChannel` | `addFunctionGeneratorChannel` | `daq.createSession` | `daq.getDevices`

Properties
DurationInSeconds | Rate

More About

- “Waveform Types” on page 12-6

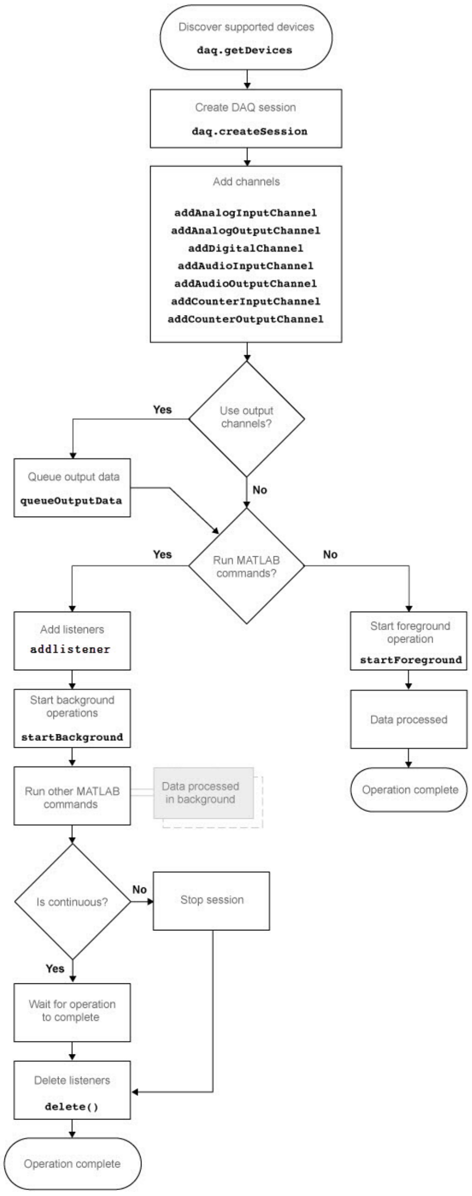
Using the Session-Based Interface

Session-Based Interface Workflow

In this section...

Working with Sessions

Use the session object to communicate with data acquisition devices, such as National Instruments devices including a CompactDAQ chassis. The following diagram shows the general workflow for session operations.



Use the `daq.createSession` function to create a data acquisitions session. See "Session-Based Interface Workflow" on page 5-2 for more information.

You can also synchronize operations within the session. See “Synchronization” on page 14-2 for more information.

Session-Based Interface and Data Acquisition Toolbox

Data Acquisition Toolbox and the MATLAB technical computing environment use the session-based interface to communicate with National Instruments devices, including a CompactDAQ chassis. You can operate in the foreground, where the operation blocks MATLAB until complete, or in the background, where MATLAB continues to run additional MATLAB commands in parallel with the hardware operation. See Session Architecture on page 4-2 for more information.

You can create a session with both analog input and analog output channels and configure acquisition and generation simultaneously. See “Acquire Data and Generate Signals Simultaneously” on page 7-31 for more information.

See Also

Related Examples

- “Transition Your Code to Session-Based Interface” on page 15-2

Digital Input and Output

Digital subsystems transfer digital or logical values in bits via digital lines. You can perform clocked and non-clocked digital operations using the session-based interface in the Data Acquisition Toolbox.

For more information see “Digital Subsystem Channels” on page 10-2.

Discover Hardware Devices

Discover the supported data acquisition devices on your system.

Step 1. Discover hardware devices.

```
d  = daq.getDevices
```

```
d =

Data acquisition devices:

index Vendor Device ID             Description
----- ------ --------- -----------------------------------
1     ni     cDAQ1Mod1 National Instruments NI 9205
2     ni     cDAQ2Mod1 National Instruments NI 9201
3     ni     Dev1      National Instruments USB-6211
4     ni     PXI1Slot2 National Instruments PXI-4461
```

Click the device ID for detailed device information.

Step 2. Get detailed device information.

```
d(3)
```

```
ans =

ni: National Instruments USB-6211 (Device ID: 'Dev1')
   Analog input subsystem supports:
      4 ranges supported
      Rates from 0.1 to 250000.0 scans/sec
      16 channels ('ai0' - 'ai15')
      'Voltage' measurement type

   Analog output subsystem supports:
      -10 to +10 Volts range
      Rates from 0.1 to 250000.0 scans/sec
      2 channels ('ao0','ao1')
      'Voltage' measurement type

   Digital subsystem supports:
      8 channels ('port0/line0' - 'port1/line3')
      'InputOnly','OutputOnly' measurement types

   Counter input subsystem supports:
      Rates from 0.1 to 80000000.0 scans/sec
      2 channels ('ctr0','ctr1')
      'EdgeCount','PulseWidth','Frequency','Position' measurement types

   Counter output subsystem supports:
      Rates from 0.1 to 80000000.0 scans/sec
      2 channels ('ctr0','ctr1')
      'PulseGeneration' measurement type


Properties, Methods, Events
```

Detailed device information includes:

- Subsystem type
- Rate
- Number of available channels
- Measurement type

Create a Session

This example shows how to create a session and add channels to the session and use the session to acquire and generate data. You can also configure session and channel properties needed for your operation.

Step 1. Create a data acquisition session.

```
 s = daq.createSession('ni')
```

```
s =

Data acquisition session using National Instruments hardware:
   Will run for 1 second (1000 scans) at 1000 scans/second.
   No channels have been added.
```

Once you create a session object, add channels using `addAnalogInputChannel`, `addAnalogOutputChannel`, `addCounterInputChannel`, and `addCounterOutputChannel` functions.

Step 2. Configure session properties.

Change the sessions duration to 10 seconds:

```
s.DurationInSeconds = 10
```

```
s =

Data acquisition session using National Instruments hardware:
   Will run for 10 seconds (10000 scans) at 1000 scans/second.
   No channels have been added.
```

Step 3. Add channels to the session.

Add an analog input channel to the session:

```
s.addAnalogInputChannel('cDAQ1Mod1','ai0', 'Voltage')
```

```
ans =

Data acquisition session using National Instruments hardware:
   Will run for 10 seconds (10000 scans) at 1000 scans/second.
   Number of channels: 1
      index Type  Device    Channel MeasurementType      Range       Name
      ----- ---- --------- ------- --------------- ---------------- ----
      1     ai   cDAQ1Mod1 ai0     Voltage (Diff)  -10 to +10 Volts
```

Step 4. Change channel properties.

Examine the channel properties.

```
s.Channels
```

```
ans =

Data acquisition analog input voltage channel 'ai0' on device 'cDAQ1Mod1':

       Coupling: DC
 TerminalConfig: Differential
          Range: -10 to +10 Volts
           Name: empty
             ID: 'ai0'
         Device: [1x1 daq.ni.CompactDAQModule]
MeasurementType: 'Voltage'
```

Change the `TerminalConfig` property to `'SingleEnded'`.

```
s.Channels.TerminalConfig = 'SingleEnded'
```

```
s =

Data acquisition session using National Instruments hardware:
   Will run for 10 seconds (10000 scans) at 1000 scans/second.
   Number of channels: 1
      index Type  Device   Channel   MeasurementType        Range       Name
      ----- ---- --------- ------- ------------------- ---------------- ----
      1     ai   cDAQ1Mod1 ai0     Voltage (SingleEnd) -10 to +10 Volts
```

See Also

Related Examples

- "Acquire Counter Input Data" on page 9-3
- "Generate Pulse Data on a Counter Channel" on page 9-7

More About

- "Analog Input and Output"
- "Transition Your Code to Session-Based Interface" on page 15-2

Support Package Installer

Data Acquisition Toolbox Supported Hardware

Get Support Package Now

As of this release, Data Acquisition Toolbox supports the following hardware.

Support Package	Vendor	Earliest Release Available	Last Release Available
Analog Devices Hardware on page 6-3	Analog Devices	R2015b	Current
Digilent Analog Discovery Hardware on page 6-3	Digilent	R2014a	Current
Measurement Computing Hardware on page 6-3	Measurement Computing	R2017a	Current
National Instruments NI-DAQmx Devices on page 6-3	National Instruments	R2014a	Current
Windows Sound Cards on page 6-3	Microsoft	R2014a	Current

For a complete list of supported hardware, see Hardware Support.

See Also

Related Examples

- “Examples by Vendor” on page B-7

More About

- “Limitations by Vendor” on page B-2

Install Hardware Support Package for Vendor Support

In this section...
"Install Support Packages" on page 6-3
"Update or Uninstall Support Packages" on page 6-3

To communicate with a data acquisition device, you need to install the required support package on your system for the device vendor. Data Acquisition Toolbox support packages are available for the following vendors:

- Analog Devices (ADALM1000)
- Digilent (Analog Discovery)
- Measurement Computing
- Microsoft (Windows Sound cards)
- National Instruments (NI-DAQmx)

Install Support Packages

To install the required support package for a specific vendor and device:

1 On the MATLAB **Home** tab, in the **Environment** section, click **Add-Ons > Get Hardware Support Packages**.
2 In the left pane of the Add-On Explorer, scroll to **Filter by Type** and check **Hardware Support Packages**.
3 Under **Filter by Vendor** check the vendor of your device. The Add-On Explorer displays support packages for that vendor. Click the support package for your vendor and device.
4 Click **Install > Install**. Sign in to your MathWorks® account if necessary, and proceed.

Update or Uninstall Support Packages

To uninstall support packages:

On the MATLAB **Home** tab, in the **Environment** section, click **Add-Ons > Manage Add-Ons**.

To update existing support packages:

On the MATLAB **Home** tab, in the **Environment** section, click **Add-Ons > Check for Updates > Hardware Support Packages**.

See Also

More About

- "Get and Manage Add-Ons" (MATLAB)

Session Based Analog Input and Output

Acquire Data in the Foreground

This example shows how to acquire voltage data from an NI 9205 device with ID cDAQ1Mod1.

Create a session object and save it to the variable, `s`:

```
s = daq.createSession('ni')

s =

Data acquisition session using National Instruments hardware:
   Will run for 1 second (1000 scans) at 1000 scans/second.
   Operation starts immediately.
      No channels have been added.
```

By default, the acquisition is configured to run for a duration of 1 second to acquire 1000 scans, at the rate of 1000 scans per second.

Change the duration of the acquisition to 2 seconds:

```
s.DurationInSeconds = 2.0

s =

Data acquisition session using National Instruments hardware:
   Will run for 2 seconds (2000 scans) at 1000 scans/second.
   No channels have been added.
```

The acquisition now runs for 2 seconds acquiring 2000 scans at the default rate.

Add an analog input 'Voltage' channel named `'ai0'`:

```
addAnalogInputChannel(s,'cDAQ1Mod1','ai0','Voltage')

ans =

Data acquisition session using National Instruments hardware:
   Will run for 1 second (1000 scans) at 1000 scans/second.
   Operation starts immediately.
      Number of channels: 1
      index Type  Device   Channel MeasurementType  Range            Name
      ----- ---- --------- ------- ---------------  ---------------- ----
      1     ai   cDAQ1Mod1 ai0     Voltage (Diff)   -10 to +10 Volts
```

For NI devices, use either a terminal name, like `'ai2'`, or a numeric equivalent like `2` for the channel ID.

Acquire the data and store it in the variable, `data` and plot it:

```
data = startForeground(s);
plot (data)
```

Change the number of scans to 4096.

```
s.NumberOfScans = 4096

s =

Data acquisition session using National Instruments hardware:
   Will run for 4096 scans (4.096 seconds) at 1000 scans/second.
   Operation starts immediately.
      Number of channels: 1
      index Type  Device    Channel MeasurementType  Range            Name
      ----- ---- --------- ------- ---------------  ---------------- ----
      1     ai   cDAQ1Mod1 ai0     Voltage (Diff)   -10 to +10 Volts
```

Changing the number of scans changed the duration of the acquisition to 4.096 seconds at the default rate of 1000 scans per second.

Acquire the data and store it in the variable, `data` and plot it:

```
data = startForeground(s);
plot (data)
```

See Also

Related Examples

- "Acquire Data in the Background" on page 7-6

Acquire Data from Multiple Channels

This example shows how to acquire data from multiple channels, and from multiple devices on the same chassis. In this example, you acquire voltage data from an NI 9201 device with ID cDAQ1Mod4 and an NI 9205 device with ID cDAQ1Mod1.

Create a session object and add two analog input 'Voltage' channels on cDAQ1Mod1 with channel ID 0 and 1:

```
s = daq.createSession('ni');
addAnalogInputChannel(s,'cDAQ1Mod1',0:1,'Voltage');
```

Add an additional channel on a separate device, cDAQ1Mod6 with channel ID 0. For NI devices, use either a terminal name, like `ai0`, or a numeric equivalent like `0`. Store this channel in the variable `ch`.

```
ch = addAnalogInputChannel(s,'cDAQ1Mod6','ai0','Voltage')

ch =

Data acquisition analog input channel 'ai0' on device 'cDAQ1Mod16':

      Coupling: DC
TerminalConfig: Differential
         Range: -10 to +10 Volts
          Name: empty
            ID: 'ai0'
        Device: [1x1 daq.ni.CompactDAQModule]
 ADCTimingMode: ''
```

View the session object to see the three channels:

```
s

s =

Data acquisition session using National Instruments hardware:
   Will run for 1 second (1000 scans) at 1000 scans/second.
   Number of channels: 3
      index Type  Device   Channel   MeasurementType        Range       Name
      ----- ---- --------- ------- ------------------- ---------------- ----
      1     ai   cDAQ1Mod1 ai0     Voltage (SingleEnd) -10 to +10 Volts
      2     ai   cDAQ1Mod1 ai1     Voltage (SingleEnd) -10 to +10 Volts
      3     ai   cDAQ1Mod6 ai0     Voltage (Diff)      -10 to +10 Volts
```

Acquire the data and store it in the variable, `data` and plot it:

```
data = startForeground(s);
plot(data)
```

Change the properties of the channel `'ai0'` on cDAQ1Mod6 and display `ch`:

```
ch.TerminalConfig ='SingleEnded';
ch.Name = 'Velocity sensor';
ch
```

```
ch =

Data acquisition analog input channel 'ai0' on device 'cDAQ1Mod6':

      Coupling: DC
TerminalConfig: SingleEnded
         Range: -10 to +10 Volts
          Name: 'Velocity sensor'
            ID: 'ai0'
        Device: [1x1 daq.ni.CompactDAQModule]
 ADCTimingMode: empty
```

Acquire the data and store it in the variable, `data`, and plot it:

```
data = startForeground(s);
plot(data)
```

See Also

Related Examples

- "Acquire Data in the Foreground" on page 7-2
- "Acquire Data in the Background" on page 7-6

Acquire Data in the Background

This example shows how to acquire data in the background using events and listeners.

A background acquisition depends on events and listeners to allow your code to access data as the hardware acquires it and to react to any errors as they occur. For more information, see Events and Listeners — Concepts in the MATLAB Object-Oriented Programming documentation. Use events to acquire data in the background. In this example, you acquire data from an NI 9205 device with ID cDAQ1Mod1 using a listener and a `DataAvailable` event.

Listeners execute a callback function when notified that the event has occurred. Use `Session.addlistener` to create a listener object that executes your callback function.

Create an NI session object and an analog input 'Voltage' channel on cDAQ1Mod1:

```
s = daq.createSession('ni');
addAnalogInputChannel(s,'cDAQ1Mod1', 'ai0', 'Voltage');
```

Add the listener for the `DataAvailable` event and assign it to the variable `lh`:

```
lh = addlistener(s,'DataAvailable', @plotData);
```

For more information on events, see Events and Listeners — Concepts in the MATLAB Object-Oriented Programming documentation.

Create a simple callback function to plot the acquired data and save it as `plotData.m` in your working directory:

```
 function plotData(src,event)
     plot(event.TimeStamps, event.Data)
 end
```

Here, `src` is the session object for the listener and `event` is a `daq.DataAvailableInfo` object containing the data and associated timing information.

Acquire the data and see the plot update while MATLAB is running:

```
startBackground(s);
```

When the operation is complete, delete the listener:

```
delete (lh)
```

See Also

Related Examples

- “Acquire Data in the Foreground” on page 7-2

Acquire Data from an Accelerometer

This example shows how to acquire and display data from an accelerometer attached to a vehicle driven under uneven road conditions.

Discover Devices that Support Accelerometers

To discover a device that supports Accelerometers, click the name of the device in the list in the Command window, or access the device in the array returned by `daq.getDevices` command. This example uses National Instruments® CompactDAQ Chassis NI cDAQ-9178 and module NI 9234 with ID `cDAQ1Mod3`.

```
devices = daq.getDevices
devices(3)
```

```
devices =

Data acquisition devices:

index Vendor Device ID           Description
----- ------ --------- --------------------------------
1     ni     cDAQ1Mod1 National Instruments NI 9205
2     ni     cDAQ1Mod2 National Instruments NI 9263
3     ni     cDAQ1Mod3 National Instruments NI 9234
4     ni     cDAQ1Mod4 National Instruments NI 9201
5     ni     cDAQ1Mod5 National Instruments NI 9402
6     ni     cDAQ1Mod6 National Instruments NI 9213
7     ni     cDAQ1Mod7 National Instruments NI 9219
8     ni     cDAQ1Mod8 National Instruments NI 9265
9     ni     Dev1      National Instruments PCIe-6363
10    ni     Dev2      National Instruments NI ELVIS II


ans =

ni: National Instruments NI 9234 (Device ID: 'cDAQ1Mod3')
   Analog input subsystem supports:
      -5.0 to +5.0 Volts range
      Rates from 1000.0 to 51200.0 scans/sec
      4 channels ('ai0','ai1','ai2','ai3')
      'Voltage','Accelerometer','Microphone','IEPE' measurement types

This module is in slot 3 of the 'cDAQ-9178' chassis with the name 'cDAQ1'.
```

Add an Accelerometer Channel

Create a session, and add an analog input channel with the `Accelerometer` measurement type.

```
s = daq.createSession('ni');
addAnalogInputChannel(s,'cDAQ1Mod3', 0, 'Accelerometer');
```

Set Session Rate and Duration

Change the scan rate to 4000 scans per second and the duration to 30 seconds.

```
s.Rate = 4000;
s.DurationInSeconds = 30;
s
```

```
s =

Data acquisition session using National Instruments hardware:
   Will run for 30 seconds (120000 scans) at 4000 scans/second.
   Number of channels: 1
      index Type  Device    Channel   MeasurementType          Range        Name
      ----- ---- --------- ------- -------------------- ------------------ ----
      1     ai   cDAQ1Mod3 ai0     Accelerometer (Diff) -5.0 to +5.0 Volts
```

Set Sensitivity

You must set the `Sensitivity` value to the value specified in the accelerometer's data sheet. This example uses a ceramic shear accelerometer model 352C22 from PCB Piezotronics is used with 9.22 mV per Gravity.

```
s.Channels(1).Sensitivity = 0.00922;
s.Channels(1)
```

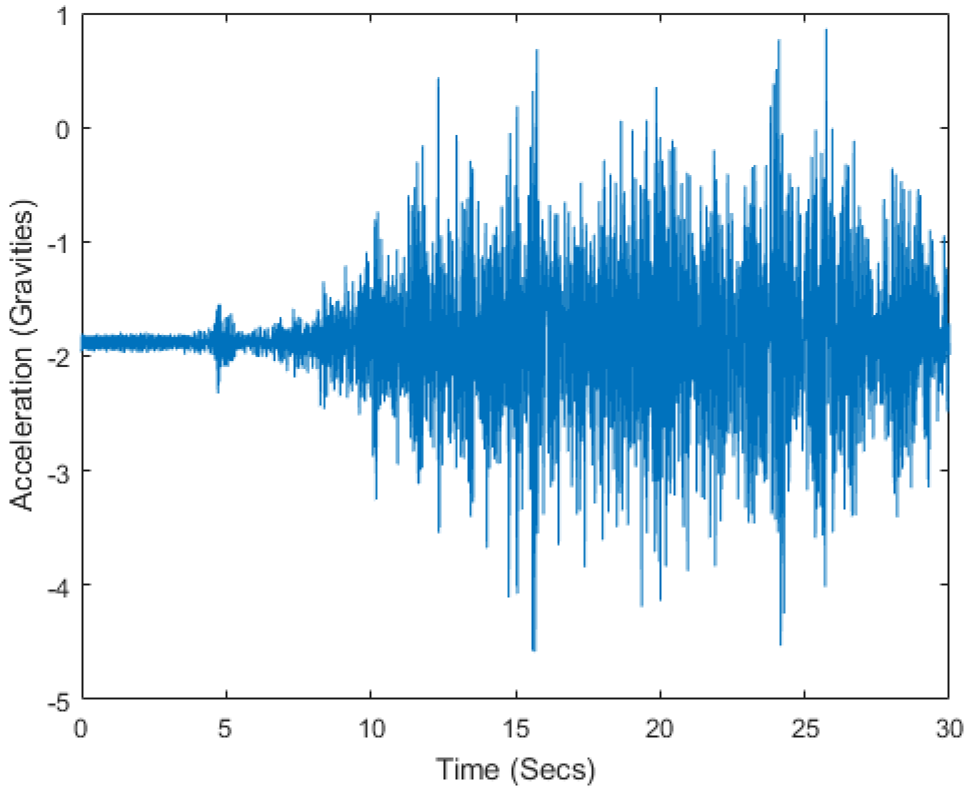
```
ans =

Data acquisition analog input accelerometer channel 'ai0' on device 'cDAQ1Mod3':

      Sensitivity: 0.00922
```

```
ExcitationCurrent: 0.001
 ExcitationSource: None
         Coupling: DC
   TerminalConfig: Differential
            Range: -5.0 to +5.0 Volts
             Name: ''
               ID: 'ai0'
           Device: [1x1 daq.ni.CompactDAQModule]
  MeasurementType: 'Accelerometer'
```

Start Acquisition and Plot the Data

Use `startForeground` to acquire and plot the data.

```
[data,time] = startForeground(s);
plot(time,data)
xlabel('Time (Secs)');
ylabel('Acceleration (Gravities)');
```



See Also

Related Examples

- “Acquire Data in the Foreground” on page 7-2
- “Acquire Data in the Background” on page 7-6

Acquire Bridge Measurements

This example shows how to acquire and plot data from an NI USB-9219 device. The device ID is `'cDAQ1Mod7'`.

Create a session object and save it to the variable `s`:

```
s = daq.createSession('ni');
```

Add an analog input channel with the `'Bridge'` measurement type and save it to the variable `ch`:

```
ch = addAnalogInputChannel(s,'cDAQ1Mod7', 'ai1', 'Bridge');
```

You might see this warning:

```
Warning: The Rate property was reduced to 2 due to the default ADCTimingMode of this device,
which is 'HighResolution'.
To increase rate, change ADCTimingMode on this channel to 'HighSpeed'.
```

To allow a higher acquisition rate, change the channel `ADCTimingMode` to `'HighSpeed'`:

```
ch.ADCTimingMode = 'HighSpeed'
```

You might see this warning:

```
Warning: This property must be the same for all channels on this device.  All channels
associated with this device were updated.
```

Change the acquisition rate to `10` scans per second.

```
s.Rate = 10
```

```
s =

Data acquisition session using National Instruments hardware:
   Will run for 1 second (10 scans) at 10 scans/second.
   Number of channels: 1
      index Type  Device    Channel MeasurementType              Range              Name
      ----- ---- --------- ------- ---------------- ----------------------------- ----
      1     ai   cDAQ1Mod7 ai1     Bridge (Unknown) -0.025 to +0.025 VoltsPerVolt
```

Set `BridgeMode` to `'Full'`, which uses all four resistors in the device to acquire the voltage values:

```
ch.BridgeMode = 'Full'
```

```
ch =
Data acquisition analog input channel 'ai1' on device 'cDAQ1Mod7':

             BridgeMode: Full
       ExcitationSource: Internal
      ExcitationVoltage: 2.5
NominalBridgeResistance: 'Unknown'
                  Range: -0.063 to +0.063 VoltsPerVolt
                   Name: empty
                     ID: 'ai1'
                 Device: [1x1 daq.ni.CompactDAQModule]
        MeasurementType: 'Bridge'
          ADCTimingMode: HighSpeed
```

Set the resistance of the bridge device to 350 ohms:

```
ch.NominalBridgeResistance = 350
```

```
ch =

Data acquisition analog input channel 'ai1' on device 'cDAQ1Mod7':

             BridgeMode: Full
       ExcitationSource: Internal
      ExcitationVoltage: 2.5
NominalBridgeResistance: 350
                  Range: -0.063 to +0.063 VoltsPerVolt
                   Name: empty
                     ID: 'ai1'
                 Device: [1x1 daq.ni.CompactDAQModule]
        MeasurementType: 'Bridge'
          ADCTimingMode: HighSpeed
```

Save the acquired data to a variable and start the acquisition:

```
data = startForeground(s);
```

Plot the acquired data:

```
plot(data)
```

See Also

Related Examples

- “Acquire Data in the Foreground” on page 7-2
- “Acquire Data in the Background” on page 7-6

Acquire Sound Pressure Data

This example shows how to acquire sound data from an NI 9234. The device is in an NI cDAQ-9178 chassis, on slot 3 with ID `cDAQ1Mod3`.

Create a session, and add an analog input channel with `Microphone` measurement type:

```
s = daq.createSession('ni');
addAnalogInputChannel(s,'cDAQ1Mod3', 0, 'Microphone');
```

Set the channels sensitivity to `0.037 v/pa`.

```
s.Channels.Sensitivity = 0.037;
```

Examine the channel properties:

```
s.Channels(1)
```

```
ans =

Data acquisition analog input microphone channel 'ai0' on device 'cDAQ1Mod3':

          Sensitivity: 0.037
MaxSoundPressureLevel: 136
    ExcitationCurrent: 0.002
     ExcitationSource: Internal
             Coupling: AC
       TerminalConfig: PseudoDifferential
                Range: -135 to +135 Pascals
                 Name: ''
                   ID: 'ai0'
               Device: [1x1 daq.ni.CompactDAQModule]
      MeasurementType: 'Microphone'
        ADCTimingMode: ''
```

Change the maximum sound pressure level to `100db` and examine channel properties.

```
s.Channels.MaxSoundPressureLevel = 100;
s.Channels(1)
```

```
ans =

Data acquisition analog input microphone channel 'ai0' on device 'cDAQ1Mod3':

          Sensitivity: 0.037
MaxSoundPressureLevel: 100
    ExcitationCurrent: 0.002
     ExcitationSource: Internal
             Coupling: AC
       TerminalConfig: PseudoDifferential
                Range: -135 to +135 Pascals
```

```
              Name: ''
                ID: 'ai0'
            Device: [1x1 daq.ni.CompactDAQModule]
   MeasurementType: 'Microphone'
     ADCTimingMode: ''
```

Set acquisition session duration to 4 seconds.

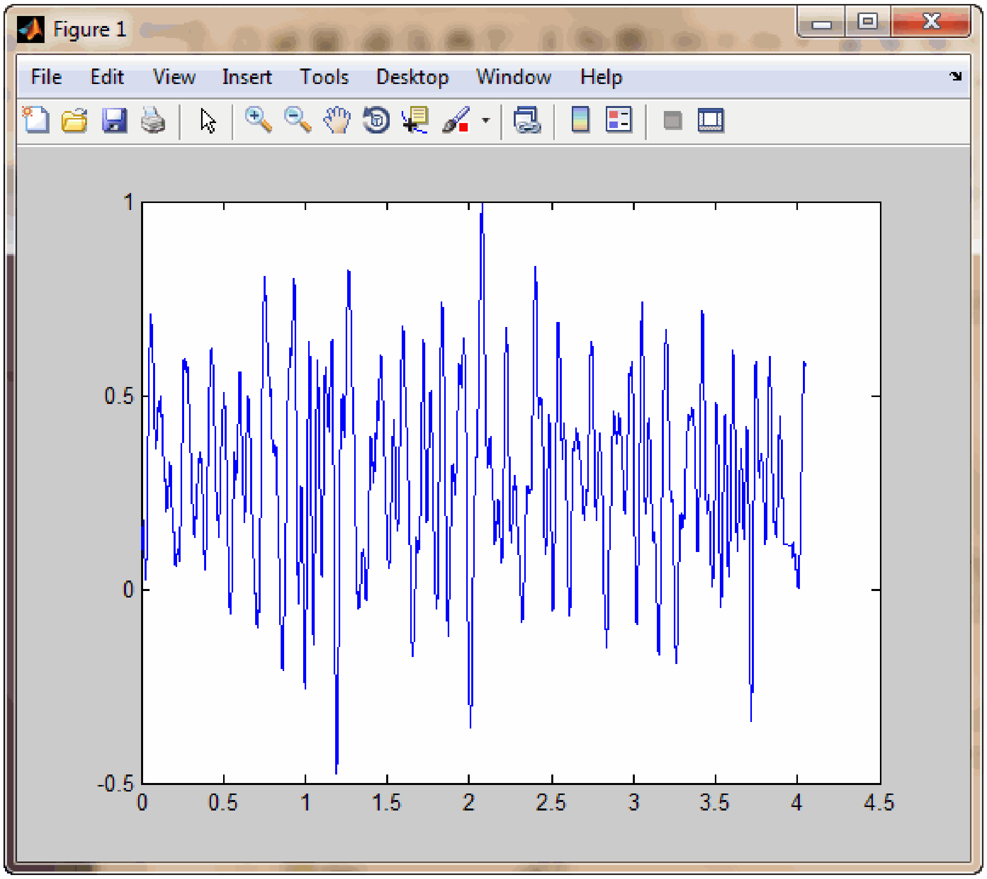
```
s.DurationInSeconds = 4;
```

Acquire the data against time and save it in a variable.

```
[data,time] = startForeground(s);
```

Plot the data.

```
plot(time, data)
```



See Also

Related Examples

- "Acquire Data in the Foreground" on page 7-2
- "Acquire Data in the Background" on page 7-6

Acquire IEPE Data

This example shows how to acquire IEPE data using an NI 9234. The device is in an NI cDAQ-9178 chassis on slot 3 with ID `cDAQ1Mod3`.

Create a session, and add an analog input channel with `IEPE` measurement type.

```
s = daq.createSession('ni');
addAnalogInputChannel(s,'cDAQ1Mod3',0,'IEPE');
```

Change the channel property `ExcitationCurrent` to `0.004` volts.

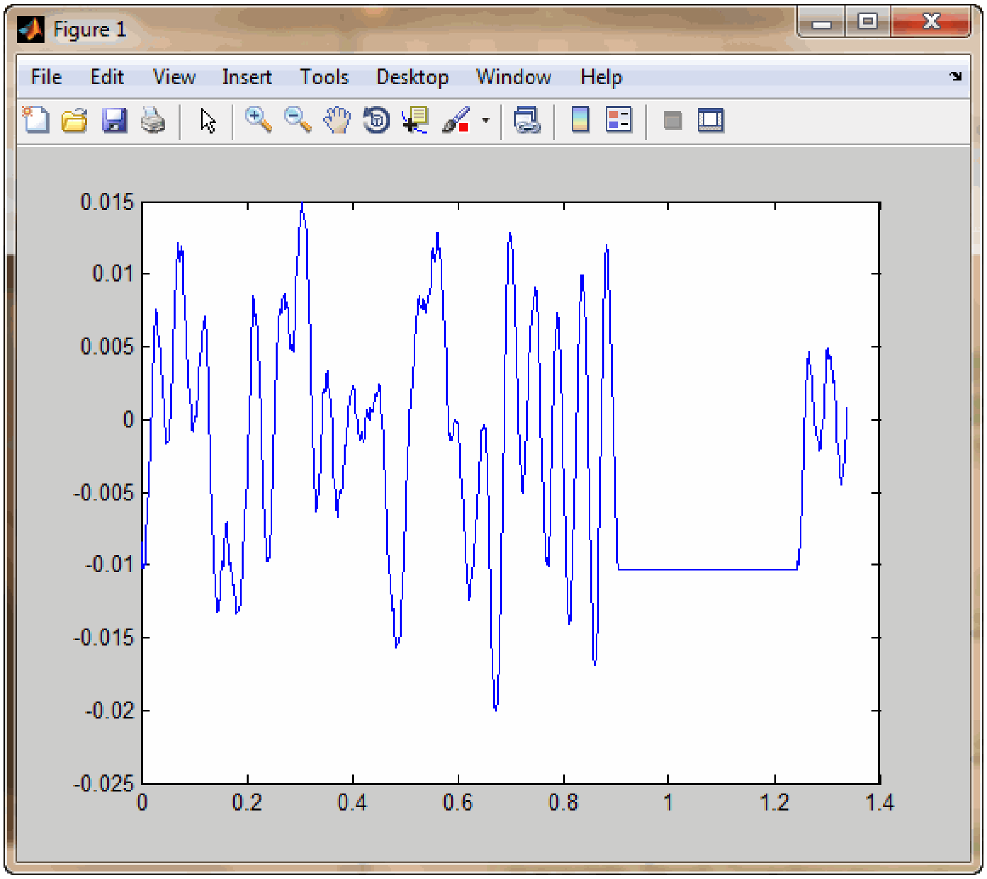
```
s.Channels(1).ExcitationCurrent = .004;
```

Acquire the data against time and save it in a variable.

```
[data,time] = startForeground(s);
```

Plot the data.

```
plot(time,data)
```



See Also

Related Examples

- "Acquire Data in the Foreground" on page 7-2
- "Acquire Data in the Background" on page 7-6

Getting Started Acquiring Data with Digilent® Analog Discovery™

This example shows how to acquire analog input voltage data (at a sampling rate of 300kHz). The dynamic range of the incoming signal is -2.5 to 2.5 volts. You will use the session-based interface with the Digilent Analog Discovery hardware.

Create a session with a Digilent device

Discover Digilent devices connected to your system using `daq.getDevices` and create a session using the listed Digilent device.

```
s = daq.createSession('digilent')
```

```
s = 

Data acquisition session using Digilent Inc. hardware:
   Will run for 1 second (10000 scans) at 10000 scans/second.
   No channels have been added.
```

Add an analog input channel

Add an analog input channel with device ID `AD1` and channel ID `1`. Set the measurement type to `Voltage`.

```
ch = addAnalogInputChannel(s,'AD1', 1, 'Voltage')
```

```
ch = 

Data acquisition analog input voltage channel '1' on device 'AD1':

       Coupling: DC
 TerminalConfig: Differential
          Range: -25 to +25 Volts
           Name: ''
             ID: '1'
         Device: [1x1 daq.di.DeviceInfo]
MeasurementType: 'Voltage'
```

Set session and channel properties

Set the sampling rate to 300kHz and the channel range to -2.5 to 2.5 volts. Set the duration to 0.5 seconds.

```
s.Rate = 300e3;
s.Channels.Range = [-2.5 2.5];
s.DurationInSeconds = 0.5
```

```
s =

Data acquisition session using Digilent Inc. hardware:
   Will run for 0.5 seconds (150000 scans) at 300000 scans/second.
   Number of channels: 1
      index Type Device Channel MeasurementType       Range         Name
      ----- ---- ------ ------- --------------- ------------------ ----
      1     ai   AD1    1       Voltage (Diff)  -2.5 to +2.5 Volts
```

Acquire a single sample

Acquire a single scan on-demand, measuring the data and trigger time.

```
[singleReading, triggerTime] = inputSingleScan(s)
```

```
singleReading =

   -0.0104


triggerTime =

   7.3532e+05
```

Acquire timestamped data

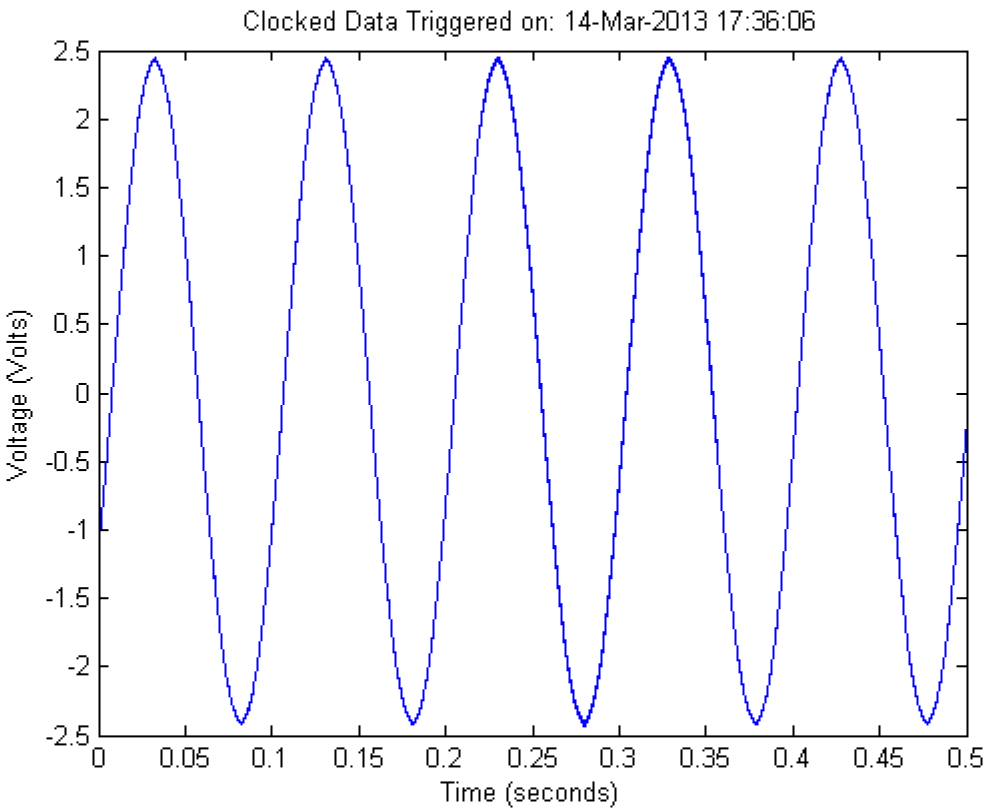
Start a clocked foreground acquisition.

```
[data, timestamps, triggerTime] = startForeground(s);
```

Display the results

```
plot(timestamps, data);
xlabel('Time (seconds)')
```

```
ylabel('Voltage (Volts)')
title(['Clocked Data Triggered on: ' datestr(triggerTime)])
```



See Also

Related Examples

- “Acquire Data in the Foreground” on page 7-2
- “Acquire Data in the Background” on page 7-6

Generate Signals in the Foreground

This example shows how to generate data using an NI 9263 device with ID cDAQ1Mod2.

Create a session object and save it to the variable, `s`:

```
s = daq.createSession('ni');
```

Change the scan rate of the session object to generate 10,000 scans per second:

```
s.Rate = 10000

s =

Data acquisition session using National Instruments hardware:
   Will run for 1 second (10000 scans) at 10000 scans/second.
   Operation starts immediately.
      No channels have been added.
```

Add an analog output `'Voltage'` channel:

```
addAnalogOutputChannel(s,'cDAQ1Mod2',0,'Voltage')

ans =

Data acquisition session using National Instruments hardware:
   No data queued.  Will run at 1000 scans/second.      Number of channels: 1
      index Type  Device   Channel MeasurementType      Range       Name
      ----- ---- --------- ------- --------------- ---------------- ----
      1     ao   cDAQ1Mod2 ao0     Voltage         -10 to +10 Volts
```

Specify the channel ID on NI devices using a terminal name, like `'ao1'`, or a numeric equivalent like `1`.

Create the data to output:

```
outputData = linspace(-1, 1, 2200)';
```

Queue the data:

```
queueOutputData(s,outputData);
```

The duration changes to 0.22 seconds based on the length of the queued data and the specified scan rate. When the session contains output channels, duration and number of scans become read-only properties of the session. The number of scans in a session is determined by the amount of data queued and the duration is determined by $\frac{s.ScansQueued}{s.Rate}$.

Display the session object to see this change:

```
s

s =

Data acquisition session using National Instruments hardware:
   Will run for 2200 scans (0.22 seconds) at 10000 scans/second.
   .All devices synchronized using cDAQ1 CompactDAQ chassis backplane. (Details)
      Number of channels: 1
      index Type  Device    Channel MeasurementType Range            Name
      ----- ---- --------- ------- --------------- ---------------- ----
      1     ao   cDAQ1Mod2 ao0     Voltage (Diff)  -10 to +10 Volts
```

Generate the data. MATLAB returns once the generation is complete.

```
startForeground(s);
```

See Also

Related Examples

- “Generate Signals in the Background” on page 7-26

Generate Signals Using Multiple Channels

This example shows how to generate data from multiple channels and multiple devices. The example generates data using channels from an NI 9263 voltage device with ID cDAQ1Mod2 and an NI 9265 current device with ID cDAQ1Mod8.

Create an NI session object and add two analog output `'Voltage'` channels to cDAQ1Mod2:

```
s = daq.createSession('ni');
addAnalogOutputChannel(s,'cDAQ1Mod2', 2:3, 'Voltage');
```

Step 2. Add one output 'Current' channel on cDAQ1Mod8:

```
addAnalogOutputChannel(s,'cDAQ1Mod8', 'ao2', 'Current')
ans =

Data acquisition session using National Instruments hardware:
   No data queued.  Will run at 1000 scans/second.
   All devices synchronized using cDAQ1 CompactDAQ chassis backplane. (Details)
      Number of channels: 3
      index Type  Device   Channel MeasurementType Range            Name
      ----- ---- --------- ------- --------------- ---------------- ----
      1     ao   cDAQ1Mod2 ao2     Voltage (Diff)  -10 to +10 Volts
      2     ao   cDAQ1Mod2 ao3     Voltage (Diff)  -10 to +10 Volts
      3     ao   cDAQ1Mod8 ao2     Current         0 to +0.020 A
```

Specify the channel ID on NI devices using a terminal name, like `ao1`, or a numeric equivalent like `1`.

Create one set of data to output for each added channel:

```
outputData(:,1) = linspace(-1,1,1000);
outputData(:,2) = linspace(-2,2,1000)';
outputData(:,3) = linspace(0,0.02,1000)';
```

Queue the output data:

```
queueOutputData(s,outputData);
```

Step 5. Generate the data:

```
startForeground(s);
```

See Also

Related Examples

- “Generate Signals in the Foreground” on page 7-22
- “Generate Signals in the Background” on page 7-26

Generate Signals in the Background

This example shows how to generate signals in the background.

Create an NI session object and add an analog output `'Voltage'` channel to cDAQ1Mod2:

```
s = daq.createSession('ni');
addAnalogOutputChannel(s,'cDAQ1Mod2', 'ao0', 'Voltage');
```

Specify the channel ID on NI devices using a terminal name, like `ao1`, or a numeric equivalent like `1`.

Create the data to output:

```
outputData = (linspace(-1, 1, 1000)');
```

Queue the output data:

```
queueOutputData(s,outputData);
```

Generate the signal:

```
startBackground(s);
```

You can execute other MATLAB commands while the generation is in progress. In this example, issue a `pause()`, which causes the MATLAB command line to wait for you to press any key.

```
pause();
```

See Also

Related Examples

- “Generate Signals in the Foreground” on page 7-22

Generate Signals in the Background Continuously

This example shows how to continuously generate signals. A continuous background generation depends on events and listeners to allow your code to enable continuous queuing of data and to react to any errors as they occur. For details, see Events and Listeners — Concepts in the MATLAB Object-Oriented Programming documentation. In this example, you generate from an NI 9263 device with ID cDAQ1Mod2 using a listener on the `DataRequired` event.

Listeners execute a callback function when notified that the event has occurred. Use `Session.addlistener` to create the listener object that executes your callback function.

Create an NI session object and add an analog output `'Voltage'` channel on cDAQ1Mod2:

```
s = daq.createSession('ni');
addAnalogOutputChannel(s,'cDAQ1Mod2', 'ao0', 'Voltage');
```

Specify the channel ID on NI devices using a terminal name, like `'ao1'`, or a numeric equivalent like `1`.

Create the data to output and queue the output data.

```
queueOutputData(s,linspace(1, 10, 1000)');
```

Add the listener to the `DataRequired` event and assign it to the variable `lh`:

```
lh = addlistener(s,'DataRequired',@queueMoreData);
```

Step 4. Create a simple callback function to generate the data and save it as `queueMoreData.m` in your working folder:

```
  function queueMoreData(src,event)
        queueOutputData(s,linspace(1, 10, 1000)');
  end
```

Generate the signal:

```
startBackground(s);
```

You can execute other MATLAB commands while the generation is in progress. In this example, issue a `pause()`, which causes the MATLAB command line to wait for you to press any key.

```
pause();
```

Delete the listener:

```
delete(lh)
```

See Also

Related Examples

- “Generate Signals in the Background” on page 7-26

Getting Started Generating Data with Digilent® Analog Discovery™

This example shows how to generate analog output voltage data (at a rate of 300kHz). The output voltage-range of the outgoing signal is -5.0 to +5.0 volts. You will use the session-based interface with Digilent Analog Discovery hardware.

Create a session with a Digilent device

Discover Digilent devices connected to your system using `daq.getDevices` and create a session using the listed Digilent device.

```
s = daq.createSession('digilent')
```

```
s =

Data acquisition session using Digilent Inc. hardware:
   Will run for 1 second (10000 scans) at 10000 scans/second.
   No channels have been added.
```

Add an analog output channel

Add an analog output channel with device ID `AD1` and channel ID `1`. Set the measurement type to `Voltage`.

```
ch = addAnalogOutputChannel(s,'AD1', 1, 'Voltage')
```

```
ch =

Data acquisition analog output voltage channel '1' on device 'AD1':

 TerminalConfig: SingleEnded
          Range: -5.0 to +5.0 Volts
           Name: ''
             ID: '1'
         Device: [1x1 daq.di.DeviceInfo]
MeasurementType: 'Voltage'
```

Generate a single sample

Generate a single scan on-demand.

```
outVal = 2;
outputSingleScan(s,outVal);
```

Set session and channel properties

Set the generation rate to 300kHz.

```
rate = 300e3;
s.Rate = rate;
```

Define the output waveform

Generate a 10 Hz sine-wave for half a second. The length of the output waveform and the specified output rate define the duration of the waveform.

```
f = 10;
duration = 0.5;
t = (1:(duration*rate))/rate;
output = sin(2*pi*f*t)';
```

Generate continuous data

Queue some data and start a clocked foreground generation.

```
queueOutputData(s,output);
startForeground(s);
```

See Also

Related Examples

- “Generate Signals in the Foreground” on page 7-22
- “Generate Signals in the Background” on page 7-26

Acquire Data and Generate Signals Simultaneously

This example shows how to acquire data from an NI 9205 device with ID cDAQ1Mod1 and generate signals using an NI 9263 device with ID cDAQ1Mod2.

You can acquire data and generate signals at the same time, on devices on the same chassis. When the session contains output channels, duration and number of scans become read-only properties of the session. The number of scans in a session is determined by the amount of data queued, and the duration is determined by $\frac{s.ScansQueued}{s.Rate}$.

Step 1. Create an NI session object and add one analog input channel on cDAQ1Mod1 and one analog output channel on cDAQ1Mod2:

```
s = daq.createSession('ni');
addAnalogInputChannel(s,'cDAQ1Mod1', 'ai0', 'Voltage');
addAnalogOutputChannel(s,'cDAQ1Mod2', 'ao0', 'Voltage')
```

```
ans =

Data acquisition session using National Instruments hardware:
   No data queued.  Will run at 1000 scans/second.
   Number of channels: 2
      index Type  Device   Channel   MeasurementType        Range       Name
      ----- ---- --------- ------- ------------------- ---------------- ----
      1     ai   cDAQ1Mod1 ai0     Voltage (Diff)      -10 to +10 Volts
      2     ao   cDAQ1Mod2 ao0     Voltage (SingleEnd) -10 to +10 Volts
```

Queue the output data:

```
queueOutputData(s,linspace(-1, 10, 2500)');
```

Display the session object to see the change in duration and the number of scans. These values change based on the amount of data queued.

```
s
```

```
s =

Data acquisition session using National Instruments hardware:
   Will run for 2500 scans (2.5 seconds) at 1000 scans/second.
   All devices synchronized using cDAQ1 CompactDAQ chassis backplane. (Details)
      Number of channels: 2
      index Type  Device   Channel MeasurementType Range            Name
      ----- ---- --------- ------- --------------- ---------------- ----
      1     ai   cDAQ1Mod1 ai0     Voltage (Diff)  -10 to +10 Volts
      2     ao   cDAQ1Mod2 ao0     Voltage (Diff)  -10 to +10 Volts
```

Acquire the data store it in the variable, `acquiredData`:

```
acquiredData = startForeground(s);
plot(acquiredData)
```

See Also

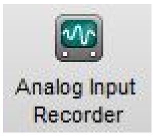
Related Examples

- “Generate Signals in the Foreground” on page 7-22
- “Generate Signals in the Background” on page 7-26
- “Acquire Data in the Foreground” on page 7-2
- “Acquire Data in the Background” on page 7-6

Acquire Data with the Analog Input Recorder

This topic shows how to use the Analog Input Recorder app to view and record data from an NI USB-6218 device.

To open the Analog Input Recorder, on the MATLAB Toolstrip, on the **Apps** tab, in the **Test and Measurement** section, click the Analog Input Recorder.



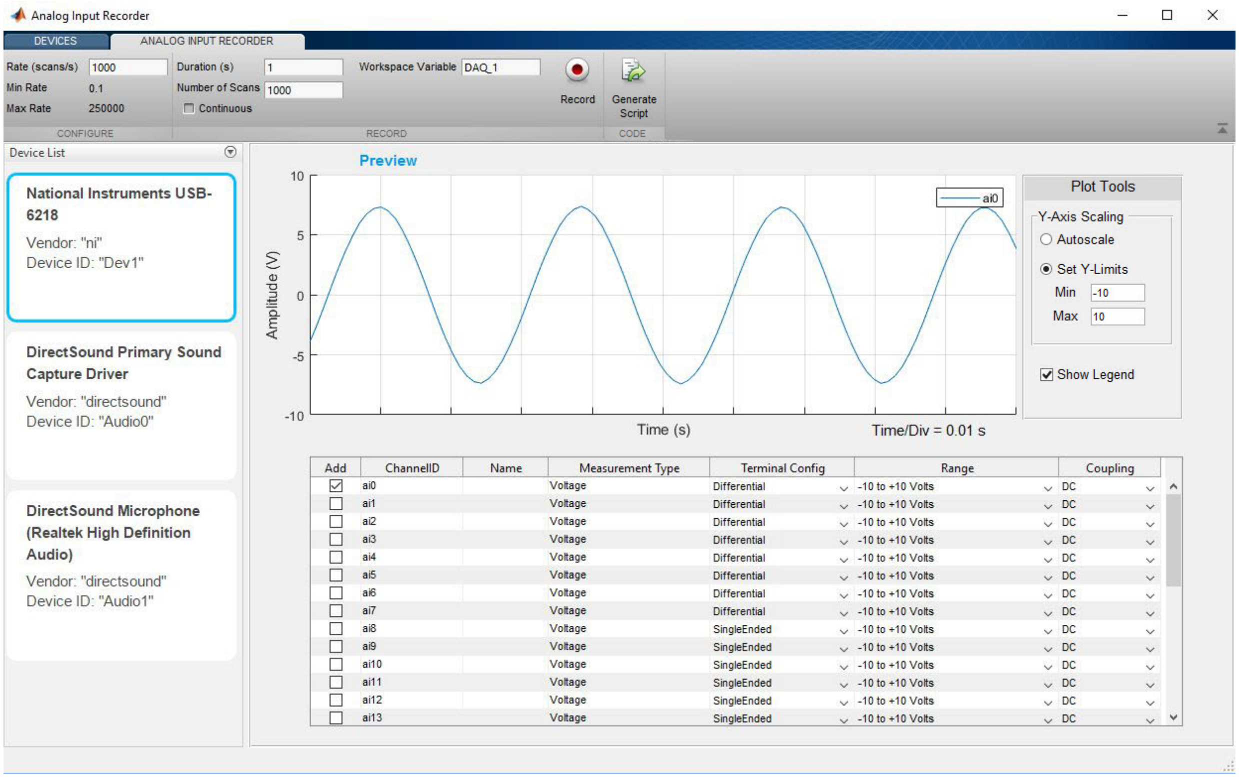
Upon opening, the Analog Input Recorder attempts to find all your attached analog and audio input devices.

Note Opening the Analog Input Recorder deletes all your existing data acquisition sessions in MATLAB.

The data acquisition session created by the Analog Input Recorder is not accessible from the MATLAB command line.

If you plug in a device while the app is open, you must refresh the listing for access to the device. On the **Devices** tab, click **Refresh**. Use the same procedure to remove a device from the listing after unplugging it.

Select the device you want to use in the **Device List**. The app immediately starts a preview of the analog input using default settings.



Rate (scans/s) 1000
Min Rate 0.1
Max Rate 250000
Duration (s) 1
Number of Scans 1000
☐ Continuous
Workspace Variable DAQ_1

National Instruments USB-6218
Vendor: "ni"
Device ID: "Dev1"

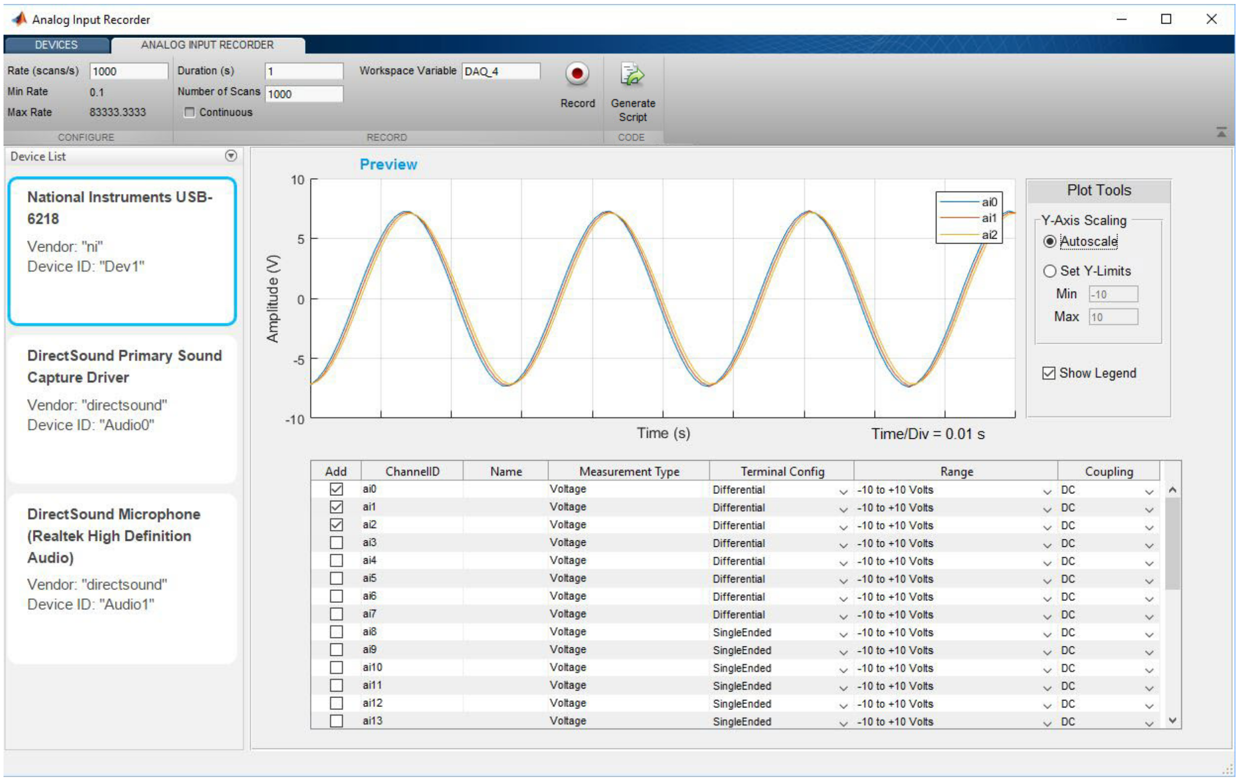
DirectSound Primary Sound Capture Driver
Vendor: "directsound"
Device ID: "Audio0"

DirectSound Microphone (Realtek High Definition Audio)
Vendor: "directsound"
Device ID: "Audio1"

Y-Axis Scaling: ○ Autoscale ● Set Y-Limits
Min -10
Max 10
☑ Show Legend

Add	ChannelID	Name	Measurement Type	Terminal Config	Range	Coupling
☑	ai0		Voltage	Differential	-10 to +10 Volts	DC
☐	ai1		Voltage	Differential	-10 to +10 Volts	DC
☐	ai2		Voltage	Differential	-10 to +10 Volts	DC
☐	ai3		Voltage	Differential	-10 to +10 Volts	DC
☐	ai4		Voltage	Differential	-10 to +10 Volts	DC
☐	ai5		Voltage	Differential	-10 to +10 Volts	DC
☐	ai6		Voltage	Differential	-10 to +10 Volts	DC
☐	ai7		Voltage	Differential	-10 to +10 Volts	DC
☐	ai8		Voltage	SingleEnded	-10 to +10 Volts	DC
☐	ai9		Voltage	SingleEnded	-10 to +10 Volts	DC
☐	ai10		Voltage	SingleEnded	-10 to +10 Volts	DC
☐	ai11		Voltage	SingleEnded	-10 to +10 Volts	DC
☐	ai12		Voltage	SingleEnded	-10 to +10 Volts	DC
☐	ai13		Voltage	SingleEnded	-10 to +10 Volts	DC

Modify any scan and channel settings for your specific needs. The following image shows the app displaying three channels of the device. Notice that the **Max Rate** value has changed with the number of channels; this relationship depends on the device.



Rate (scans/s) 1000
Min Rate 0.1
Max Rate 83333.3333
Duration (s) 1
Number of Scans 1000
☐ Continuous
Workspace Variable DAQ_4

National Instruments USB-6218
Vendor: "ni"
Device ID: "Dev1"

DirectSound Primary Sound Capture Driver
Vendor: "directsound"
Device ID: "Audio0"

DirectSound Microphone (Realtek High Definition Audio)
Vendor: "directsound"
Device ID: "Audio1"

● Autoscale
○ Set Y-Limits
Min -10
Max 10
☑ Show Legend

Add	ChannelID	Name	Measurement Type	Terminal Config	Range	Coupling
☑	ai0		Voltage	Differential	-10 to +10 Volts	DC
☑	ai1		Voltage	Differential	-10 to +10 Volts	DC
☑	ai2		Voltage	Differential	-10 to +10 Volts	DC
☐	ai3		Voltage	Differential	-10 to +10 Volts	DC
☐	ai4		Voltage	Differential	-10 to +10 Volts	DC
☐	ai5		Voltage	Differential	-10 to +10 Volts	DC
☐	ai6		Voltage	Differential	-10 to +10 Volts	DC
☐	ai7		Voltage	Differential	-10 to +10 Volts	DC
☐	ai8		Voltage	SingleEnded	-10 to +10 Volts	DC
☐	ai9		Voltage	SingleEnded	-10 to +10 Volts	DC
☐	ai10		Voltage	SingleEnded	-10 to +10 Volts	DC
☐	ai11		Voltage	SingleEnded	-10 to +10 Volts	DC
☐	ai12		Voltage	SingleEnded	-10 to +10 Volts	DC
☐	ai13		Voltage	SingleEnded	-10 to +10 Volts	DC

Set values for **Number of Scans** or **Duration**, and **Rate**.

Check **Continuous** if you want to override the duration or number of scans. In this mode, recording will continue until you explicitly stop it.

When you are ready to start recording data, click **Record**.

When recording is complete, either because the specified number of scan is recorded or you click **Stop**, the logged data is assigned to the indicated MATLAB Workspace variable. By default, the variable starts as DAQ_1, and its name is incremented with every recording, but you can specify any valid MATLAB variable name. The variable is assigned an M-by-N timetable, where M is the number of scans and N is the number of channels.

The following commands show the beginning of the acquired timetable for a multiple channel recording.

```
whos
```

```
  Name           Size                Bytes  Class        Attributes

  DAQ_4      1000x3                  33315  timetable
```

View the first four rows of the timetable.

```
DAQ_4(1:4,:)
```

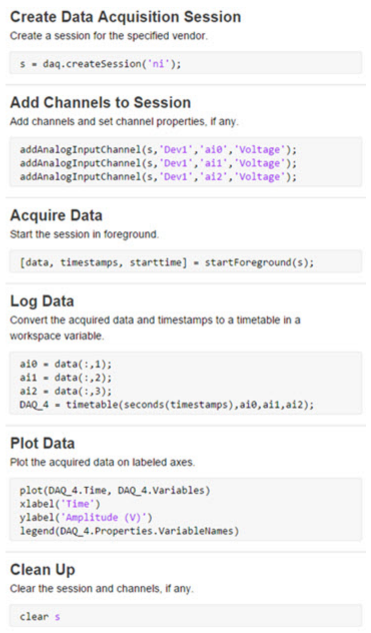
```
ans =

  4×3 timetable

       Time         ai0         ai1         ai2   
    _________    _______    ________    _______

    0 sec        0.59036      1.1226     1.6268
    0.001 sec    -1.0661    -0.49113    0.07001
    0.002 sec    -2.6327     -2.0683    -1.4901
    0.003 sec    -4.0592     -3.5349    -2.9738
```

The timestamp elements of the table are relative to the first scan. The absolute time of the first scan is available in the timetable `UserData` property. For example,

```
datestr(DAQ_4.Properties.UserData.StartTime,'dd-mmm-yyyy HH:MM:SS')
```

```
    '30-Jun-2017 15:24:10'
```

In the Analog Input Recorder, click **Generate Script** for the app to open the MATLAB editor and display the code for recording data. The following code is generated for the finite (non-continuous) 3-channel recording of this example. Notice that this code uses the `startForeground` function; a continuous recording would use `startBackground`.



Create Data Acquisition Session

Create a session for the specified vendor.

```
s = daq.createSession('ni');
```

Add Channels to Session

Add channels and set channel properties, if any.

```
addAnalogInputChannel(s,'Dev1','ai0','Voltage');
addAnalogInputChannel(s,'Dev1','ai1','Voltage');
addAnalogInputChannel(s,'Dev1','ai2','Voltage');
```

Acquire Data

Start the session in foreground.

```
[data, timestamps, starttime] = startForeground(s);
```

Log Data

Convert the acquired data and timestamps to a timetable in a workspace variable.

```
ai0 = data(:,1);
ai1 = data(:,2);
ai2 = data(:,3);
DAQ_4 = timetable(seconds(timestamps),ai0,ai1,ai2);
```

Plot Data

Plot the acquired data on labeled axes.

```
plot(DAQ_4.Time, DAQ_4.Variables)
xlabel('Time')
ylabel('Amplitude (V)')
legend(DAQ_4.Properties.VariableNames)
```

Clean Up

Clear the session and channels, if any.

```
clear s
```

See Also

Apps

Analog Input Recorder

More About

- “Generate Signals with the Analog Output Generator” on page 7-38
- “Timetables” (MATLAB)

Generate Signals with the Analog Output Generator

This topic shows how to use the Analog Output Generator app to define and generate signals from an audio device.

To open the Analog Output Generator, on the MATLAB Toolstrip, on the **Apps** tab, in the **Test and Measurement** section, click the Analog Output Generator.

Upon opening, the Analog Output Generator attempts to find all your attached analog and audio output devices.

Note Opening the Analog Output Generator deletes all your existing data acquisition sessions in MATLAB.

The data acquisition session created by the Analog Output Generator is not accessible from the MATLAB command line.

If you plug in a device while the app is open, you must refresh the listing for access to the device. On the **Devices** tab, click **Refresh**. Use the same procedure to remove a device from the listing after unplugging it.

Select the device you want to use in the **Device List**. By default, the app immediately displays a preview of a test signal.

Use the following steps to produce an audio output of the "Hallelujah" chorus from Handel's *Messiah*.

1. Select the device for your output. This might be the primary sound driver, speakers, or a headset.
2. Load the sound data into the workspace with the following command in MATLAB:

   ```
   load handel
   ```

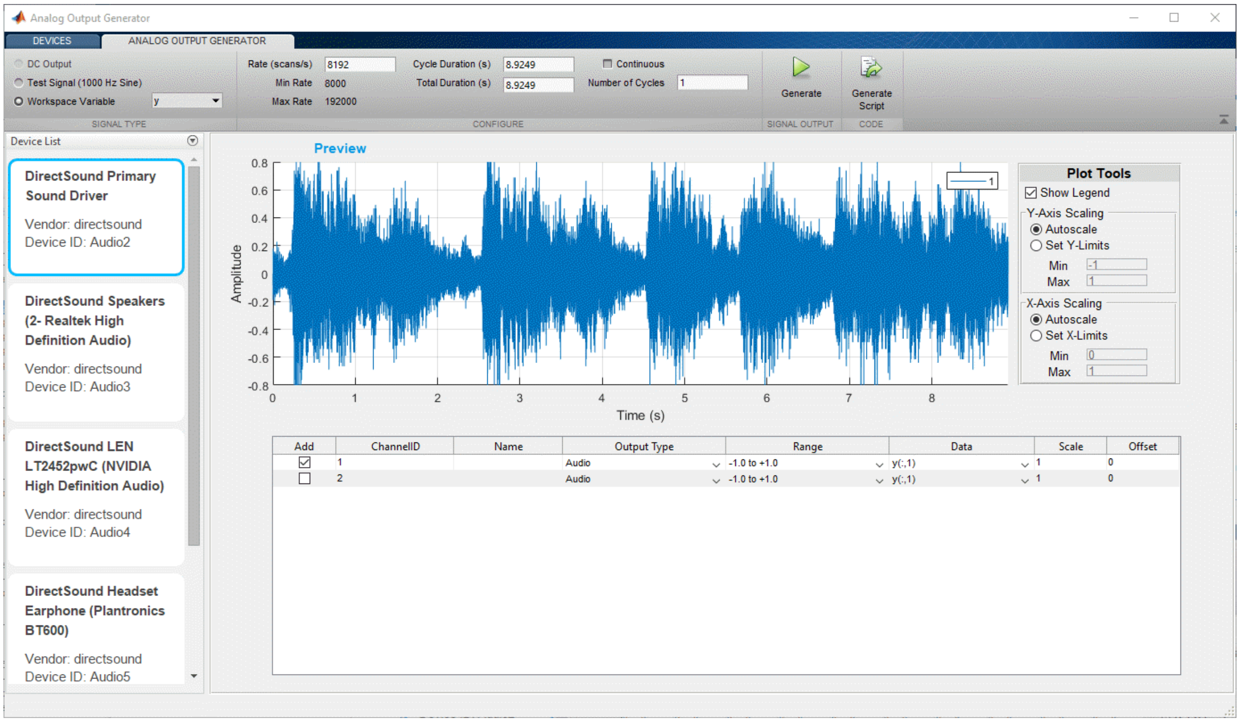
This loads two variables into your workspace. The sound data is contained in the variable `y`. The sampling rate is contained in the variable `Fs`. You will need to know the sampling rate, so display its value.

```
Fs
```

```
8192
```

3 In the Analog Output Generator, select **Workspace Variable**. In the adjacent selection list, choose `y`. This indicates the source of the data for the generator to output.

4 Enter the `Fs` value of `8192` in the **Rate** text box in the Analog Output Generator. This indicates the sampling rate. The apps should now look something like this.



DC Output
Test Signal (1000 Hz Sine)
[✓ Workspace Variable] y

Rate (scans/s) 8192
Min Rate 8000
Max Rate 192000
Cycle Duration (s) 8.9249
Total Duration (s) 8.9249
Continuous
Number of Cycles 1

DirectSound Primary Sound Driver
Vendor: directsound
Device ID: Audio2

DirectSound Speakers (2- Realtek High Definition Audio)
Vendor: directsound
Device ID: Audio3

DirectSound LEN LT2452pwC (NVIDIA High Definition Audio)
Vendor: directsound
Device ID: Audio4

DirectSound Headset Earphone (Plantronics BT600)
Vendor: directsound
Device ID: Audio5

✓ Show Legend
Y-Axis Scaling: [✓ Autoscale] Set Y-Limits, Min -1, Max 1
X-Axis Scaling: [✓ Autoscale] Set X-Limits, Min 0, Max 1

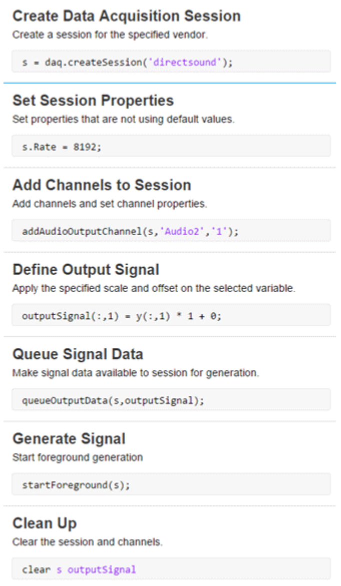
Add	ChannelID	Name	Output Type	Range	Data	Scale	Offset
✓	1		Audio	-1.0 to +1.0	y(:,1)	1	0
	2		Audio	-1.0 to +1.0	y(:,1)	1	0

5 Click **Generate** to produce the sound output.

If you were successful in producing a sound output, try experimenting with some of the settings in the app. For example, modify the Rate value or the Number of Cycles.

Tip If you could not hear any sound, use the **Test Signal** option to generate a constant tone. Check all your hardware connections and different devices in the app until you hear the tone.

In the Analog Output Generator, click **Generate Script** for the app to open the MATLAB editor and display the code for producing the signal. The code is generated for the finite (non-continuous) output of this example. Notice that this code uses the `startForeground` function; a continuous output would use `startBackground`.



Create Data Acquisition Session

Create a session for the specified vendor.

```
s = daq.createSession('directsound');
```

Set Session Properties

Set properties that are not using default values.

```
s.Rate = 8192;
```

Add Channels to Session

Add channels and set channel properties.

```
addAudioOutputChannel(s,'Audio2','1');
```

Define Output Signal

Apply the specified scale and offset on the selected variable.

```
outputSignal(:,1) = y(:,1) * 1 + 0;
```

Queue Signal Data

Make signal data available to session for generation.

```
queueOutputData(s,outputSignal);
```

Generate Signal

Start foreground generation

```
startForeground(s);
```

Clean Up

Clear the session and channels.

```
clear s outputSignal
```

See Also

Apps
Analog Output Generator

More About

- “Acquire Data with the Analog Input Recorder” on page 7-33

Analog Devices Active Learning Module

Analog Devices ADALM1000 Hardware

MATLAB supports the Analog Devices ADALM1000 active learning module. ADALM1000 is an inexpensive evaluation platform designed for learning the fundamentals of electrical engineering. You can download associated teaching materials, reference designs, and lab projects from Analog Devices.

The support package lets you perform the following tasks in MATLAB with the ADALM1000:

- Generate voltages and waveforms, 0 to +5 V
- Sink or source current, -200 ma to +200 ma
- Simultaneously source voltage and measure current on the same channel
- Simultaneously source current and measure voltage on the same channel

See Also

More About

- “Generate and Measure Signals with Analog Devices ADALM1000” on page 8-3
- “Analog Devices ADALM1000 Limitations” on page B-6

External Websites

- ADALM1000 Overview

Generate and Measure Signals with Analog Devices ADALM1000

In this section...

Updated Function Syntax

To accommodate the ADALM1000, the following Data Acquisition Toolbox functions allow vendor-specific argument options:

- `daq.createSession` accepts the vendor argument `'adi'`.
- `addAnalogInputChannel` and `addAnalogOutputChannel` accept the device name argument `'SMU1'` (source-measurement unit), and the channel ID arguments `'A'` and `'B'` to correspond with the channel labels on the ADALM1000 module.

Source Voltage and Measure Current

This example shows how to source a voltage while measuring current on the same channel, to calculate load resistance. First program the ADALM1000 to provide a constant 5 V supply to the load, and then measure the current on the same device channel.

Set up a Data Acquisition Toolbox session to operate the ADALM100.

```
s = daq.createSession('adi');
```

Define an analog output channel in the session to source voltage from device channel `A`.

```
addAnalogOutputChannel(s,'SMU1','A','Voltage');
```

Define an analog input channel in the session to measure current on that same device channel `A`.

```
addAnalogInputChannel(s,'SMU1','A','Current');
```

Generate an output voltage, and measure the current.

```
V_load = 5;
outputSingleScan(s,V_load);
I_load = inputSingleScan(s);
outputSingleScan(s,0);  % Reset device output.
R_load = V_load/I_load
```

```
R_load =

   50.3005
```

Note The ADALM1000 continues to generate the last value programmed until you release the session. When you are finished with your signals, reset the device to output 0 volts.

Generate a Pulse

This example shows how to generate a 1-millisecond, 5-volt pulse, surrounded on either side by 10 milliseconds at 0 volts.

```
pdata = zeros(2100,1); % Column vector of 2100 samples.
pdata (1001:1100) = 5; % Pulse in middle of vector.

s = daq.createSession('adi');
addAnalogOutputChannel(s,'SMU1','B','Voltage');

queueOutputData(s,pdata);
s   % View channel configuration and scan settings.
```

```
s =

Data acquisition session using Analog Devices Inc. hardware:
   Will run for 2100 scans (0.021 seconds) at 100000 scans/second.
   Number of channels: 1
      index Type Device Channel   MeasurementType        Range       Name
      ----- ---- ------ ------- ------------------- --------------- ----
      1     ao   SMU1   B       Voltage (SingleEnd) 0 to +5.0 Volts
```

```
startForeground(s);
```

Generate Waveforms

This example shows how to simultaneously generate a 1-kHz square wave on channel A, and a 100-Hz sine wave on channel B. Each output lasts for 5 seconds.

The example requires two session channels for device channels A and B, both as output channels for voltage.

```
s = daq.createSession('adi');
addAnalogOutputChannel(s,'SMU1','A','Voltage');
addAnalogOutputChannel(s,'SMU1','B','Voltage');
```

Define the two waveforms.

```
Sq = zeros(500000,1); % 500k sample column vectors
Sw = zeros(500000,1);

% Define square wave:
for r = 1:100:499900;
    Sq(r:r+49) = 5;  % Set first 50 of each 100 samples to 5 v.
end

% Define sine wave:
for a = 1:500000
    Sw(a) = sin(a*2*pi/1000);
end
Sw = Sw + 1;  % Shift for positive voltage output

queueOutputData(s,[Sq Sw])
s   % View channel configuration and scan settings.
```

```
s =

Data acquisition session using Analog Devices Inc. hardware:
   Will run for 500000 scans (5 seconds) at 100000 scans/second.
   Number of channels: 2
      index Type Device Channel   MeasurementType        Range       Name
      ----- ---- ------ ------- ------------------- --------------- ----
      1     ao   SMU1   A       Voltage (SingleEnd) 0 to +5.0 Volts
      2     ao   SMU1   B       Voltage (SingleEnd) 0 to +5.0 Volts
```

Start generation that lasts for 5 seconds.

```
startForeground(s);
```

See Also

Functions
`addAnalogInputChannel` | `addAnalogOutputChannel` | `daq.createSession` | `inputSingleScan` | `outputSingleScan` | `startForeground`

More About

- “Analog Devices ADALM1000 Hardware” on page 8-2
- “Analog Devices ADALM1000 Limitations” on page B-6

External Websites

- ADALM1000 Overview

Session-Based Counter Input and Output

Analog and Digital Counters

Use digital and analog counters to count clock ticks and external events. Counters output a pulse train or count rising or falling edges and measure many quantities including:

- Frequency
- Edges
- PWM
- Position
- Pulse generation

Counters enable timed acquisition and synchronization.

See Also

Related Examples

- “Acquire Counter Input Data” on page 9-3
- “Generate Pulse Data on a Counter Channel” on page 9-7

Acquire Counter Input Data

In this section...

addCounterInputChannel

Use `addCounterInputChannel` to add a channel that acquires edge count from a device. You can acquire a single input data or an array by acquiring in the foreground. For details, see "Session-Based Interface Workflow" on page 5-2 for more information.

Acquire a Single EdgeCount

This example shows how to acquire a single falling edge data from an NI USB-9402 with device ID `'cDAQ1Mod5'`.

Step 1. Create a session object and save it to the variable `s`.

```
s = daq.createSession('ni');
```

Step 2. Add a counter channel with an `'EdgeCount'` measurement type.

```
ch = addCounterInputChannel(s,'cDAQ1Mod5', 'ctr0', 'EdgeCount')
```

```
ans =

Data acquisition session using National Instruments hardware:
   Will run for 1 second (1000 scans) at 1000 scans/second.
   Operation starts immediately.
      Number of channels: 1
      index Type  Device    Channel MeasurementType Range Name
      ----- ---- --------- ------- --------------- ----- ----
      1     ci   cDAQ1Mod5 ctr0    EdgeCount       n/a
```

Step 3. Change the `ActiveEdge` property to `'Falling'` and view the channel properties to see the change:

```
ch.ActiveEdge = 'Falling'

ans =

Data acquisition counter input edge count channel 'ctr0' on device 'cDAQ1Mod5':

      ActiveEdge: Falling
  CountDirection: Increment
    InitialCount: 0
        Terminal: 'PFI0'
IsCounterRunning: false
            Name: empty
              ID: 'ctr0'
          Device: [1x1 daq.ni.CompactDAQModule]
 MeasurementType: 'EdgeCount'
```

Step 4. Acquire a single scan.

```
inputSingleScan(s)

ans =

   133
```

Step 5. Reset counters from the initial count and acquire the count again.

```
resetCounters(s);
inputSingleScan(s)

ans =

71
```

Acquire a Single Frequency Count

This example shows how to acquire a single frequency scan from an NI USB-9402 with device ID `'cDAQ1Mod5'`.

Step 1. Create an acquisition session.

```
s = daq.createSession('ni');
```

Step 2. Add a counter channel with a `'Frequency'` measurement type.

```
addCounterInputChannel('cDAQ1Mod5', 'ctr0', 'Frequency')

ans =
      index Type  Device    Channel MeasurementType Range Name
      ----- ---- --------- ------- --------------- ----- ----
      1     ci   cDAQ1Mod5 ctr0    Frequency       n/a
```

Step 3. Acquire a single scan.

```
s.inputSingleScan
```

```
ans =

  9.5877e+003
```

Acquire Counter Input Data in the Foreground

This example shows how to acquire rising edge data from an NI USB-9402 with device ID `'cDAQ1Mod5'`, and plot the acquired data.

Step 1. Create an acquisition session.

```
s = daq.createSession('ni');
```

Step 2. Add a counter channel with an `'EdgeCount'` measurement type.

```
addCounterInputChannel(s,'cDAQ1Mod5', 'ctr0', 'EdgeCount')
```

```
ans =

Data acquisition session using National Instruments hardware:
   Will run for 10 seconds (10000 scans) at 1000 scans/second.
   Number of channels: 1
      index Type  Device    Channel MeasurementType Range Name
      ----- ---- --------- ------- --------------- ----- ----
      1     ci   cDAQ1Mod5 ctr0    EdgeCount       n/a
```

The counter input channel requires an external clock to perform a foreground acquisition. If you do not have an external clock, add an analog input channel from a clocked device on the same CompactDAQ chassis to the session. This example uses an NI 9205 device on the same chassis with the device ID `'cDAQ1Mod1'`.

Step 3. Add an analog input channel with a `'Voltage'` measurement type.

```
addAnalogInputChannel(s,'cDAQ1Mod1', 'ai1', 'Voltage');
```

Step 4. Acquire the data and store it in the variable `data` and plot it.

```
data =  startForeground(s);
plot (data)
```

The plot displays results from both channels in the session object:

- EdgeCount measurement
- Analog input data

Generate Pulse Data on a Counter Channel

In this section...

Use addCounterOutputChannel

Use `addCounterOutputChannel` to add a channel that generates pulses on a counter/timer subsystem. You can generate on one channel or on multiple channels on the same device using `startForeground`.

Generate Pulses on a Counter Output Channel

This example shows how to generate pulse data on an NI USB-9402 with device ID 'cDAQ1Mod5'.

Step 1. Create a session object and save it to the variable s:

```
s = daq.createSession('ni');
```

Step 2. Add a counter output channel with a `PulseGeneration` measurement type:

```
ch =  addCounterOutputChannel(s,'cDAQ1Mod5', 0, 'PulseGeneration')
```

```
ch =

Data acquisition counter output pulse generation channel 'ctr0' on device 'cDAQ1Mod5':

       IdleState: Low
    InitialDelay: 2.5e-008
       Frequency: 100
       DutyCycle: 0.5
        Terminal: 'PFI0'
            Name: empty
              ID: 'ctr0'
          Device: [1x1 daq.ni.CompactDAQModule]
 MeasurementType: 'PulseGeneration'
```

Step 3. Generate pulses in the foreground:

```
s.startForeground;
```

See Also

More About

- “Synchronize Counter Outputs from Multiple Devices” on page 14-11

Session Based Digital Operations

Digital Subsystem Channels

Digital subsystems transfer digital or logical values in bits via digital lines. You can perform clocked and non-clocked digital operations using the session-based interface in the Data Acquisition Toolbox.

Lines on the digital subsystem are added as channels to your session using `addDigitalChannel`. Digital channels can be:

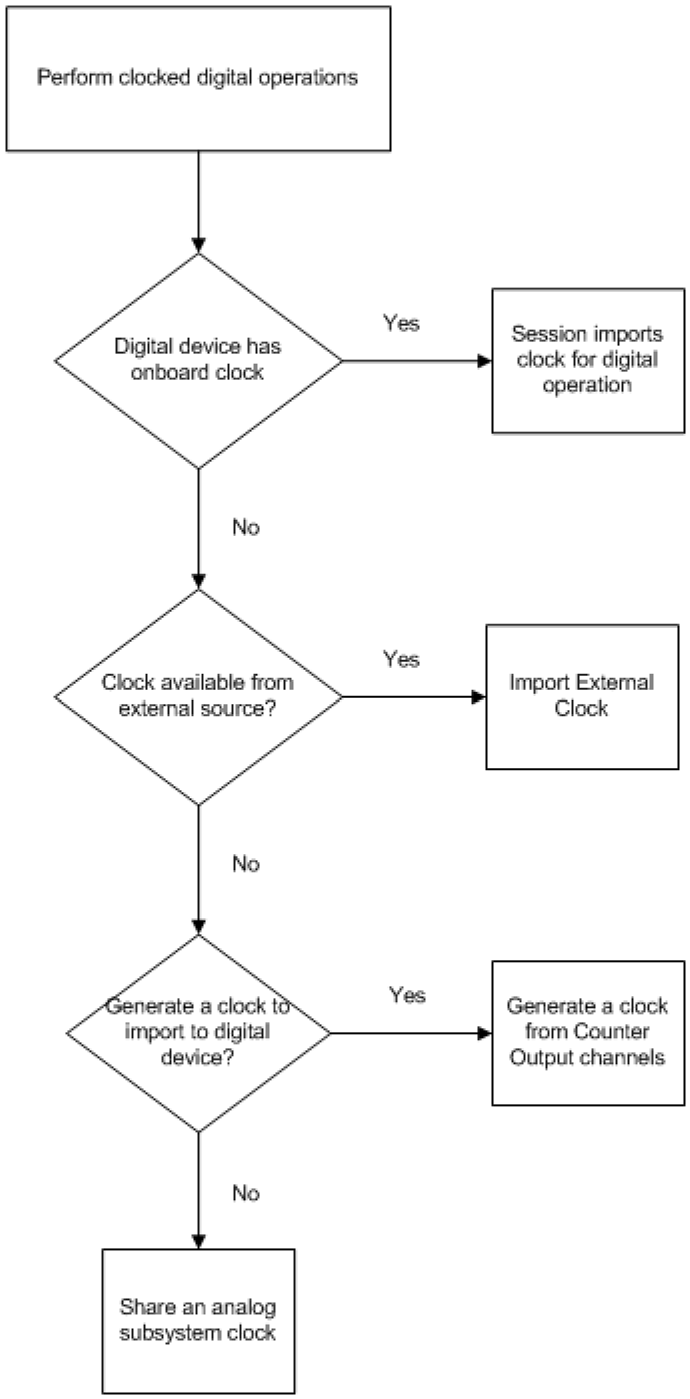
- **InputOnly**: Allows you to read digital data.
- **OutputOnly**: Allows you to write digital data.
- **Bidirectional**: Allows you to change the direction of the channel to both read and write data. By default the direction is specified as `Unknown`. You can change the direction to `Input` or `Output`.

Note If you are using bidirectional channels, you must set the Direction before you use the channel.

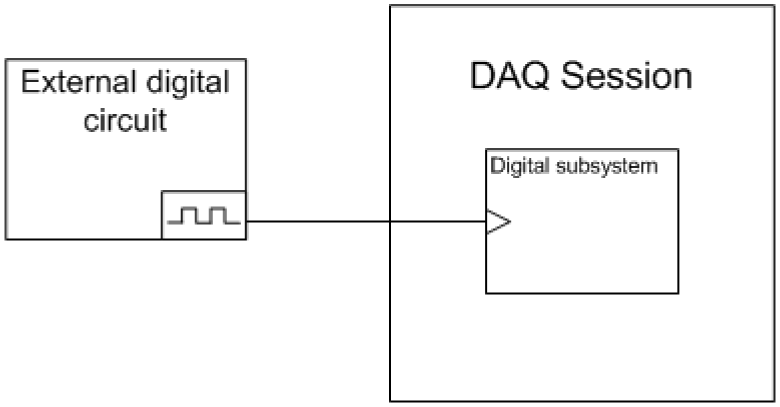
Digital Clocked Operations

With clocked operations, you can acquire or generate clocked signals at a specified scan rate for a specified duration or number of scans. These operations use hardware timing to acquire or generate at specific times. The operation is controlled by events tied to subsystem clocks. In a clocked acquisition, data is transferred from the device to your system memory and displays when the event calls for the data. In signal generation, data generated from the device is stored in memory until the configured event occurs. When an event occurs, data is sent via the digital channels to the specified devices.

Digital systems do not inherently have a clock. You can synchronize data by adding a clock in one of these ways:


Perform clocked digital operations
Digital device has onboard clock
Yes
Session imports clock for digital operation
No
Clock available from external source?
Yes
Import External Clock
No
Generate a clock to import to digital device?
Yes
Generate a clock from Counter Output channels
No
Share an analog subsystem clock

- If you have an on-board clock on your device, you can import the clock to the session.



- If your device does not have an on-board clock you can:
  - Import a clock from an external source. See “Acquire Digital Data Using an External Clock” on page 10-7 for more information.
  - Generate a clock from a Counter Output subsystem in your session and import that clock. See “Acquire Digital Data Using a Counter Output Channel as External Clock” on page 10-9 for more information.
  - Share a clock from the analog input subsystem. See “Acquire Digital Data Using a Shared Clock” on page 10-12 for more information

Access Digital Subsystem Information

This example shows how to access the device’s digital subsystem information and find line and port information using `daq.getDevices`.

Find devices connected to your system and find the ID for NI 6255.

```
d = daq.getDevices;
```

```
d =

Data acquisition devices:

index Vendor Device ID          Description
----- ------ --------- -----------------------------
```

```
1     ni     Dev1      National Instruments USB-6255
2     ni     Dev2      National Instruments USB-6363
```

View the subsystem information for `Dev1`, with index `1`.

```
subs = d(1).Subsystems;
```

View the digital subsystem information, which is the third subsystem on this device.

```
subs(3)
```

```
ans =

Digital subsystem supports:
   24 channels ('port0/line0' - 'port2/line7')
   'InputOnly','OutputOnly','Bidirectional' measurement types
```

Acquire Non-Clocked Digital Data

This example shows how to read data using two channels on an NI 6255

Find devices connected to your system and find the ID for NI 6255:

```
d = daq.getDevices;
```

```
d =

Data acquisition devices:

index Vendor Device ID           Description
----- ------ --------- -----------------------------
1     ni     Dev1      National Instruments USB-6255
2     ni     Dev2      National Instruments USB-6363
```

Create a session and add two lines from port `0` on `Dev1`:

```
s = daq.createSession('ni');
addDigitalChannel(s,'Dev1','Port0/Line0:1','InputOnly')
```

```
ans =

Data acquisition session using National Instruments hardware:
   Clocked operations using startForeground and startBackground are disabled.
   Only on-demand operations using inputSingleScan and outputSingleScan can be done.
   Number of channels: 2
      index Type Device   Channel   MeasurementType Range Name
      ----- ---- ------ ----------- --------------- ----- ----
      1     dio  Dev1   port0/line0 InputOnly       n/a
      2     dio  Dev1   port0/line1 InputOnly       n/a
```

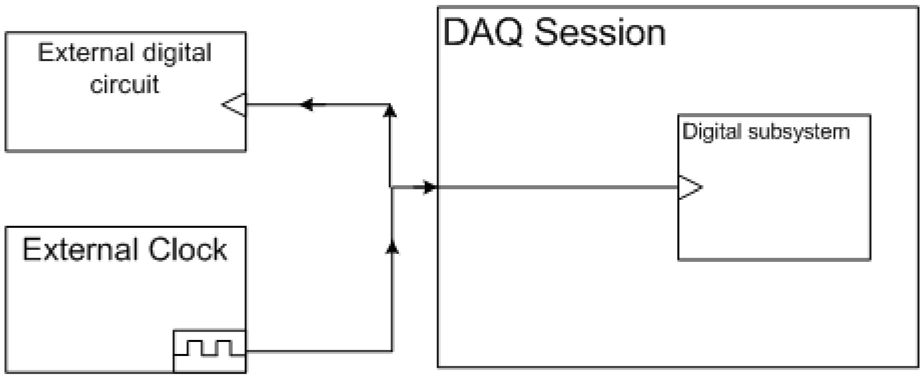
Acquire digital data:

```
inputSingleScan(s)
```

```
ans =

     1     0
```

Acquire Digital Data Using an External Clock

This example shows how to acquire digital data in the foreground by using an external scan clock.



You can use a function generator or the on-board clock from a digital circuit. Here, a function generator is physically wired to the terminal PFI9 on device NI 6255.

Create a session and add a line from port `0` line `2` on `Dev1`.

```
s = daq.createSession('ni');
addDigitalChannel(s,'Dev1','Port0/Line2','InputOnly');
```

Note Not all devices support clocked (hardware timed) digital I/O operations with `startForeground` and `startBackground`. For these devices you can use software timed operations with `inputSingleScan` and `outputSingleScan`.

Devices that support clocked digital I/O operations might not support them on all ports. Check your device specifications.

Set the rate of your session to the expected rate of your external scan clock.

```
s.Rate = 1000
```

Note Importing an external clock does not automatically set the rate of your session. Manually set the session's rate to match the expected external clock frequency.

Add an external scan clock to your device on terminal `PFI9`. For more information see `Terminals` property.

```
addClockConnection(s,'External','Dev1/PFI9','ScanClock')
```

```
ans = 


Scan Clock is provided externally and will be received by
'Dev1' at terminal 'PFI9'.

       Source: 'External'
  Destination: 'Dev1/PFI9'
         Type: ScanClock
```

Acquire clocked data and plot it.

```
dataIn = startForeground(s);
plot(dataIn)
```

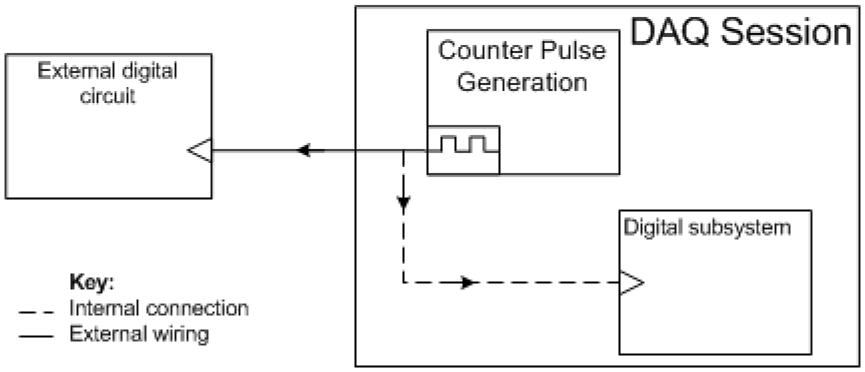
See Also

Related Examples

- “Acquire Digital Data Using a Shared Clock” on page 10-12
- “Acquire Digital Data Using a Counter Output Channel as External Clock” on page 10-9

Acquire Digital Data Using a Counter Output Channel as External Clock

This example shows how to acquire digital data using a counter output channel that generates pulses as an external clock. The counter provides the clock in this acquisition.



In this example, we will generate a clock in one session using a counter output channel and export the clock to another session that acquires clocked digital data.

Note Importing an external clock does not automatically set the rate of your session. Manually set the session's rate to match the expected external clock frequency.

Generate a Clock Using a Counter Output Channel

This example shows how to create a clock session with a counter output channel that will continuously generate frequency pulses in the background. Use this channel as an external clock in your clocked digital acquisition.

Create a clock frequency that you will use to synchronize the frequency and rate of your counter output as well as the rate of your digital acquisition in the next step.

```
clockFreq = 100;
```

Create a session and add a counter output channel with `PulseGeneration` measurement type.

```
sClk = daq.createSession('ni');
ch1 = addCounterOutputChannel(sClk,'Dev1',0,'PulseGeneration')
```

Tip Make sure the counter channel you add is not being used in a different session. You will get a terminal conflict error if the hardware is reserved in another session.

Save the counter output terminal to a variable. You will use this terminal in your digital session to specify the external clock that synchronizes your digital clocked operations.

```
clkTerminal = ch1.Terminal;
```

You will use this terminal in your digital session to specify the external clock that synchronizes your digital clocked operations.

Set the frequency of your counter session to the clock frequency.

```
ch1.Frequency = clockFreq
```

Set the session to continuous mode.

```
sClk.IsContinuous = true;
```

Use Counter Clock To Acquire Clocked Digital Data

This example shows how to create a digital input session and import an external clock from the clock session.

Create a session and add a line from port `0` line `2` on `Dev1`.

```
s = daq.createSession('ni');
addDigitalChannel(s,'Dev1','Port0/Line2','InputOnly')
```

Note Not all devices support clocked (hardware timed) digital I/O operations with `startForeground` and `startBackground`. For these devices you can use software timed operations with `inputSingleScan` and `outputSingleScan`.

Devices that support clocked digital I/O operations might not support them on all ports. Check your device specifications.

Tip PFI terminal resources may be shared. Check your device routing in NI MAX.

Set the sessions scan rate to the same as the rate and the frequency of the counter output channel.

```
s.Rate = clockFreq;
```

Import the clock from your clock session to synchronize your acquisition.

```
addClockConnection(s,'External',['Dev1/' clkTerminal],'ScanClock');
```

Start the counter output channel in the background and ensure it is running.

```
startBackground(sClk);
for i = 1:10
    if sClk.IsRunning
        break;
    else
        pause(0.1);
    end
end
```

Acquire and plot data.

```
dataIn = startForeground(s);
plot(dataIn)
```

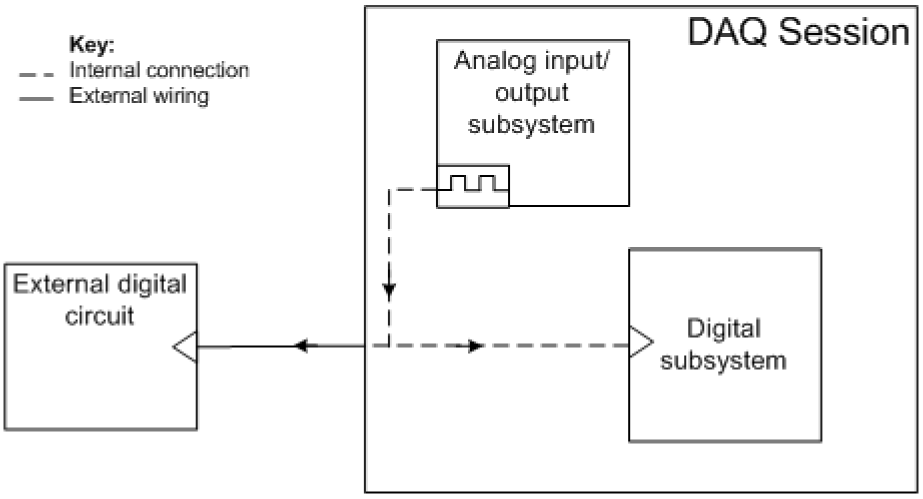
See Also

Related Examples

- “Acquire Digital Data Using a Shared Clock” on page 10-12
- “Acquire Digital Data Using an External Clock” on page 10-7

Acquire Digital Data Using a Shared Clock

This example shows how to share the clock with the analog input subsystem on your device with the digital subsystem and acquire automatically synchronized clocked data. You do not need any physical connections to share the clock. For information on automatic synchronization see Automatic Synchronization.



Create a session and add a line from port `0` line `2` on `Dev1`.

```
s = daq.createSession('ni');
addDigitalChannel(s,'Dev1','Port0/Line2','InputOnly')
```

Note Not all devices support clocked (hardware timed) digital I/O operations with `startForeground` and `startBackground`. For these devices you can use software timed operations with `inputSingleScan` and `outputSingleScan`.

Devices that support clocked digital I/O operations might not support them on all ports. Check your device specifications.

Add an analog input channel to your session.

```
addAnalogInputChannel(s,'Dev1',0,'Voltage')
```

```
ans =
Data acquisition session using National Instruments hardware:
   Will run for 1 second (1000 scans) at 1000 scans/second.
   Number of channels: 2
      index Type Device   Channel   MeasurementType      Range       Name
      ----- ---- ------ ----------- --------------- ---------------- ----
      1     dio  Dev1   port0/line2 InputOnly       n/a
      2     ai   Dev1   ai0         Voltage (Diff)  -10 to +10 Volts
```

Plot the acquired digital data.

```
dataIn = startForeground(s);
plot(dataIn(:,1))
```

See Also

Related Examples

- “Acquire Digital Data Using an External Clock” on page 10-7
- “Acquire Digital Data Using a Counter Output Channel as External Clock” on page 10-9

Acquire Digital Data Using an External Clock via Chassis PFI Terminal

The following example shows how to acquire clocked digital data using an external clock provided at the CompactDAQ chassis PFI terminal.

```
%% Setup
% cDAQ 9178 chassis
% cDAQ2Mod3 is 9402
% Digital signal is connected to PFI0 terminal of NI 9402 module
% Function generator provides 100 kHz clock to PFI0 terminal on NI 9178
% chassis

s = daq.createSession('ni');
addDigitalChannel(s, 'cDAQ2Mod3', 'Port0/Line0', 'InputOnly');
addClockConnection(s, 'External', 'cDAQ2/PFI0', 'ScanClock');
s.Rate = 100E+3;
[data, timestamps] = startForeground(s);
plot(timestamps, data);
```

See Also

Acquire Digital Data in Hexadecimal Values

This example shows how to write data using two channels on an NI 6255.

Find devices connected to your system and find the ID for NI 6255.

```
d = daq.getDevices;
```

```
d =

Data acquisition devices:

index Vendor Device ID           Description
----- ------ --------- -----------------------------
1     ni     Dev1      National Instruments USB-6255
2     ni     Dev2      National Instruments USB-6363
```

Create a session and add four lines from port `0` on `Dev1`.

```
s = daq.createSession('ni');
addDigitalChannel(s,'Dev1','Port0/Line0:3`','InputOnly')
```

```
ans =

Data acquisition session using National Instruments hardware:
   Clocked operations using startForeground and startBackground are disabled.
   Only on-demand operations using inputSingleScan and outputSingleScan can be done.
   Number of channels: 4
      index Type Device   Channel   MeasurementType Range Name
      ----- ---- ------ ----------- -------------- ----- ----
      1     dio  Dev1   port0/line0 InputOnly       n/a
      2     dio  Dev1   port0/line1 InputOnly       n/a
      3     dio  Dev1   port0/line2 InputOnly       n/a
      4     dio  Dev1   port0/line3 InputOnly       n/a
```

Acquire digital data in hexadecimal values.

```
binaryVectorToHex(inputSingleScan(s))
```

```
ans =

C
```

Control Stepper Motor using Digital Outputs

This example shows how to control a stepper motor using digital output ports.

Discover Devices Supporting Digital Output

To discover a device that supports digital output:

- Issue `daq.getDevices` in the Command window.
- Click on the device name in the list returned by the command.

```
devices = daq.getDevices
```

```
devices =

Data acquisition devices:

index Vendor Device ID           Description
----- ------ --------- -------------------------------
1     ni     cDAQ1Mod1 National Instruments NI 9205
2     ni     cDAQ1Mod2 National Instruments NI 9263
3     ni     cDAQ1Mod3 National Instruments NI 9234
4     ni     cDAQ1Mod4 National Instruments NI 9201
5     ni     cDAQ1Mod5 National Instruments NI 9402
6     ni     cDAQ1Mod6 National Instruments NI 9213
7     ni     cDAQ1Mod7 National Instruments NI 9219
8     ni     cDAQ1Mod8 National Instruments NI 9265
9     ni     Dev1      National Instruments PCIe-6363
10    ni     Dev2      National Instruments NI ELVIS II
```

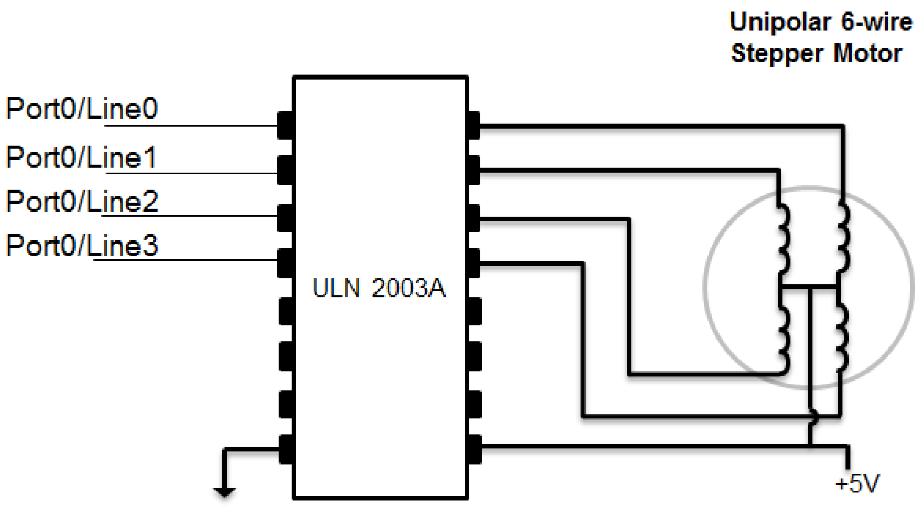
This example uses a National Instruments® ELVIS II with ID `Dev2`. Verify that its digital subsystem supports the `OutputOnly` measurement type.

```
devices(10)
```

```
ans =

ni: National Instruments NI ELVIS II (Device ID: 'Dev2')
   Analog input subsystem supports:
      7 ranges supported
      Rates from 0.0 to 1250000.0 scans/sec
      16 channels ('ai0' - 'ai15')
      'Voltage' measurement type

   Analog output subsystem supports:
      -5.0 to +5.0 Volts,-10 to +10 Volts ranges
      Rates from 0.0 to 2857142.9 scans/sec
      2 channels ('ao0','ao1')
      'Voltage' measurement type

   Digital subsystem supports:
      39 channels ('port0/line0' - 'port2/line6')
      'InputOnly','OutputOnly','Bidirectional' measurement types

   Counter input subsystem supports:
      Rates from 0.1 to 80000000.0 scans/sec
      2 channels ('ctr0','ctr1')
      'EdgeCount' measurement type

   Counter output subsystem supports:
      Rates from 0.1 to 80000000.0 scans/sec
      2 channels ('ctr0','ctr1')
      'PulseGeneration' measurement type
```

Hardware Setup Description

This example uses a Portescap 20M020D1U 5V 18 Degree Unipolar Stepper Motor. The TTL signals produced by the digital I/O system are amplified by a Texas Instruments ULN2003AIN High Voltage High Current Darlington Transistor Array, as shown in this schematic:



Add Digital Output Only Channels

Create a session, and add 4 digital channels on port 0, lines 0-3. Set the measurement type to `OutputOnly`. These are connected to the four control lines for the stepper motor.

```
s = daq.createSession('ni');
addDigitalChannel(s,'Dev2','port0/line0:3','OutputOnly')
```

```
Warning: A channel that does not support clocked sampling was added to the
session. Clocked operations using startForeground and startBackground will be
disabled. Only on-demand operations using inputSingleScan and outputSingleScan
can be done.

ans =

  1×604 char array

Data acquisition session using National Instruments hardware:
   Clocked operations using startForeground and startBackground are disabled.
   Only on-demand operations using inputSingleScan and outputSingleScan can be done.
   Number of channels: 4
```

```
index Type Device   Channel   MeasurementType Range Name
----- ---- ------ ----------- --------------- ----- ----
1     dio  Dev2   port0/line0 OutputOnly      n/a
2     dio  Dev2   port0/line1 OutputOnly      n/a
3     dio  Dev2   port0/line2 OutputOnly      n/a
4     dio  Dev2   port0/line3 OutputOnly      n/a
```

Define Motor Steps

Refer to the Portescap motor wiring diagram describing the sequence of 4 bit patterns. Send this pattern sequentially to the motor to produce counterclockwise motion. Each step turns the motor 18 degrees. Each cycle of 4 steps turns the motor 72 degrees. Repeat this sequence five times to rotate the motor 360 degrees.

```
step1 = [1 0 1 0];
step2 = [1 0 0 1];
step3 = [0 1 0 1];
step4 = [0 1 1 0];
```

Rotate Motor

Use `outputSingleScan` to output the sequence to turn the motor 72 degrees counterclockwise.

```
outputSingleScan(s,step1);
outputSingleScan(s,step2);
outputSingleScan(s,step3);
outputSingleScan(s,step4);
```

Repeat sequence 50 times to rotate the motor 10 times counterclockwise.

```
for motorstep = 1:50
    outputSingleScan(s,step1);
    outputSingleScan(s,step2);
    outputSingleScan(s,step3);
    outputSingleScan(s,step4);
end
```

To turn the motor 72 degrees clockwise, reverse the order of the steps.

```
outputSingleScan(s,step4);
outputSingleScan(s,step3);
outputSingleScan(s,step2);
outputSingleScan(s,step1);
```

Turn Off All Outputs

After you use the motor, turn off all the lines to allow the motor to rotate freely.

```
outputSingleScan(s,[0 0 0 0]);
```

Generate Non-Clocked Digital Data

This example shows how to write data to two lines on an NI 625

Find devices connected to your system and find the ID for NI 6255.

```
d = daq.getDevices;
```

```
d =

Data acquisition devices:

index Vendor Device ID           Description
----- ------ --------- -----------------------------
1     ni     Dev1      National Instruments USB-6255
2     ni     Dev2      National Instruments USB-6363
```

Create a session and add two lines from port `0` on `Dev1`.

```
s = daq.createSession('ni');
addDigitalChannel(s,'Dev1','Port0/Line0:1','OutputOnly')
```

```
Data acquisition session using National Instruments hardware:
   Clocked operations using startForeground and startBackground are disabled.
   Only on-demand operations using inputSingleScan and outputSingleScan can be done.
   Number of channels: 2
      index Type Device   Channel   MeasurementType Range Name
      ----- ---- ------ ----------- --------------- ----- ----
      1     dio  Dev1   port0/line0 OutputOnly      n/a
      2     dio  Dev1   port0/line1 OutputOnly      n/a
```

Generate digital data.

```
outputSingleScan(s,[1,0])
```

Generate Signals Using Decimal Data Across Multiple Lines

This example shows how to convert decimal data and output to two lines on an NI 6255.

Find devices connected to your system and find the ID for NI 6255.

```
d = daq.getDevices;
```

```
d =

Data acquisition devices:

index Vendor Device ID          Description
----- ------ --------- ----------------------------
1     ni     Dev1      National Instruments USB-6255
2     ni     Dev2      National Instruments USB-6363
```

Create a session and add two lines from port `0` on `Dev1`.

```
s = daq.createSession('ni');
addDigitalChannel(s,'Dev1','Port0/Line0:1','OutputOnly')
```

```
Data acquisition session using National Instruments hardware:
   Clocked operations using startForeground and startBackground are disabled.
   Only on-demand operations using inputSingleScan and outputSingleScan can be done.
   Number of channels: 2
      index Type Device   Channel   MeasurementType Range Name
      ----- ---- ------ ----------- --------------- ----- ----
      1     dio  Dev1   port0/line0 OutputOnly      n/a
      2     dio  Dev1   port0/line1 OutputOnly      n/a
```

Convert the decimal number 2 to a binary vector and output the result

```
outputSingleScan(s,decimalToBinaryVector(2))
```

Generate and Acquire Data on Bidirectional Channels

This example shows how to use a bidirectional channel and read and write data using the same two lines on an NI 6255.

Find devices connected to your system and find the ID for NI 6255.

```
d = daq.getDevices;
```

```
d =

Data acquisition devices:

index Vendor Device ID          Description
----- ------ --------- -----------------------------
1     ni     Dev1      National Instruments USB-6255
2     ni     Dev2      National Instruments USB-6363
```

Create a session and add two lines from port 0 and 2 lines from port 1 on Dev1.

```
s = daq.createSession('ni');
addDigitalChannel(s,'Dev1','Port0/Line0:1','Bidirectional');
addDigitalChannel(s,'Dev1','Port1/Line0:1','Bidirectional')
```

```
Data acquisition session using National Instruments hardware:
   Clocked operations using startForeground and startBackground are disabled.
   Only on-demand operations using inputSingleScan and outputSingleScan can be done.
   Number of channels: 4
      index Type Device   Channel       MeasurementType     Range Name
      ----- ---- ------ ----------- ---------------------- ----- ----
      1     dio  Dev1   port0/line0 Bidirectional (Unknown) n/a
      2     dio  Dev1   port0/line1 Bidirectional (Unknown) n/a
      3     dio  Dev1   port1/line0 Bidirectional (Unknown) n/a
      4     dio  Dev1   port1/line1 Bidirectional (Unknown) n/a
```

Set the direction on all channels to output data.

```
for i = 1:4
    s.Channels(i).Direction = 'Output';
end
```

Generate digital data.

```
outputSingleScan(s,[1,0, 1, 0])
```

Change the direction on all channels to input data

```
for i = 1:4
    s.Channels(i).Direction = 'Input';
end
```

Acquire digital data.

```
inputSingleScan(s)
```

```
ans =

     1     0     1     0
```

You can also use the MATLAB `deal` function to change direction on all channels together.

```
[s.channels(:).Direction] = deal('Input');
```

Generate Signals on Both Analog and Digital Channels

This example shows how to generate signals when the session contains both analog and digital channels.

Find devices connected to your system and find the ID for NI 6255.

```
d = daq.getDevices;
```

```
d =

Data acquisition devices:

index Vendor Device ID          Description
----- ------ --------- -----------------------------
1     ni     Dev1      National Instruments USB-6255
2     ni     Dev2      National Instruments USB-6363
```

Create a session and add two digital lines from port `0` on `Dev1`.

```
s = daq.createSession('ni');
addDigitalChannel(s,'Dev1','Port0/Line0:1','OutputOnly')
```

```
Data acquisition session using National Instruments hardware:
   Clocked operations using startForeground and startBackground are disabled.
   Only on-demand operations using inputSingleScan and outputSingleScan can be done.
   Number of channels: 2
      index Type Device   Channel   MeasurementType Range Name
      ----- ---- ------ ----------- --------------- ----- ----
      1     dio  Dev1   port0/line0 OutputOnly      n/a
      2     dio  Dev1   port0/line1 OutputOnly      n/a
```

Add an analog output channel from Dev1.

```
addAnalogOutputChannel(s,'Dev1',0,'Voltage')
```

Output data on both the digital and analog channels.

```
outputSingleScan(s,[decimalToBinaryVector(2),1.23])
```

Output Digital Data Serially Using a Software Clock

This example shows how to generate signals serially using software clocks and the `timer` function.

Create a session and add two digital lines from port `0` on `Dev1` to output signals.

```
s = daq.createSession('ni');
addDigitalChannel(s,'Dev1','Port0/Line0:1','OutputOnly');
```

You will use `Port0/line0` as the output clock and `Port0/line1` as the serial data output.

Specify serial data to be transferred at 500 bits/sec.

```
serialData = [1 0 1 1 0 0 1 1];
```

Configure the software clock using a timer object, which has.

- A period of one micro second.
- `BusyMode` set to `queue` to accommodate clock stretching and start the timer.

```
t = timer('TimerFcn',{@sendData,s,serialData}, ...
    'Period', 0.001,...
    'ExecutionMode','fixedRate',...
    'BusyMode','queue');

start(t);
```

Define the `sendData` function and output data.

```
function sendData(obj, ~,s,serialData)
% Declare clock and bitNumber as persistent variables.
persistent clock;
persistent bitNumber;
% Persistent variables retain their values in memory between multiple calls

% to the function. Initialize the clock and the bit number for serial data
% transfer:

if isempty(clock)
    clock = 1;
end
% bitNumber is used to index into the serial data that needs to be sent.
if isempty(bitNumber)
    bitNumber = 0;
end
% Execute all calls to the function:
clock = ~clock;
```

```
% When the function reaches the end of the serial data, stop, reset the
% persistent variables to initial state and delete the timer:
if bitNumber > numel(serialData)
    stop(obj);
    % Reset variables for next run
    bitNumber = 0;
    clock = 1;
    disp('Stopping software timer. Command sent!')
    return
end
% Output the serial data and clock in your session:
outputSingleScan(s,[clock serialData(bitNumber)]);
end
```

Delete the timer after all the serial bits are output.

```
pause(.1)
delete(t);
```

Multichannel Audio

Multichannel Audio Input and Output

You can acquire and generate audio signals using one or more available channels of a supported audio device. You can also simultaneously operate channels on multiple supported audio devices. Currently Data Acquisition Toolbox supports audio channels for devices that work with DirectSound interface. Using the session-based interface, you can:

- Acquire and generate audio signals either in sequence or as separate operations.
- Acquire and generate signals in parallel where the signals may share the start time.
- Acquire the data in the background and filter or process the input data simultaneously. You can generate data immediately in response to the processed input data. In this case, both the acquisition and generation operations start and stop together.

You cannot read directly from or write directly to files using the multichannel audio feature. Use `audioread` and `audiowrite`.

Multichannel Audio Session Rate

The session rate in an audio session is the rate at which the session samples audio data. All channels in a session have the same session rate. The default session rate for an audio session is 44100 Hz. If you have multiple devices in the session, make sure that they can all operate at a common session rate. For standard sample rates, see StandardSampleRates property.

You can choose a value that is in between the standard values. The toolbox will quantize the set rate to the closest standard rate. If you choose a rate outside the ranges of the standard rates, the session may use it if the device you are using supports it. To use non-standard rates you must set UseStandardSampleRates to `false`. You cannot set the rate below the standard minimum rate or above the standard maximum rate.

Multichannel Audio Range

Data you acquire or generate using audio channels contains double-precision values. These values are normalized to the range of `-1` to `+1`. The session represents data acquired or generated in amplitude without units. The audio session Range property is read-only and set at `[-1 1]`.

Acquire Multichannel Audio Data

This example shows how to acquire audio data for seven seconds and plot the data.

Discover audio devices installed on your system and create a session for DirectSound devices.

```
d = daq.getDevices
s = daq.createSession('directsound')
```

Add two audio input channels for the microphone with id `Audio1`. Make sure that a microphone is plugged into the appropriate jack.

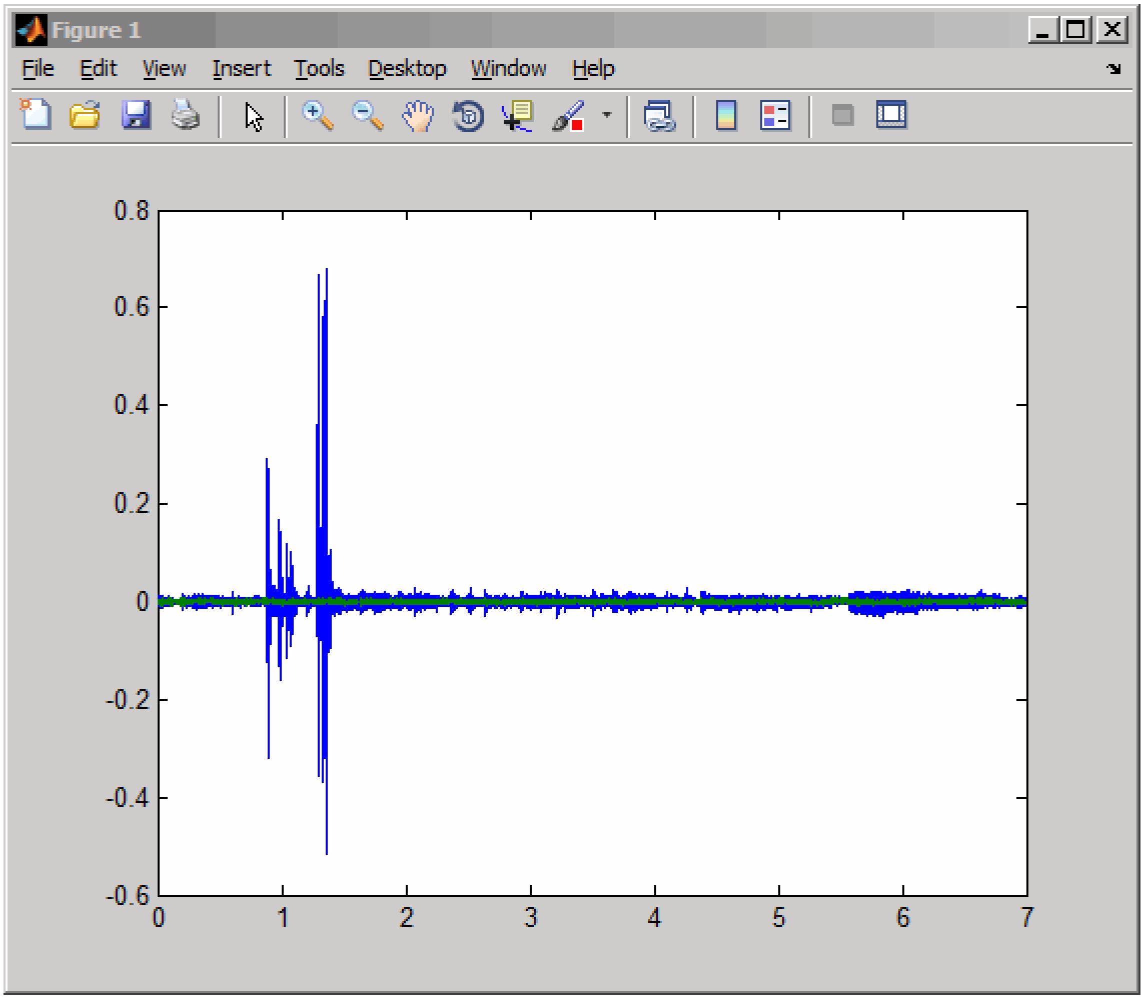
```
addAudioInputChannel(s,'Audio1', 1:2);
```

Set the session to run for 7 seconds and play an audio segment for the microphone to pick up.

```
s.DurationInSeconds = 7
```

Acquire data in the foreground and plot the data versus time.

```
[data,t] = startForeground(s);
plot(t, data);
```



Generate Audio Signals

This example shows how to generate audio signals using a Session. This example uses, but does not require, a 5.1 channel sound system.

In this example you generate an audio signal using the sound card on your computer using a 5.1 channel speaker setup. Before you begin, verify that your environment is set up so that you can generate data with your sound card. For more information refer to "Troubleshooting in Data Acquisition Toolbox".

Load Audio Signal

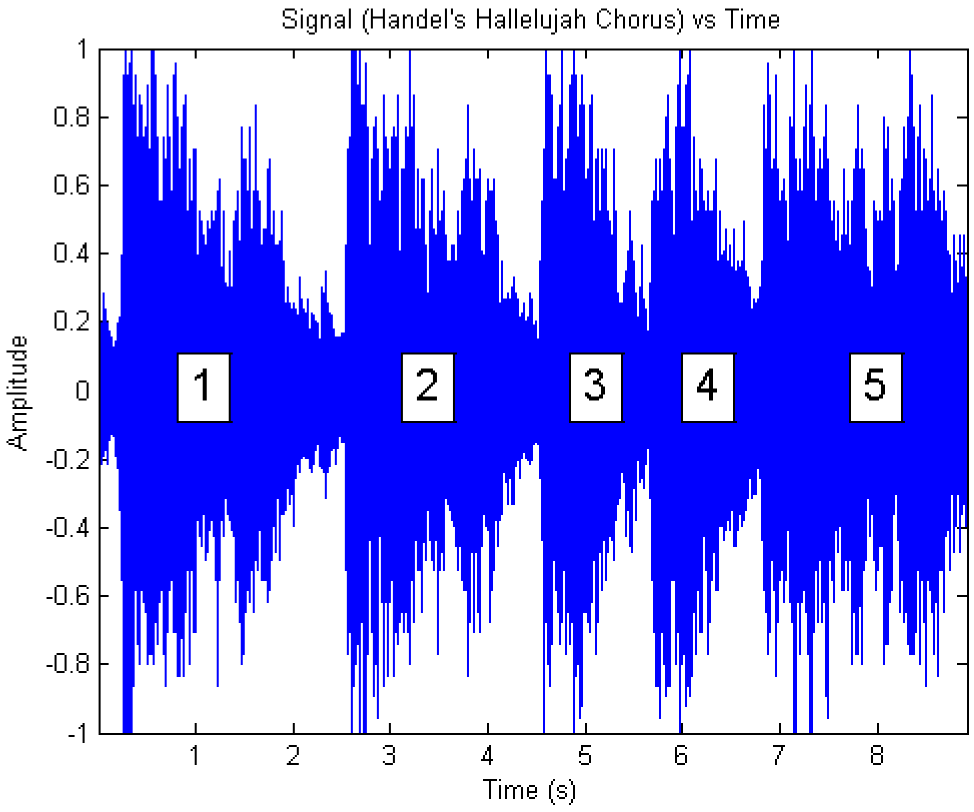
Load an audio file containing a sample of Handel's "Hallelujah Chorus."

```
load handel;
```

Plot Audio Signal

Plot data in order to identify five distinct segments. Each segment represents a "Hallelujah" in the chorus. The segments are annotated as 1 to 5.

```
ly = length(y);
lspan = 1:ly;
t = lspan/Fs;

hf = figure;
plot(t,y./max(y))
axis tight;
title('Signal (Handel''s Hallelujah Chorus) vs Time');
xlabel('Time (s)');
ylabel('Amplitude');

markers = struct('xpos',[0.2,0.4,0.55,0.65,0.8],'string',num2str([1:5]'));
for i = 1:5,
    annotation(hf,'textbox',[markers.xpos(i) 0.48 0.048 0.080],'String', markers.strin
end
```



View All Available Audio Devices

```
d = daq.getDevices
```

```
d =

Data acquisition devices:

index    Vendor     Device ID                                   Description
----- ----------- --------- ------------------------------------------------------------
1     directsound Audio0    DirectSound Primary Sound Capture Driver
2     directsound Audio1    DirectSound Microphone (High Definition Audio Device)
3     directsound Audio2    DirectSound HP 4120 Microphone (2- HP 4120)
```

```
4     directsound Audio3    DirectSound Microphone (Plantronics .Audio 400 DSP)
5     directsound Audio4    DirectSound Digital Audio (S/PDIF) (High Definition Audio D
6     directsound Audio5    DirectSound Primary Sound Driver
7     directsound Audio6    DirectSound Speakers (Plantronics .Audio 400 DSP)
8     directsound Audio7    DirectSound HP 4120 (2- HP 4120)
9     directsound Audio8    DirectSound Speakers (High Definition Audio Device):1
10    directsound Audio9    DirectSound Speakers (High Definition Audio Device):2
```

This example uses a 5.1 channel sound system with device ID 'Audio8'.

```
dev = d(9)
```

```
dev =

directsound: DirectSound Speakers (High Definition Audio Device):1 (Device ID: 'Audio8
   Audio output subsystem supports:
      -1.0 to +1.0  range
      Rates from 80.0 to 1000000.0 scans/sec
      8 channels ('1' - '8')
      'Audio' measurement type
```

Create an Audio Session

1. Create a session with `directsound` as the vendor and add an audio output channel to it.

```
s = daq.createSession('directsound');
noutchan = 6;
addAudioOutputChannel(s, dev.ID, 1:noutchan);
```

2. Update the session rate to match the audio sampling rate.

```
s.Rate = Fs
```

3. Queue the same waveform to all available channels/speakers. If additional, different voices are available, these should be queued to the appropriate channels.

```
queueOutputData(s,repmat(y,1,noutchan));
```

4. Start foreground generation. You should hear a sample of Handel's "Hallelujah Chorus." "Hallelujah" should be voiced five times, one for each segment depicted in the figure on all channels of the speaker system.

```
startForeground(s);
```

5. Close the figure.

```
close(hf);
```

```
s =

Data acquisition session using DirectSound hardware:
   No data queued.  Will run at 8192 scans/second.
   Number of channels: 6
      index Type Device Channel MeasurementType     Range     Name
      ----- ---- ------ ------- --------------- ------------ ----
      1     audo Audio8 1       Audio           -1.0 to +1.0
      2     audo Audio8 2       Audio           -1.0 to +1.0
      3     audo Audio8 3       Audio           -1.0 to +1.0
      4     audo Audio8 4       Audio           -1.0 to +1.0
      5     audo Audio8 5       Audio           -1.0 to +1.0
      6     audo Audio8 6       Audio           -1.0 to +1.0
```

Waveform Function Generation

Digilent Analog Discovery Devices

MATLAB supports the Digilent Analog Discovery design kit, a low-cost, portable USB DAQ device. The kit enables project-based learning for analog circuit design. For professors and course instructors, the kit comes with downloadable teaching materials, reference designs, and lab projects.

The Data Acquisition Toolbox Support Package for Digilent Analog Discovery hardware lets you perform the following tasks in MATLAB:

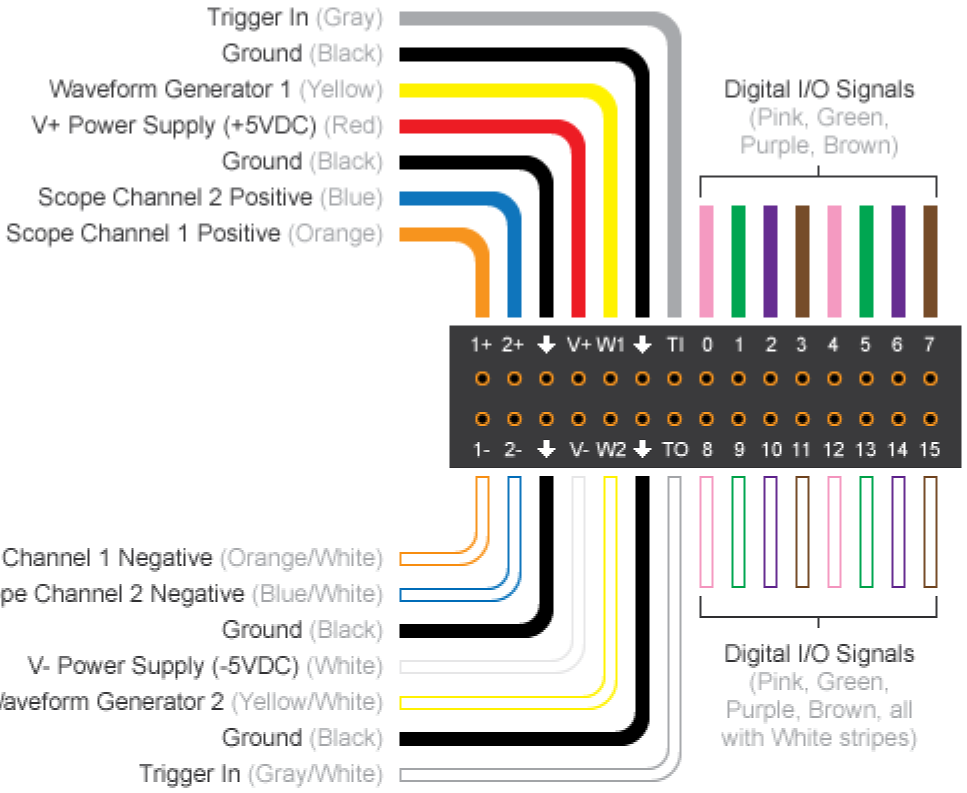
- Read data from oscilloscope channels.
- Control and generate data from waveform generators.
- Characterize ICs and measure behavior of the circuit and IC components.
- Configure the sampling rate of the Analog Discovery device.
- Trigger the start of your data acquisition.
- Find and display Digilent Analog Discovery device settings.

Use the Add-on Explorer to download required drivers. For more information see “Install Hardware Support Package for Vendor Support” on page 6-3.

For examples on Digilent data acquisition and generation see “Getting Started Acquiring Data with Digilent® Analog Discovery™” on page 7-19 and “Getting Started Generating Data with Digilent® Analog Discovery™” on page 7-29.

Digilent Waveform Function Generation Channels

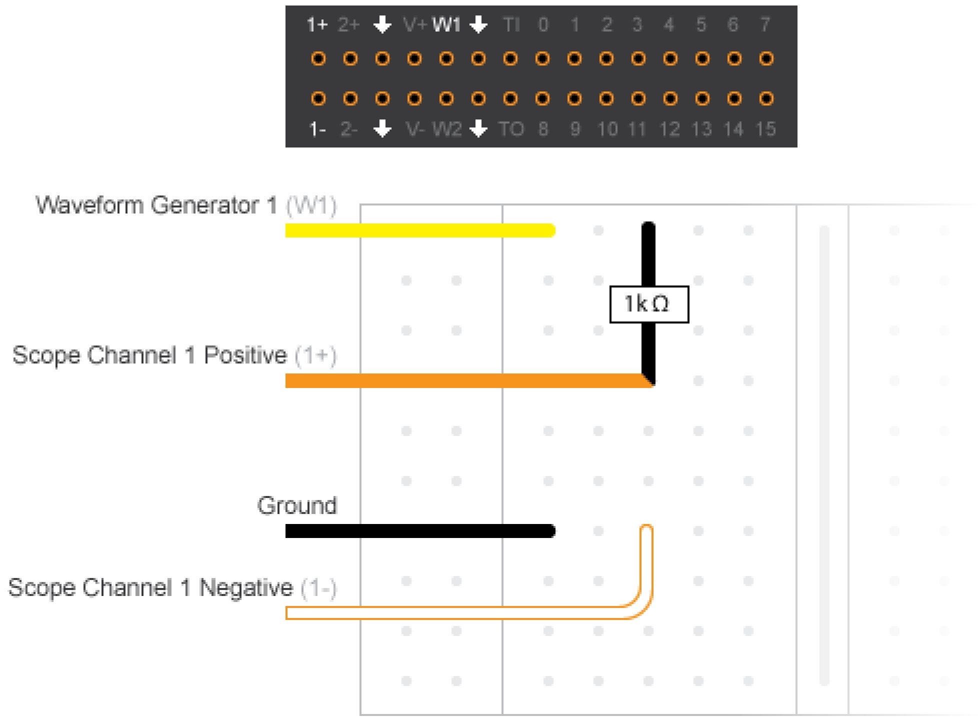
Waveform function generator channels on a Digilent device can generate both standard and arbitrary waveform functions. For more information on waveform types, see “Waveform Types” on page 12-6. This diagram shows you the pin configuration on a typical Digilent Analog Discovery device. The yellow and the yellow/white lines represent the waveform channels, marked by `W1` and `W2` on the device.


Trigger In (Gray)
Ground (Black)
Waveform Generator 1 (Yellow)
V+ Power Supply (+5VDC) (Red)
Ground (Black)
Scope Channel 2 Positive (Blue)
Scope Channel 1 Positive (Orange)
Digital I/O Signals (Pink, Green, Purple, Brown)
1+ 2+ ↓ V+ W1 ↓ TI 0 1 2 3 4 5 6 7
1- 2- ↓ V- W2 ↓ TO 8 9 10 11 12 13 14 15
Scope Channel 1 Negative (Orange/White)
Scope Channel 2 Negative (Blue/White)
Ground (Black)
V- Power Supply (-5VDC) (White)
Waveform Generator 2 (Yellow/White)
Ground (Black)
Trigger In (Gray/White)
Digital I/O Signals (Pink, Green, Purple, Brown, all with White stripes)

To test the Analog Discovery device create this connection to acquire the generated waveform, and use it with corresponding code:

- `1+` (scope channel 1 positive) to `WI` through a `1K` resistor.
- `1–` (scope channel 1 negative) `W2` to `GND`.

This diagram depicts these connections on a breadboard.


1+ 2+ V+ W1 TI 0 1 2 3 4 5 6 7
1- 2- V- W2 TO 8 9 10 11 12 13 14 15
Waveform Generator 1 (W1)
1kΩ
Scope Channel 1 Positive (1+)
Ground
Scope Channel 1 Negative (1-)

Unlike analog input channels, the function generation channels control their own waveform frequency. If your session contains both function generation channels and any other types of acquisition channels, the function generation channels will have their own frequency and all other channels will inherit the sessions frequency. If you have analog input channels in the session with function generation channels, the analog input channels start first and act as a trigger for function generation channels.

See Also

DutyCycle | Offset | Phase | gain

Related Examples

- "Generate a Standard Waveform Using Waveform Function Generation Channels" on page 12-9
- "Generate an Arbitrary Waveform Using Waveform Function Generation Channels" on page 12-11

More About

- "Waveform Types" on page 12-6

Waveform Types

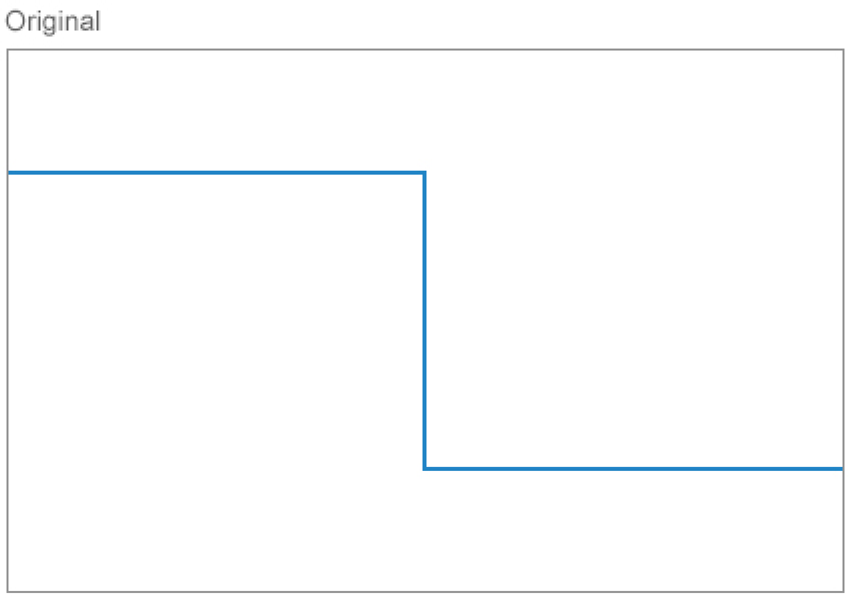
Your hardware can support generation of arbitrary waveforms or standard waveforms, or both. If your device supports standard waveforms, you can set the gain and offset to control the output. Standard waveform types include:

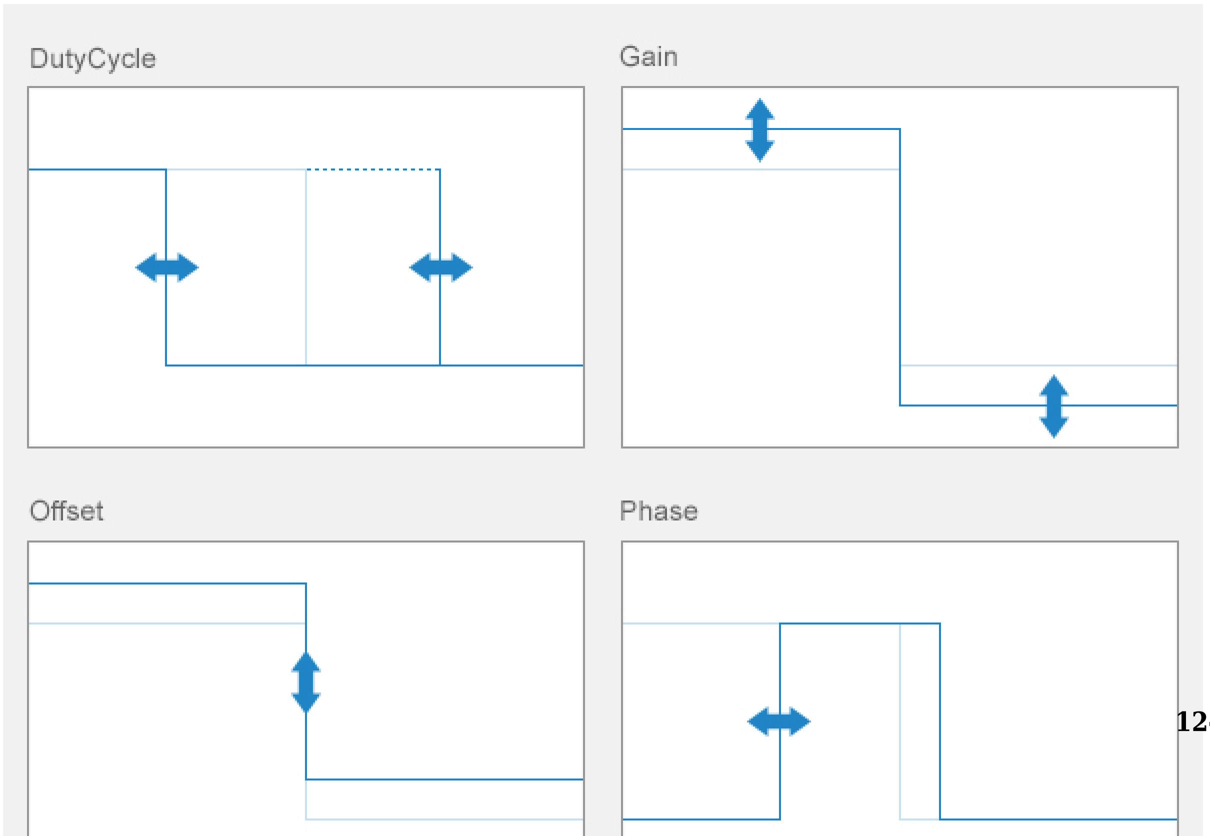
- Sine
- Square
- Triangle
- RampUp
- RampDown
- DC

You can control the behavior of different waveform types using the associated properties. The table shows you which properties work with the supported waveform types for Digilent devices.

	Frequency	Gain	Offset	Phase	DutyCycle
Sine	✓	✓	✓	✓	
Square	✓	✓	✓	✓	✓
Triangle	✓	✓	✓	✓	✓
RampUp	✓	✓	✓	✓	✓
RampDown	✓	✓	✓	✓	✓
DC			✓		
Arbitrary	✓				

This diagram illustrates how these properties affect a standard square waveform.


Original


DutyCycle
Gain
Offset
Phase

Standard waveforms cannot be clipped. You must keep Gain and Offset values within voltage range. You cannot change Gain and Offset of arbitrary waveforms.

See Also

DutyCycle | Offset | Phase | gain

Related Examples

- “Generate a Standard Waveform Using Waveform Function Generation Channels” on page 12-9
- “Generate an Arbitrary Waveform Using Waveform Function Generation Channels” on page 12-11

More About

- “Digilent Waveform Function Generation Channels” on page 12-3

Generate a Standard Waveform Using Waveform Function Generation Channels

This example shows how to use the function generation channel in a session to generate a sine waveform function, at 100kHz frequency. The signal's output voltage range is set to `-5.0` to `+5.0` volts

Create a Digilent acquisition session

```
s = daq.createSession('digilent');
```

Use `daq.getdevices` to discover available Digilent devices.

Create a waveform function generation channel with a `Sine` `WaveformType`.

```
fgenCh = addFunctionGeneratorChannel(s, 'AD1', 1, 'Sine')
```

```
fgenCh =

Data acquisition sine waveform generator '1' on device 'AD1':

              Phase: 0
              Range: -5.0 to +5.0 Volts
     TerminalConfig: SingleEnded
               Gain: 1
             Offset: 0
          Frequency: 4096
       WaveformType: Sine
     FrequencyLimit: [0.0 25000000.0]
               Name: ''
                 ID: '1'
             Device: [1x1 daq.di.DeviceInfo]
    MeasurementType: 'Voltage'
```

Set the channel's amplitude to 5v using the `Gain` property and the channel frequency to `100KHz`.

```
fgenCh.Gain = 5;
fgenCh.Frequency = 100e3
```

```
fgenCh

Data acquisition sine waveform generator '1' on device 'AD1':
```

```
                 Phase: 0
                 Range: -5.0 to +5.0 Volts
        TerminalConfig: SingleEnded
                  Gain: 5
                Offset: 0
             Frequency: 100000
          WaveformType: Sine
        FrequencyLimit: [0.0 25000000.0]
                  Name: ''
                    ID: '1'
                Device: [1x1 daq.di.DeviceInfo]
       MeasurementType: 'Voltage'
```

Specify the session to run for 5 seconds and start the generation.

```
s.DurationInSeconds = 5;
startForeground(s);
```

Generate an Arbitrary Waveform Using Waveform Function Generation Channels

This example shows how to use the function generation channel in a session to generate an arbitrary waveform function, at 100kHz frequency. The signal's output voltage range is set to `-5.0` to `+5.0` volts

Create a Digilent acquisition session

```
s = daq.createSession('digilent');
```

Use `daq.getdevices` to discover available Digilent devices.

Create a waveform function generation channel with a `Arbitrary` `WaveformType`.

```
fgenCh = addFunctionGeneratorChannel(s, 'AD1', 1, 'Arbitrary')
```

```
fgenCh =

Data acquisition arbirtray waveform generator '1' on device 'AD1':

              Phase: 0
              Range: -5.0 to +5.0 Volts
     TerminalConfig: SingleEnded
               Gain: 1
             Offset: 0
          Frequency: 4096
       WaveformType: Sine
     FrequencyLimit: [0.0 25000000.0]
               Name: ''
                 ID: '1'
             Device: [1x1 daq.di.DeviceInfo]
    MeasurementType: 'Voltage'
```

Set the buffer size to `4096` and set the channel to generate a waveform repeatedly from the contents of the buffer. The channel outputs for a fixed number of times over the space of the buffer.

```
buffersize = 4096;
len = buffersize + 1;

f0 = 1;
f1 = 1 * f0;
```

```
f3 = 3 * f0;
f5 = 5 * f0;

waveform  = sin(linspace(0, 2*pi*f1, len)) + ...
            sin(linspace(0, 2*pi*f3, len)) + ...
            sin(linspace(0, 2*pi*f5, len));

waveform = 5*waveform./max(abs(waveform));
waveform(end) = [];
```

Set the `WaveformData` of the channel to the `waveform`.

```
fgenCh.WaveformData = waveform;
```

Set the frequency of the channel to `100 KHz`.

```
fgenCh.Frequency = 100e3;
```

Set the session duration to 5 seconds and generate continuous data.

```
s.DurationInSeconds = 5;
startForeground(s);
```

Triggers and Clocks

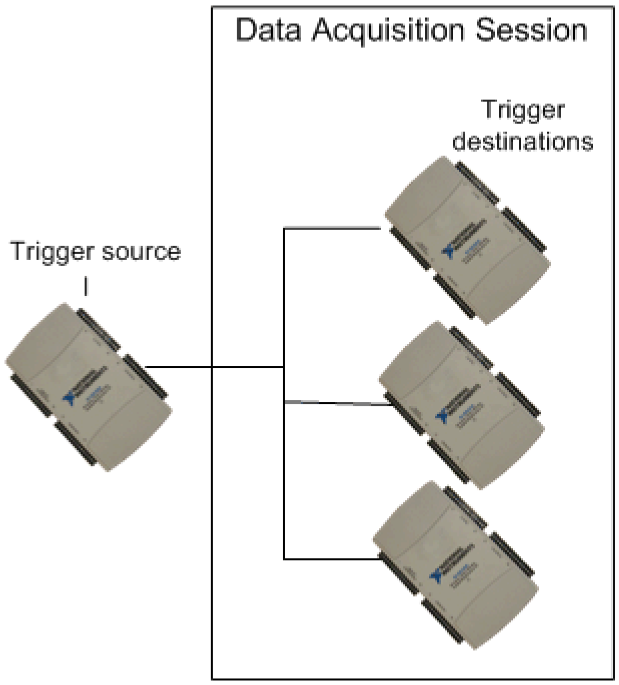
Trigger Connections

In this section...
“When to Use Triggers” on page 13-2
“External Triggering” on page 13-3

When to Use Triggers

Use triggers to simultaneously start all devices in the session. You connect a trigger source to a trigger destination, A trigger source can be either external, where the trigger comes from a source outside a session, or on a device and terminal pair within a session. Trigger destination devices can be external, where the signals are received outside the session, or devices within the session. To understand source and destination devices, see “Source and Destination Devices” on page 14-3.

Note You can have multiple destinations for your trigger, but only one source.



Note You cannot use trigger and clock connections with audio channels.

External Triggering

You can configure devices in a session to receive an external trigger. To use an external trigger source, your connection parameters must correctly specify the exact device and terminal pairs to which the external source is connected. Two circumstances of externally clocked and triggered synchronization are:

- An external hardware event that controls the operation of one or more devices in a session object. For example, opening and closing a switch starts a background acquisition on a session.
- An external hardware event synchronizes multiple devices in a session. For example, opening and closing of a switch starts a background operation across multiple devices or CompactDAQ chassis in a session.

See Also

Related Examples

- “Multiple-Device Synchronization Using USB or PXI Devices” on page 14-9
- “Multiple-Chassis Synchronization with CompactDAQ Devices” on page 14-16
- “Acquire Voltage Data Using a Digital Trigger” on page 13-5

More About

- “Synchronization” on page 14-2

Acquire Voltage Data Using a Digital Trigger

This example shows how to use a falling edge digital trigger, which occurs when a switch closes on an external source. The trigger is connected to terminal `PFI0` on device `Dev1` and starts acquiring sensor voltage data.

Create a data acquisition session and add channels.

```
 s = daq.createSession('ni');
```

Add one voltage input channel from NI USB-6211 with device ID `'Dev1'`.

```
addAnalogInputChannel(s,'Dev1',0,'Voltage');
```

Connect the switch to terminal `'PFI0'` on NI USB-6211. The trigger comes from the switch, which is an external source.

```
addTriggerConnection(s,'External','Dev1/PFI0','StartTrigger')
```

```
ans =


Start Trigger is provided externally and will be received by 'Dev1' at terminal 'PFI0'.

     TriggerType: 'Digital'
TriggerCondition: RisingEdge
          Source: 'External'
     Destination: 'Dev1/PFI0'
            Type: StartTrigger
```

Set `TriggerCondition` property to `'FallingEdge'`.

```
c = s.Connections(1);
c.TriggerCondition = 'FallingEdge';
```

Acquire data and store it in `dataIn`. The session waits for the trigger to occur, and starts acquiring data when the switch closes.

```
dataIn = startForeground(s);
```

See Also

Related Examples

- “Multiple-Device Synchronization Using USB or PXI Devices” on page 14-9
- “Multiple-Chassis Synchronization with CompactDAQ Devices” on page 14-16

More About

- “Synchronization” on page 14-2
- “Trigger Connections” on page 13-2

Clock Connections

In this section...

When to Use Clocks

Use clocks to synchronize operations on all connected devices in the session. You connect a clock source to a clock destination. A clock source can be either external, where the clock signal comes from a source outside a session, or on a device and terminal pair within a session. Destination devices can be external, where the signals are received outside the session, or devices within the session. To understand source and destination devices, see “Source and Destination Devices” on page 14-3.

Note You cannot use trigger and clock connections with audio channels.

Import Scan Clock from External Source

To import a scan clock from an external source, you must connect the external clock to a terminal and device pair on a device in your session. Two circumstances of externally clocked synchronization include:

- Synchronizing operations on all devices within a session by sharing the clock on a device within the session or an external clock
- Synchronizing operations on all devices within a session and some external devices, by sharing an external clock

Note Importing an external clock does not automatically set the rate of your session. Manually set the session’s rate to match the expected external clock frequency.

Export Scan Clock to External System

This example shows how to add a scan clock to a device and output the clock to a device outside your session, which is connected to an oscilloscope. The scan clock controls the operations on the external device.

Create a session and add one voltage input channel from NI USB-6211 with device ID `'Dev1'`.

```
s = daq.createSession('ni');
addAnalogInputChannel(s,'Dev1', 0, 'Voltage');
```

Add an external clock to terminal `'PFI6'` on `'Dev1'` and connect it to an external destination.

```
addClockConnection(s,'Dev1/PFI6','External','ScanClock')
```

```
ans = 


Scan Clock for 'Dev1' will available at terminal 'PFI6' for external use.

       Source: 'Dev1/PFI6'
  Destination: 'External'
         Type: ScanClock
```

Acquire data and store it in `dataIn`.

```
dataIn = startForeground(s);
```

See Also

Related Examples

- “Multiple-Device Synchronization Using USB or PXI Devices” on page 14-9
- “Multiple-Chassis Synchronization with CompactDAQ Devices” on page 14-16

More About

- “Synchronization” on page 14-2

Session-Based Synchronization

Synchronization

Synchronization of data acquisition operations between multiple channels or devices has two aspects:

- Start trigger: The signal to initiate all operations
- Scan clock: The timing for repeated generation or acquisition of signals at a clocked rate

Synchronization can involve the coordination of triggering, clocking, or both. To synchronize the start of operations on multiple channels or devices, they must use a shared start trigger. To synchronize the clocked scanning operations on multiple channels or devices, they must use a shared scan clock.

The following definitions summarize some concepts of synchronization:

Type of Synchronization	Description
Start trigger synchronization	Channels or devices are configured to simultaneously start their operations from a shared start trigger.
Scan clock synchronization	Channels or devices use a shared scan clock to generate or measure signals.
Perfect synchronization	Channels or devices use both a shared start trigger and a shared scan clock. This does not imply a specific skew or latency performance between devices or between channels on a device.
Automatic synchronization	The default start trigger synchronization and scan clock synchronization supported by a data acquisition session, the driver, and the hardware. This is the extent of synchronization provided by a data acquisition session without any explicit synchronization configuration. When a session starts, it sends a start trigger signal to all connected channels in the session. The driver and device might support synchronization from that moment forward. For example, in some devices all channels use the same internal scan clock and a shared start trigger, so they are automatically synchronized without further configuration of the session.

Shared Triggers and Shared Scan Clocks

Typical data acquisition devices provide synchronization between their channels of the same subsystem. For example, all the analog input channels on one card use a shared scan clock. A data acquisition session can configure start trigger and scan clock connections for wider synchronization needs. Use shared start triggers and shared scan clocks to synchronize data between:

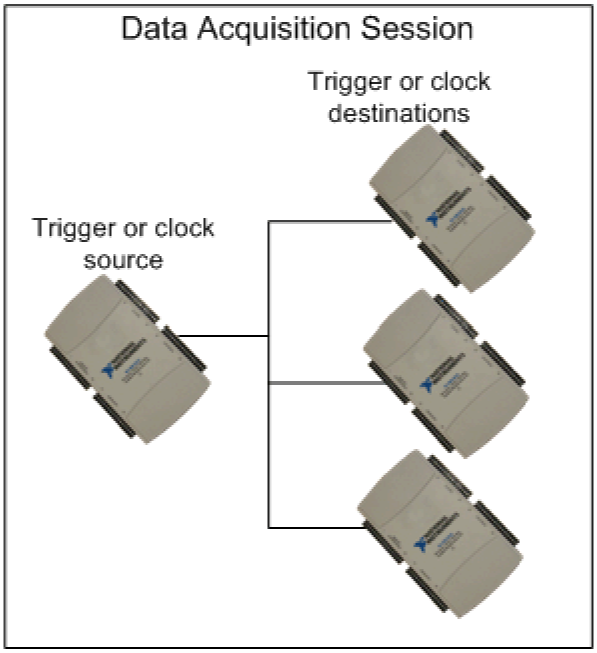
- Multiple subsystems in a device (analog input, analog output, counter input, etc.)
- Multiple devices
- Multiple CompactDAQ or PXI chassis

Note Counter output channels run independently and are unaffected by synchronization connections.

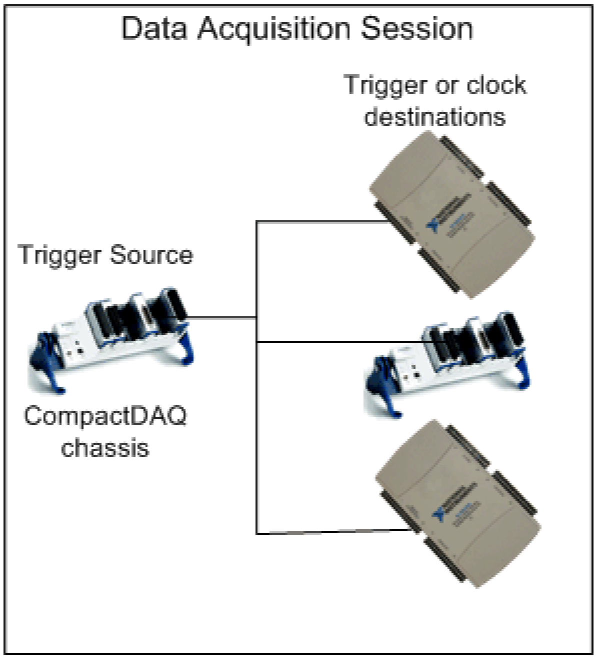
Source and Destination Devices

You can share start triggers and scan clock connections to synchronize operations within a session. Synchronization connections can be:

- Devices in a session connected to a start trigger or scan clock source on another device in the session



- Devices and chassis in a session connected to a start trigger or scan clock source on another device in the session



A source device and terminal pair generates the synchronization signal and is connected to the destination device and terminal pairs. You must physically connect the source and destination terminals, unless they are internally connected. Check your device specifications for more information. Synchronization connections are added from the source device to one or more destination devices.

- The source device provides the start trigger or scan clock signals.
- The destination device receives a start trigger or scan clock signal.

For example, if you determine that a terminal on `Dev1` will provide a start trigger and a terminal on `Dev2` will receive that trigger, then `Dev1` becomes your source device and `Dev2` your destination device. You can have multiple destinations for your trigger and clock connections, but only one source.

Use `addTriggerConnection` to add start trigger connections, and `addClockConnection` to add a scan clock connection to your session.

Automatic Synchronization

In most cases, a session automatically starts all its devices at the same time when you start an operation. You must configure them to start synchronously when devices are not on a single chassis and do not share a clock. If you have not configured synchronization on such devices, the start operation reduces the latency between devices, running them very close together to achieve near-simultaneous signals. However, devices are automatically and perfectly synchronized in the session if they are:

- Subsystems on a single device in the session. This synchronizes your analog input, analog output, and counter input channels.

  Note Counter output channels run independently and are unaffected by synchronization connections.

- Modules on a single CompactDAQ chassis in the session.
- PXI modules synchronized with a reference clock on a PXI chassis. For perfect synchronization, you must share a trigger as well. See “Acquire Synchronized Data Using PXI Devices” on page 14-12 for more information.

Synchronization Scenarios

You must employ different techniques for synchronization, depending on the configurations of your channels, devices, and chassis. The following sections describe these different scenarios.

Multiple Channels on the Same Device or Module

In this topic, hardware that performs the signal conversion when not plugged into a chassis is referred to as a *device*; this includes USB devices. When the conversion hardware is a card plugged into a chassis, it is usually referred to as a *module*.

Data Acquisition Toolbox session software is based on the assumption that all channels of the same acquisition device or module use the same internal scan clock and start trigger. As such, these channels meet the requirements for perfect synchronization. For most vendors, this includes digital channels, analog channels, and counter input channels, but does not include counter output channels.

The following topics illustrate this scenario, providing automatic synchronization between multiple channels.

- “Acquire Data from Multiple Channels using MCC Devices”
- “Acquiring and Generating Data at the Same Time with Digilent® Analog Discovery™”

Exceptions: Some devices do not support setting the source of the start trigger or do not internally route start trigger signals between subsystems. These include National Instruments myDAQ and USB-6002. In such devices, only channels of the same subsystem support start trigger synchronization by default.

Multiple Modules in the Same CompactDAQ Chassis

Modules in the same CompactDAQ chassis use the chassis scan clock and start trigger. The Data Acquisition Toolbox session interface configures the chassis scan clock rate and issues the start trigger signal. The chassis in turn provides synchronized signals to its modules.

The following examples illustrate this scenario, providing synchronization between multiple modules in the same chassis without external connections or extra programming.

- “Acquire Data and Generate Signals at the Same Time”
- “Count Pulses on a Digital Signal Using NI Devices”
- “Measure Frequency Using NI Devices”
- “Measure Pulse Width Using NI Devices”

Exceptions: Some CompactDAQ modules have their own onboard clocks, for example, DSA modules.

Multiple Modules in the Same PXI Chassis

Modules in a PXI chassis share a common scan clock, but a Data Acquisition Toolbox session does not synchronize the start trigger for multiple modules in the chassis by default. The start triggers of multiple DSA modules can be synchronized using the AutoSyncDSA property, while other PXI modules require an external trigger connection for start trigger synchronization.

The following topics illustrate these scenarios, showing how to synchronize start triggers on multiple modules.

- “Synchronize DSA Devices” on page 14-18
- “Synchronize DSA PXI Devices Using AutoSyncDSA” on page 14-12
- “Acquire Synchronized Data Using PXI Devices” on page 14-12

Multiple Devices Without Chassis or in Different Chassis

This scenario represents multiple devices or modules in their most independent configuration. The configuration could be multiple USB devices, for example, or modules in separate chassis. Neither the start triggers nor the scan clocks of these devices are synchronized by default.

The following topics illustrates these scenarios, showing how to synchronize start triggers and scan clocks on multiple devices without chassis or in different chassis, by way of an external connection.

- "Acquire Synchronized Data Using USB Devices" on page 14-9
- "Multiple-Chassis Synchronization with CompactDAQ Devices" on page 14-16
- "Synchronize Counter Outputs from Multiple Devices" on page 14-11
- "Acquire Data from Two Devices at Different Rates"

See Also

More About

- "Multiple-Device Synchronization Using USB or PXI Devices" on page 14-9
- "Synchronize DSA Devices" on page 14-18

Multiple-Device Synchronization Using USB or PXI Devices

You can synchronize multiple devices in a session using a shared scan clock and shared start trigger. You can synchronize devices using either PFI or RTSI lines.

Requirement You must register your RTSI cable using the National Instruments Measurement & Automation Explorer.

Acquire Synchronized Data Using USB Devices

This example shows how to acquire synchronized voltage data from multiple devices using a shared start trigger and a shared scan clock. Analog input channels on all three devices are connected to the same function generator.

Create a data acquisition session and add one voltage input channel from each device:

- NI USB-6211 with device ID `'Dev1'`
- NI USB 6218 with device ID `'Dev2'`
- NI USB 6255 with device ID `'Dev3'`

```
s = daq.createSession('ni');
addAnalogInputChannel(s,'Dev1',0,'Voltage');
addAnalogInputChannel(s,'Dev2',0,'Voltage');
addAnalogInputChannel(s,'Dev3',0,'Voltage');
```

Choose terminal `'PFI4'` on `'Dev1'` as the start trigger source. Connect the trigger source to terminal `'PFI0'` on `'Dev2'` and to `'PFI0'` on `'Dev3'`, which are the destination devices.

```
addTriggerConnection(s,'Dev1/PFI4','Dev2/PFI0','StartTrigger');
addTriggerConnection(s,'Dev1/PFI4','Dev3/PFI0','StartTrigger');
```

Choose terminal `'PFI5'` on `'Dev1'` as the scan clock source. Connect it to `'PFI1'` on `'Dev2'`, and to `'PFI1'` on `'Dev3'`.

```
s.addClockConnection('Dev1/PFI5','Dev2/PFI1','ScanClock');
s.addClockConnection('Dev1/PFI5','Dev3/PFI1','ScanClock')
```

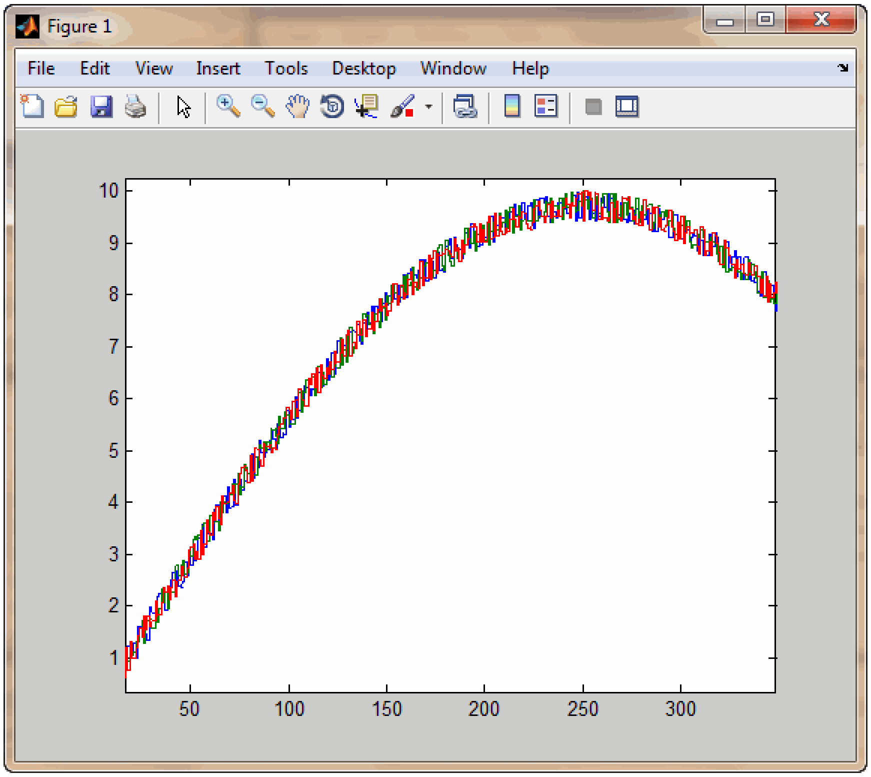
```
ans =
Start Trigger is provided by 'Dev1' at 'PFI4' and will be received by:
         'Dev2' at terminal 'PFI0'
         'Dev3' at terminal 'PFI0'
Scan Clock is provided by 'Dev1' at 'PFI5' and will be received by:
         'Dev2' at terminal 'PFI1'
         'Dev3' at terminal 'PFI1'

   index     Type      Source   Destination
   ----- ------------ --------- -----------
   1     StartTrigger Dev1/PFI4 Dev2/PFI0
   2     StartTrigger Dev1/PFI4 Dev3/PFI0
   3     ScanClock    Dev1/PFI5 Dev2/PFI1
   4     ScanClock    Dev1/PFI5 Dev3/PFI1
```

Acquire data and assign it to `dataIn`.

```
dataIn = startForeground(s);
```

Plot the data.

```
plot(dataIn)
```



All channels are connected to the same function generator, and therefore display overlapping signals, indicating synchronization.

Synchronize Counter Outputs from Multiple Devices

This example shows how to synchronize the start trigger of counter output operations from two channels on different devices.

```
s = daq.createSession('ni');
addCounterOutputChannel(s,'Dev1','ctr0','PulseGeneration');
addCounterOutputChannel(s,'Dev2','ctr0','PulseGeneration');
addTriggerConnection(s,'Dev1/PFI0','Dev2/PFI0','StartTrigger');
startForeground(s)
```

This example uses two USB or PCI devices, but could be modified for channels across CompactDAQ or PXI chassis. If you have counter output CompactDAQ modules in the same chassis, it is not necessary to call `addTriggerConnection`; but it is required for multiple modules in the same PXI chassis.

Synchronize DSA PXI Devices Using AutoSyncDSA

This example shows how to acquire synchronized data from two Dynamic Signal Analyzer (DSA) PXI devices, NI PXI-4462 and NI PXI-4461, using the AutoSyncDSA property.

Create an acquisition session and add one voltage analog input channel from each of the two PXI devices

```
s = daq.createSession('ni');
addAnalogInputChannel(s,'PXI1Slot2',0,'Voltage');
addAnalogInputChannel(s,'PXI1Slot3',0,'Voltage');
```

Acquire data in the foreground without synchronizing the channels:

```
[data,time] = startForeground(s);
plot(time,data)
```

The data returned is not synchronized.

Synchronize the two channels using the `AutoSyncDSA` property:

```
s.AutoSyncDSA = true;
```

Acquire data in the foreground and plot it:

```
[data,time] = startForeground(s);
plot(time, data)
```

The data is now synchronized.

Acquire Synchronized Data Using PXI Devices

This example shows how to acquire voltage data from two PXI devices on the same chassis, using a shared start trigger to synchronize operations within your session. PXI devices have a shared reference clock that automatically synchronizes scan clocking. You need to add only start trigger connections to synchronize operations in your session with PXI devices. Analog input channels on all devices are connected to the same function generator.

Create a data acquisition session and add one voltage input channel from each NI-PXI 4461 device with IDs `'PXI1Slot2'` and `'PXI1Slot3'`.

```
s = daq.createSession('ni');
addAnalogInputChannel(s,'PXI1Slot2',0,'Voltage');
addAnalogInputChannel(s,'PXI1Slot3',0,'Voltage');
```

Add a start trigger connection to terminal `'PXI_Trig0'` on `'PXI1Slot2'` and connect it to terminal `'PXI_Trig0'` on `'PXI1Slot3'`. PXI cards are connected through the backplane, so you do not have to wire them physically.

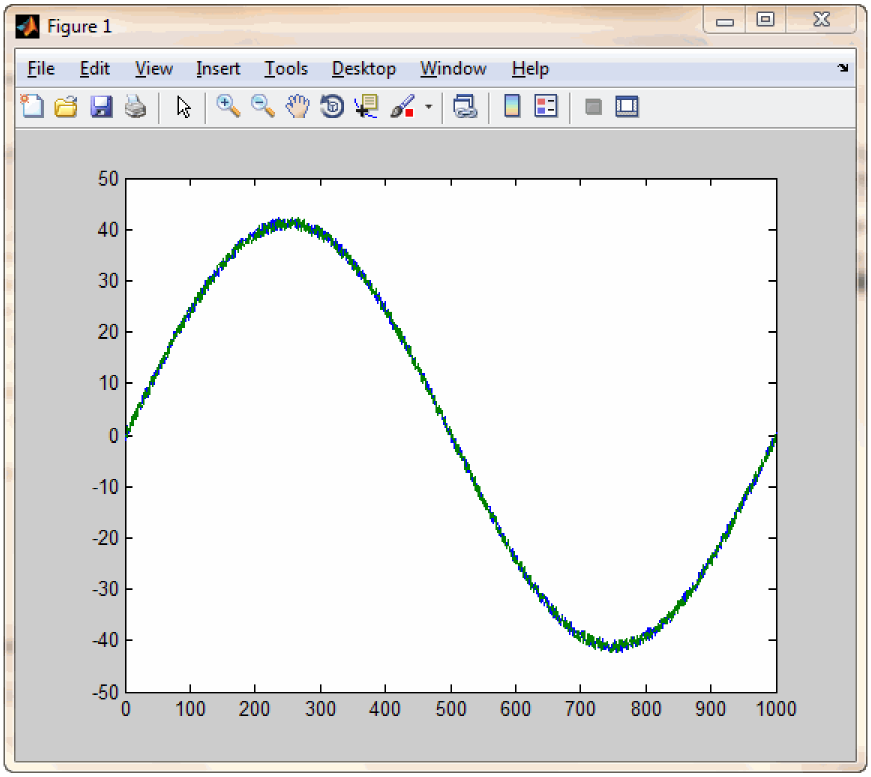
```
addTriggerConnection(s,'PXI1Slot2/PXI_Trig0','PXI1Slot3/PXI_Trig0','StartTrigger');
```

Acquire data and assign it to `dataIn`.

```
dataIn = startForeground(s);
```

Plot the data.

```
plot(dataIn)
```



All channels are connected to the same function generator and have a shared reference clock. The signals overlap, indicating perfect synchronization.

See Also

More About

- “Multiple-Chassis Synchronization with CompactDAQ Devices” on page 14-16
- “Generate Pulse Data on a Counter Channel” on page 9-7

Synchronize with PFI on CompactDAQ Chassis Without Terminals

This example shows how to use the external trigger and external clock functionality on a CompactDAQ chassis without PFI terminals, by using the PFI terminals on digital I/O CompactDAQ modules.

Some CompactDAQ chassis (e.g., NI 9174 and 9172) do not support built-in triggers, because they do not have external BNC PFI connectors on the chassis itself. However, the PFI pins for these chassis can be accessed through a digital module such as the NI 9402.

```
%% Setup
% cDAQ 9174
% cDAQ2Mod3 is 9402
% cDAQ2Mod4 is 9201

%% For the start trigger
s = daq.createSession('ni');
addAnalogInputChannel(s,'cDAQ2Mod4','ai0','Voltage');
addTriggerConnection(s,'External','cDAQ2Mod3/PFI0','StartTrigger');
[data,timestamps] = startForeground(s);
plot(timestamps,data);

%% For the external scan clock
% Function generator provides 100 kHz clock to PFI1 terminal on NI 9402
s = daq.createSession('ni');
addDigitalChannel(s,'cDAQ2Mod3','Port0/Line2','InputOnly');
addClockConnection(s,'External','cDAQ2Mod3/PFI1','ScanClock');
s.Rate = 100E+3;
[data,timestamps] = startForeground(s);
plot(timestamps,data);
```

See Also

Multiple-Chassis Synchronization with CompactDAQ Devices

This example shows how to acquire voltage data from two devices, each on a separate CompactDAQ chassis, using a shared trigger and clock to synchronize operations within your session.

You can synchronize multiple CompactDAQ chassis in a session using one chassis to provide clocking and triggering for all chassis in the session. Clock and trigger sources are attached to terminals on the chassis, itself. All modules on the chassis as well as other connected devices, are synchronized using these signals.

Create a data acquisition session and add channels. Add one voltage input channel each from the two NI 9201 devices with IDs `'cDAQ1Mod1'` and `'cDAQ2Mod1'`.

```
s = daq.createSession('ni');
addAnalogInputChannel(s,'cDAQ1Mod1',0,'Voltage');
addAnalogInputChannel(s,'cDAQ2Mod1',0,'Voltage');
```

Choose terminal `'PFI0'` on `'cDAQ1'` as your trigger source and connect it to terminal `'PFI0'` on `'cDAQ2'`. Make sure the wiring on the hardware runs between these two terminals. Note that you are using the chassis and terminal pair here, not device and terminal pair.

```
addTriggerConnection(s,'cDAQ1/PFI0','cDAQ2/PFI0','StartTrigger');
```

Choose terminal `'PFI1'` on `'cDAQ1'` as your clock source and connect it to terminal `'PFI1'` on `'cDAQ2'`. Make sure the wiring on the hardware runs between these terminals.

```
addClockConnection(s,'cDAQ1/PFI1','cDAQ2/PFI1','ScanClock');
```

Acquire data and store it in `dataIn`.

```
dataIn = startForeground(s);
```

See Also

More About

- “Synchronize Counter Outputs from Multiple Devices” on page 14-11

Synchronize DSA Devices

In this section...

The Digital Signal Analyzer (DSA) product family is designed to make highly accurate audio frequency measurements. You can synchronize other PCI and PXI product families using "Trigger Connections" on page 13-2 and "Clock Connections" on page 13-7. To synchronize PXI and PCI family of DSA devices you need to use a sample clock with time-based synchronization or a reference clock time based synchronization. The AutoSyncDSA property allows you to automatically enable both homogeneous and heterogeneous synchronization between PCI and PXI device families. AutoSyncDSA property automatically configures all the necessary clocks, triggers, and sync pulses needed to synchronize DSA devices in your session.

PXI DSA Devices

PXI devices are synchronized using the PXI chassis backplane, which includes timing and triggering buses. You can automatically synchronize these device series both homogeneously (within the same series) and heterogeneously (across separate series) in the same session.

- PXI/e 446x series
- PXI/e 449x series
- PXI 447x series

Hardware Restrictions

Before you synchronize, ensure that your device combinations adhere to these hardware restrictions:

PXI/e 446x and 449x Series

Chassis restriction

You can synchronize these series using either a PXI or a PXIe chassis. Make sure all your modules are on the same chassis.

Slot placement restriction

You can use any slot on the chassis that supports your module.

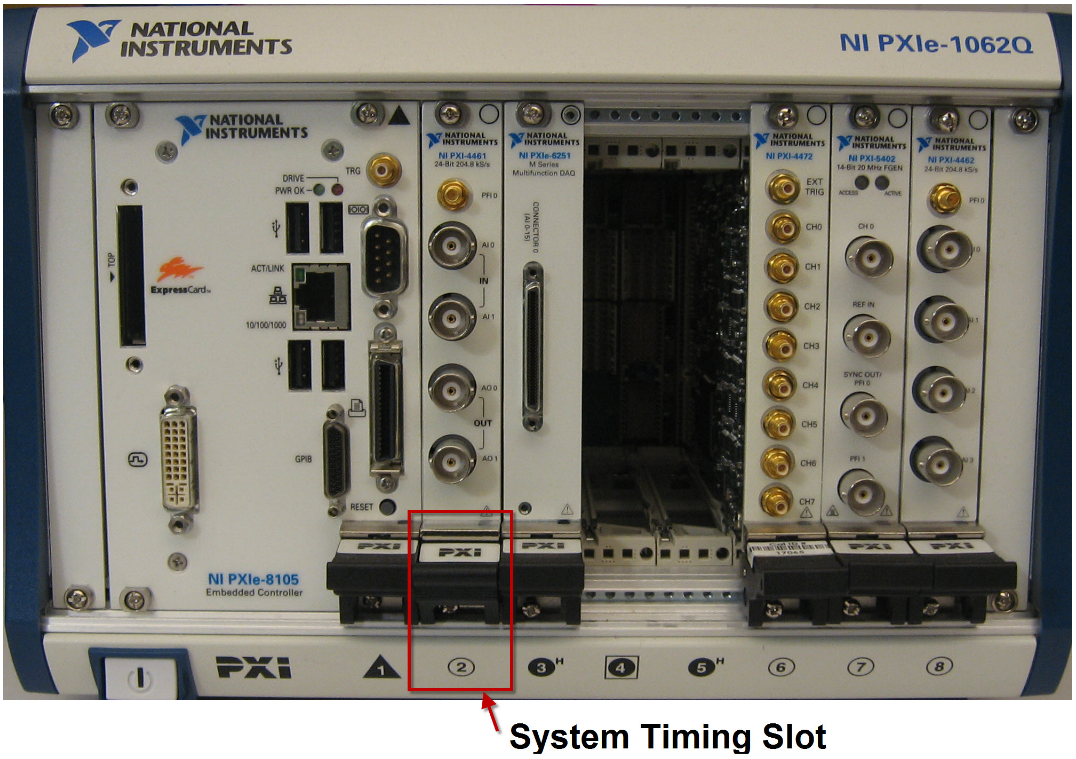
PXI 447x Series

Chassis restriction

You can synchronize this series both homogeneously and heterogeneously only on a PXI chassis. You can use them on a PXIe chassis to acquire unsynchronized data.

Slot placement restriction

On the PXI chassis, only the system timing slot can drive the trigger bus. Refer to your device manual to find the system timing slot. This image shows the system timing slot on a PXIe 1062Q chassis.



- Homogeneous synchronization: You can synchronize PXI 447x devices homogeneously as long as one device is plugged into the system timing slot of a PXI chassis.
- Heterogeneous synchronization:
    - You can synchronize a PXI 447x device with a PXI 446x device when the 446x is plugged into the system timing slot of a PXI chassis.
    - You cannot synchronize PXI 447x devices with PXI 449x devices.
    - You cannot use hybrid-slot compatible 446x devices.

DSA Device Compatibility Table

	446x Series	447x Series	449x Series
446x Series	✓	• PXI chassis only • Standard 446x device, not hybrid-slot compatible • 446x device in system timing slot	✓
447x Series	• PXI chassis only • Standard 446x device, not hybrid-slot compatible • 446x device in system timing slot	• PXI chassis only • One device in system timing slot	χ
449x Series	✓	χ	✓

PCI DSA Devices

PCI devices are synchronized use the RTSI cable. You can automatically synchronize these device series both homogeneously (within the same series) and heterogeneously (across separate series) in the same session when they are connected with a RTSI cable.

- PCI 446x series
- PCI 447x series

Note If you are synchronizing PCI devices make sure you register the RTSI cables in Measurement and Automation Explorer. For more information, see the NI knowledge base article Real-Time System Integration (RTSI) and Configuration Explained.

Synchronize DSA PCI Devices

This example shows how to acquire synchronized data from two DSA PCI devices, NI PCI-4461 and NI PCI-4462.

Connect the two devices with a RTSI cable.

Register your RTSI cable in Measurement and Automation Explorer.

Create an acquisition session and add one voltage analog input channel from each of the two PXI devices

```
s = daq.createSession('ni');
addAnalogInputChannel(s,'Dev1',0,'Voltage');
addAnalogInputChannel(s,'Dev2',0,'Voltage');
```

Synchronize the two channels using the `AutoSyncDSA` property:

```
s.AutoSyncDSA = true;
```

Acquire data in the foreground and plot it:

```
[data,time] = startForeground(s);
plot(time,data)
```

Handle Filter Delays with DSA Devices

DSA devices have a built in digital filter. You must account for filter delays when synchronizing between heterogeneous devices. Refer to your device manuals for filter delay information. For more information, see the NI knowledge base article Why Is My Data Delayed When Using DSA Devices? (Document ID: 2UI8PGX4).

Example 14.1. Account for Filter Delays

This example shows how to account for filter delays when you use the same sine wave to acquire from two different channels from two different PXI devices. Perfectly synchronized channels will show zero phase lag between the two acquired signals.

Create a session and add two analog input channels with `'Voltage'` measurement type, from National Instruments PXI-4462 and NI PXI-4472.

```
s = daq.createSession('ni');
ch1 = addAnalogInputChannel(s,'PXI1Slot2',0,'Voltage');
ch2 = addAnalogInputChannel(s,'PXI1Slot3',0,'Voltage');
```

Acquire unsynchronized data and plot it:

```
[data,time] = startForeground(s);
plot(time,data)
```

Use AutoSyncDSA to automatically configure the triggers, clocks, and sync pulses of the channels to synchronize the devices:

```
s.AutoSyncDSA = true;
```

Acquire synchronized data:

```
[data,time] = startForeground(s);
plot(time,data)
```

Calculate the phase lag between the two channels, using the device data sheet:

NI PXI 4462 data sheet specifies the phase lag to be 63 samples when `EnhancedAliasRejectionEnable` property is disabled. Check to make sure that this property is set to `false` or `0`:

```
ch1.EnhancedAliasRejectionEnable
```

```
ans =

           0
```

To synchronize signals from these devices, the phase lag should be `63-38` or `24` samples. Confirm that the data returned is 24 samples apart.

NI PXI 6672 data sheet specified the phase lag to be 38 samples when the `EnhancedAliasRejectionEnable` property is disabled. Check to make sure that this property is set to `false` or `0`:

```
ch2.EnhancedAliasRejectionEnable
```

```
ans =

           0
```

See Also

Properties
AutoSyncDSA | EnhancedAliasRejectionEnable

More About

- "Synchronize DSA PXI Devices Using AutoSyncDSA" on page 14-12

Transition Your Code to Session-Based Interface

Transition Your Code to Session-Based Interface

This topic helps you transition your code from the legacy interface to the session-based interface.

In this section...

Transition Common Workflow Commands

This table lists the legacy commands for common workflows and their corresponding session-based commands.

To do this	Legacy Command	Session-Based Command
Find supported hardware available to your system.	`daqhwinfo`	`daq.getDevices`
Registered DAQ adaptor.	`daqregister`	You do not need to register an adaptor if you are using session-based interface.
Reset MATLAB to initial state.	`daqreset`	`daqreset`
Discover newly connected hardware.	Shut down MATLAB and restart.	`daqreset`

Create analog input object and add a channel.	`ai = analoginput ('nidaq', 'Dev1');`
	`addchannel(ai, 1)`
		`s=daq.createSession('ni');`
		`addAnalogInputChannel (s,'Dev1',1,'Voltage');`

To do this	Legacy Command	Session-Based Command

Create analog output object	`ao = analogoutput ('nidaq', 'Dev1');`
	`addchannel(ao, 1)`
		`addAnalogOutputChannel (s,'Dev1',0,'Current')`

Create a digital input and output object and add a digital input line.	`dio = digitalio ('nidaq','Dev1');`
	`addline(dio,0:3,'in');`
		`s = daq.createSession('ni');`
		`addDigitalChannel (s,'Dev1','Port0/Line0:1','InputOnly');`

Create counter input channels	You cannot use counter channels in the legacy interface.	`s = daq.createSession ('ni')`
		`addCounterInputChannel (s,'Dev1','ctr0','EdgeCount')`

Start the object.	`start(ai)`	`startForeground(s);`
		for operations that block MATLAB when running.
		`startBackground (s);`
		for operations that run without blocking MATLAB.

Set rate of acquisition.	`ai.SampleRate=48000`	`s.rate=48000`
Specify an external trigger.	`ai.TriggerType= 'HwDigital';`	`addTriggerConnection (s,'External','Dev3/PFI0','StartTrigger');`

Specify a range of input signals	`ai.Channel.InputRange=[-5 5];`	`ch = addAnalogInputChannel (s,'Dev1',1,'Voltage');`
		`ch.Range = [-5 5];`

Acquire Analog Data

Legacy Interface

Using the legacy interface, you find hardware available to your system, create an analog input object and start acquisition.

1 Find hardware available to your system.

```
d = daqhwinfo;
```

2 Create an analog input object and add a channel using a National Instruments® device, with ID `Dev1`.

```
ai = analoginput('nidaq','Dev1');
addchannel(ai,1)
```

3. Set the sample rate to 8000 and start the channel.

   ```
   ai.SampleRate = 8000;
   start(ai)
   ```

Session-Based Interface

Using the session-based interface, you create a vendor session and add channels to the session. You can use any device or chassis from the same vendor available to your system and can add a combination of analog, digital, and counter input and output channels. All the channels operate together when you start the session.

Compatibility Note For devices that support both AC and DC coupling, in the session interface the default connection is DC coupling, setting the channel's Coupling property to `'DC'`. For the same devices in the legacy interface, the default was AC coupling.

---

1. Find hardware available to your system.

   ```
   d = daq.getDevices
   ```
2. Create a session for National Instruments devices.

   ```
   s = daq.createSession('ni');
   ```
3. Set the session's sample rate to 8000.

   ```
   s.Rate = 8000
   ```
4. Add an analog input channel for the device with ID Dev1 for voltage measurement, and then start the acquisition.

   ```
   addAnalogInputChannel(s,'Dev1',1,'Voltage');
   startForeground(s);
   ```

Use Triggers

Acquire analog data using hardware triggers.

Legacy Interface

Analog operations are configured to trigger immediately by default. You must specify `hwDigital` trigger type.

Create an analog input object and add two channels

1. Create an analog input object and add two channels

   ```
   ai = analoginput('nidaq','Dev1');
   chan = addchannel(ai,0:1)
   ```
2. Specify the ranges of the channel to scale the data uniformly. Configure the input type to be `SingleEnded` terminal.

   ```
   chan.InputRange = [-10 10];
   chan.UnitsRange = [-10 10];
   chan.SensorRange = [-10 10];
   chan.InputType = 'SingleEnded';
   ```
3. Specify the trigger type, source and condition. Set `TriggerRepeat` to 0.

   ```
   ai.TriggerType = 'HwDigital';
   ai.HwDigitalTriggerSource = 'PFI0';
   ai.TriggerCondition = 'PositiveEdge';
   ai.TriggerRepeat = 0;
   ```
4. Specify rate and duration.

   ```
   actualRate = setverify(ai,'SampleRate',50000);
   duration = 0.01;
   ai.SamplesPerTrigger = duration*actualRate;
   ```
5. Start the channel, wait until the channel receives the specified amount of data, and get the data.

   ```
   start(ai);

   wait(ai,duration+1);
   [data,timestamps] = getdata(ai);
   ```
6. Plot the data.

   ```
   plot(timestamps,data)
   ```

Session-Based Interface

You can specify an external event to trigger data acquisition using the session-based interface.

1. Create a session and add two analog input channels.

   ```
   s = daq.createSession('ni');
   ch = addAnalogInputChannel(s,'Dev1',0:1,'Voltage');
   ```

2 Configure the terminal and range of the channels in the session.

```
ch(1).TerminalConfig = 'SingleEnded';
ch(1).Range = [-10.0 10.0];
ch(2).TerminalConfig = 'SingleEnded';
ch(2).Range = [-10.0 10.0];
```

3 Create an external trigger connection and set the trigger to run one time.

```
addTriggerConnection(s,'External','Dev1/PFI0','StartTrigger');
s.Connections(1).TriggerCondition = 'RisingEdge';
s.TriggersPerRun = 1;
```

4 Set the rate and the duration of the acquisition.

```
s.Rate = 50000;
s.DurationInSeconds = 0.01;
```

5 Acquire data in the foreground and plot the data.

```
[data,timestamps] = startForeground(s);
plot(timestamps,data)
```

Log Data

Legacy Interface

You can log the data to disk and use `daqread` to read the data back.

1 Create the analog input object and add two channels.

```
ai = analoginput('winsound');
ch = addchannel(ai,0:1);
```

2 Define a 2-second acquisition for each trigger, set the trigger to repeat three times, and log information to the file `file00.daq`.

```
duration = 2;
ai.SampleRate = 8000;
actualRate = ai.SampleRate;
ai.SamplesPerTrigger = duration*ActualRate;
ai.TriggerRepeat = 3;
ai.LogFileName = 'file00.daq';
ai = LoggingMode = 'Disk&Memory';
```

3 Start the acquisition, wait for duration of the acquisition times the number of triggers for the acquisition to complete. Then extract all the data stored in the log file as sample-time pairs.

```
start(ai)
wait(ai, (ai.TriggerRepeat+1)*duration + 1)
[data,time] = daqread('file00.daq');
```

Session-Based Interface

The session-based interface does not have a specified file format for logging data. You can write to a file in binary mode or save data to a MAT-file.

1. Create a session and add four analog input channels from `Dev1`.

   ```
   s = daq.createSession('ni');
   ch = addAnalogInputChannel(s,'Dev1',0:3,'Voltage');
   ```
2. Set the same range and terminals for all the channels.

   ```
   ch(1).Range = [-10.0 10.0];
   ch(1).TerminalConfig = 'SingleEnded';
   ch(2).Range = [-10.0 10.0];
   ch(2).TerminalConfig = 'SingleEnded';
   ch(3).Range = [-10.0 10.0];
   ch(3).TerminalConfig = 'SingleEnded';
   ch(4).Range = [-10.0 10.0];
   ch(4).TerminalConfig = 'SingleEnded';
   ```
3. Set the session rate and duration of acquisition.

   ```
   s.Rate = 50000;
   s.DurationInSeconds = 0.01;
   ```
4. Start the acquisition and plot the data.

   ```
   [data,timestamps] = startForeground(s);
   figure
   plot(timestamps,data)
   ```
5. Save the acquired data to a MAT-file.

   ```
   fileName = 'data.mat';
   save(fileName,'timestamps','data')
   ```
6. Load data from the file into the MATLAB workspace.

   ```
   savedData = load('data.mat');
   figure;
   plot(savedData.timestamps,savedData.data)
   ```

Set Range of Analog Input Subsystem

You can specify the measurement range of an analog input subsystem.

Legacy Interface

1 Create the analog input object `ai` for a National Instruments device, and add two channels to it.

```
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:1);
```

2 Configure both channels to accept input signals in the range from -10 volts to 10 volts.

```
ai.Channel.InputRange = [-10 10];
```

Session-Based Interface

1 Create a session and add an analog input channel.

```
s = daq.createSession('ni');
ch = addAnalogInputChannel(s,'Dev1','ai1','Voltage')
```

2 Set the range from -10 volts to 10 volts.

```
ch.Range = [-10 10];
```

Fire an Event When Number of Scans Exceed Specified Value

You can specify your acquisition to watch for a specified number of scans to occur and fire an event if the acquisition exceeds the specified number.

Legacy Interface

You can use the `BufferingConfig` property to specify allocated memory for a specified channel. If the number of samples acquired exceeds the allocated memory, then an error is returned.

1 Create an analog input object `ai` for a National Instruments device and add a channel to it.

```
ai = analoginput('nidaq','Dev1');
ch = addchannel(ai,0);
```

2 Set the sample rate to 800,000.

```
ai.SampleRate = 800000)
```

3 Set the `bufferingConfigMode` to `'Manual'` and set the `bufferingConfig` value.

```
ai.bufferingConfigMode = 'Manual';
ai.bufferingConfig = [512 30];
```

Session-Based Interface

Use the NotifyWhenDataAvailableExceeds property to fire a `DataAvailable` event.

1 Create an acquisition session, add an analog input channel.

```
s = daq.createSession('ni');
addAnalogInputChannel(s,'Dev1','ai0','Voltage');
```

2 Set the Rate to 800,000 scans per second, which automatically sets the `DataAvailable` notification to automatically fire 10 times per second.

```
s.Rate = 800000;
s.NotifyWhenDataAvailableExceeds

ans =
    80000
```

3 Increase `NotifyWhenDataAvailableExceeds` to 160,000.

```
s.NotifyWhenDataAvailableExceeds = 160000;
```

Use Timeout to Block MATLAB While an Operation Completes

Legacy Interface

1 Create an analog output object `ao` for a National Instruments device, then add a channel and set it to output data at 8000 samples per second with one manual trigger.

```
ao = analogoutput('nidaq','Dev1');
ch = addchannel(ao,1);
ao.SampleRate =  8000;
ao.TriggerType = 'Manual';
ao.RepeatOutput = 0;
putdata(ao(zeros(10000,1)));
```

2 Start the acquisition and issue a wait command for the acquisition to block MATLAB for two seconds. If the acquisition does not complete in two seconds, a timeout occurs.

```
start(ao)
wait(ao,2)
```

Session-Based Interface

Background operations run without interrupting the MATLAB Command Window. You can use `wait` to block operations in the MATLAB Command Window during background operations.

1 Create an acquisition session, add an analog output channel.

```
s = daq.createSession('ni');
addAnalogOutputChannel(s,'Dev1','ao0','Voltage');
```

2 Set the session rate to 8000.

```
s.Rate = 8000
```

3 Queue some output data.

```
queueOutputData(s,zeros(10000,1));
```

4 Start the acquisition and issue a `wait` to block MATLAB. If the operation does not complete in 2 seconds, a timeout occurs.

```
startBackground(s);
s.wait(2);
```

Count Pulses

You can count pulses to clock your data acquisition.

Legacy Interface

You cannot use counter input and output channels with the legacy interface. You can use the analog input subsystem’s internal clock to create a threshold and look for consecutive samples that are on opposite sides of the threshold. This gives you results similar to using a counter input channel.

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

```
threshold = 3.5;
offsetData = [data(2:end);NaN];
risingEdge = find(data < threshold & offsetData > threshold);
fallingEdge = find(data > threshold & offsetData < threshold);
```

Session-Based Interface

Count edges of a pulse using a counter input channel on your device.

```
s.createSession('ni');
addCounterInputChannel(s,'Dev1','ctr0','EdgeCount');
c = inputSingleScan(s);
```

Troubleshooting Your Hardware

This appendix describes simple tests you can perform to troubleshoot your data acquisition hardware. The tests involve using software provided by the vendor or the operating system (sound cards), and do not involve using Data Acquisition Toolbox software. The sections are as follows.

Supported Hardware

For a list of hardware supported by Data Acquisition Toolbox, see `https://www.mathworks.com/hardware-support/data-acquistion-software.html`.

Hardware and Device Drivers

In this section...

Registering the Hardware Driver Adaptor

When you first create a device object, the associated hardware driver adaptor is automatically registered. The data acquisition engine can now make use of its services.

The hardware driver adaptors included with the toolbox are all located in the `daq/private` folder. These are the full names for each adaptor.

Supported Vendors/Device Types and Full Adaptor Names

Vendor/Device Type	Full Adaptor Name
National Instruments	`mwnidaq.dll`
Windows sound cards	`mwwinsound.dll`

Device Driver Registration

If you are using a Windows Vista™ or a Windows 7 system and cannot register device drivers, you could have UAC enabled on the system. Refer to this technical bulletin for more information.

Hardware Diagnostics

Run `daqsupport` to get diagnostic information for all installed hardware adaptors on your system. Use this information to diagnose issues with your hardware. Make sure you include this information when you contact MathWorks support.

Troubleshooting Tips

In this section...

Find Devices and Create a Session

Identify the devices you can access:

```
d  = daq.getDevices

d =

Data acquisition devices:

index Vendor Device ID              Description
----- ------ --------- -----------------------------------
1     ni     cDAQ1Mod1 National Instruments NI 9205
2     ni     cDAQ2Mod1 National Instruments NI 9201
```

Create a session object:

```
s = daq.createSession('ni');
```

For more information on the session-based information, see “Data Acquisition Session” on page 3-2.

Refer to “Session-Based Interface Workflow” on page 5-2 to learn how to communicate with CompactDAQ devices.

Is My NI-DAQ Driver Supported?

Data Acquisition Toolbox software is compatible only with specific versions of the NI-DAQ driver and is not guaranteed to work with any other versions. For a list of the NI-DAQ driver versions that are compatible with Data Acquisition Toolbox software, see `https://www.mathworks.com/hardware-support/data-acquistion-software.html`, and click the link for this vendor.

To see your installed driver version in the session-based interface, type:

```
v = daq.getVendors

v =

Number of vendors: 2

index    ID    Operational         Comment
----- -------- ----------- ------------------------
1     ni       true        National Instruments
2     digilent false       Click here for more info

Properties, Methods, Events

Additional data acquisition vendors may be available as downloadable support packages.
Open the Support Package Installer to install additional vendors.
```

If the version in the `DriverVersion` field does not match the minimum requirements specified on the product page on MathWorks website, update your drivers.

If your driver is incompatible with Data Acquisition Toolbox, verify that your hardware is functioning properly before updating drivers. If your hardware is not functioning properly, you are using unsupported drivers. For the latest NI-DAQ drivers, visit the National Instruments website at `https://www.ni.com/`.

To find driver version in the National Instruments **Measurement & Automation Explorer**:

1. Click **Start > Programs > National Instruments > Measurement & Automation Explorer**.
2. Select **Help > System Information**.

Why Doesn’t My Hardware Work?

Use the **Test Panel** to troubleshoot your National Instruments hardware. The **Test Panel** allows you to test each subsystem supported by your device, and is installed as part of the NI-DAQmx driver software. Right-click the device in the Measurement & Automation Explorer and choose **Test Panel**.

For example, to verify that the analog input subsystem on your PCIe-6363 device is operating, connect a known signal (similar to the signal produced by a function generator) to one or more channels, using a screw terminal panel.

If the **Test Panel** does not provide you with the expected results for the subsystem, and you are sure that your test setup is configured correctly, then the hardware is not performing correctly.

For your National Instruments hardware support, visit their website at `https://www.ni.com/`.

Cannot Create Session

If you try to create a session using `daq.createSession`, and you see the following error:

```
The vendor 'ni' is not known. Use 'daq.getVendors()' for a list of vendors.
```

1. get vendor information by typing:

   ```
   v = daq.getVendors
   ```

```
v =

Data acquisition vendor 'National Instruments':

            ID: 'ni'
      FullName: 'National Instruments'
AdaptorVersion: '2.17 (R2010b)'
 DriverVersion: '9.1 NI-DAQmx'
 IsOperational: true
```

If you do not see output like this, see “Cannot Find Hardware Vendor” on page A-7.

Why Was My Session Deleted?

This warning:

```
A session was deleted while it was running.
```

occurs when you start background operations in the session and the session is silently deleted. This could be caused by the session going out of scope at the end of a MATLAB function, before the background task completes. To avoid this, insert a pause after `startBackground`.

Cannot Find Hardware Vendor

If you try to get vendor information using `daq.getVendors` in the session-based interface, and receive one of the following errors:

- NI-DAQmx driver mismatch:

  ```
  Diagnostic Information from vendor: NI: There was a driver error while
  loading the MEX file to communicate with National Instruments hardware.
  It is possible that the NI-DAQmx driver is not installed or is older than
  the required minimum version of '8.7'.
  ```

  Install the NI-DAQmx driver of version specified in the error message.

  If you have a version of the NI-DAQmx driver already installed, update your installation to the minimum required version suggested in the error message.

- No vendors found:

  ```
  No data acquisition vendors available.
  ```

  Reinstall Data Acquisition Toolbox software.

- Corrupted or missing toolbox components:

  ```
  Diagnostic Information from vendor: NI: The required MEX file to communicate
  with National Instruments hardware is not in the expected location:
  ```

  Reinstall Data Acquisition Toolbox software.

  ```
  Diagnostic Information from vendor: NI: The required MEX file to communicate
  with National Instruments hardware exists but appears to be corrupt:
  ```

  Reinstall Data Acquisition Toolbox software.

Cannot Find Devices

If you try to find information using `daq.getDevices` and:

- Do not see the expected device listed. For example, if you are looking for an NI 9263 and NI 9265 and you type:

  ```
  d = daq.getDevices

  d =

  Data acquisition devices:

  index Vendor Device ID          Description
  ----- ------ --------- -----------------------------
  1     ni     cDAQ1Mod1 National Instruments NI 9205
  2     ni     cDAQ1Mod3 National Instruments NI 9203
  3     ni     cDAQ1Mod4 National Instruments NI 9201
  4     ni     cDAQ1Mod6 National Instruments NI 9213
  6     ni     cDAQ1Mod8 National Instruments NI 9265
  ```

  To refresh the toolbox, type

  ```
  daqreset
  ```

  If you still do not see the devices, go to the National Instruments Measurement & Automation Explorer (NI MAX) and examine the devices installed on your CompactDAQ chassis.

- Receive one of the following errors

  - `No data acquisition devices available.`

    - Go to NI MAX and examine the devices installed on your CompactDAQ chassis.
    - If you cannot see your devices in NI MAX, check to see if you have turned on and connected your chassis.
    - If you have turned on and connected your chassis and issued `daqreset`, and you can see the devices in NI MAX, reinstall Data Acquisition Toolbox software.
  - `The requested subsystem 'AnalogInput' does not exist on this device`.

    You could be:

    - Using an output device to add input channels. See `daq.getDevices` to learn more about an installed device.
    - Using an unsupported device. See “Supported Hardware” on page A-2.
  - `The requested subsystem 'AnalogOutput' does not exist on this device`.

    You could be:

    - Using an input device to add output channels. See `daq.getDevices` to learn more about an installed device.
    - Using an unsupported device. See “Supported Hardware” on page A-2.
- If you are using NI 9402 with the counter/timer subsystem with the cDAQ-9172 chassis, plug the module into slots 5 or 6 only. If you plug the module into one of the other slots, it will not show any counter/timer subsystem.
- If you are using an Ethernet CompactDAQ chassis, reserve the chassis in National Instruments Measurement & Automation Explorer first. Only one system can reserve this chassis at a time.

What Is a Reserved Hardware Error?

If you receive the following error:

```
The hardware associated with this session is reserved. If you are using it in another
session use the release function to unreserve the hardware. If you are using it in an
external program exit that program. Then try this operation again.
```

Identify the session that is currently not using this device, but has reserved it and release the associated hardware resources. If the device is reserved by:

Another session in the current MATLAB program.

Do one of the following:

- Use `release` to release the device from the session that is not using the device.
- Delete the session object.

Another session in a separate MATLAB program.

Do one of the following:

- Use `release` to release the device from the session that is not using the device.
- Delete the session object.
- Exit the MATLAB program.

Another application.

Exit the other application.

In none of these measures work, reset the device from NI MAX.

Note Your network device may also appear as unsupported in the device information if it is reserved or disconnected.

What Are Devices with an Asterisk (*)?

If you get device information and see a device listed with an asterisk (*) next to it, then the toolbox does not support this device.

```
d = daq.getDevices

d =

Data acquisition devices:

index Vendor Device ID           Description
----- ------ --------- ------------------------------
1     ni     cDAQ1Mod1 National Instruments NI 9401
2     ni     cDAQ1Mod7 National Instruments NI 9219
3     ni     cDAQ2Mod1 National Instruments NI 9205
4     ni     cDAQ2Mod2 National Instruments NI 9263
5     ni     cDAQ2Mod3 National Instruments NI 9203
6     ni     cDAQ2Mod4 National Instruments NI 9201
7     ni     cDAQ2Mod5 National Instruments NI 9265
8     ni     cDAQ2Mod6 National Instruments NI 9213
9     ni     cDAQ2Mod7 National Instruments NI 9227
10    ni     cDAQ2Mod8 National Instruments NI 9422
11    ni     Dev2      National Instruments PCIe-6363
```

```
12    ni     Dev3      National Instruments USB-6255
13    ni     Dev4      National Instruments USB-9233
14    ni     Dev5      * National Instruments PCI-6601
15    ni     Dev6      National Instruments PCI-6220
16    ni     Dev8      * National Instruments PCI-6509

* Device currently not supported. See documentation on Unsupported Devices for more information.
```

- Make sure that your network device is not reserved and not disconnected.
- For a list of supported devices, see `https://www.mathworks.com/hardware-support/data-acquistion-software.html`.

Network Device Appears with an Asterisk (*)

- If your network device appears as unsupported or unavailable, make sure that the device is connected and reserved in National Instruments Measurement and Automation Explorer. Issue `daqreset` to reset devices settings.
- If you see this timeout error when communicating with a network device:

  ```
  Network timeout error while communicating with device 'cDAQ9188-1595393Mod4'
  ```

  reconnect the device in National Instruments Measurement and Automation Explorer and issue `daqreset` to reset devices settings.

ADC Overrun Error with External Clock

If you see this error when you synchronize acquisition using an external clock,

```
ADC Overrun Error: If you are using an external clock, make sure that
the clock frequency matches session rate.
```

- check your external clock for the presence of noise or glitches.
- check the frequency of your external clock. Make sure that it matches the session's rate.

Cannot Add Clock Connection to PXI Devices

When you try to synchronize operations using a PXI 447x series device, you see this error:

```
"DSA device 'PXI1Slot2' does not support sample clock synchronization. Check device's user manual.
```

National Instruments DSA devices like the PXI 447x, do not support sample clock synchronization. You cannot synchronize these devices in the session-based interface using `addClockConnection`.

Cannot Complete Long Foreground Acquisition

When you try to acquire data in the foreground for a long period, you may get an out-of-memory error. Switch to background acquisitions and process data as it is received or save the data to a file to mitigate this issue.

Cannot Use PXI 4461 and 4462 Together

You cannot use PXI 4461 and 4462 together for synchronization, when PXI 4461 is in the timing slot of the chassis.

Counters Restart When You Call Prepare

Counters stop running in the background when you call `prepare` to perform clocked operations. This operation resets counters and restarts them when the new operation starts.

Cannot Get Correct Scan Rate with Digilent Devices

The scan rate when you use a Digilent device, can be limited by the hardware's buffer size. See "Digilent Analog Discovery Hardware Limitations" on page B-4 for more information on maximum and minimum allowable rates.

Cannot Simultaneously Acquire and Generate with myDAQ Devices

You cannot acquire and generate synchronous data using myDAQ devices because they do not share a hardware clock. If you have both input and output channels in a session and you start the session, you will see near-simultaneous acquisition and generation. See "Automatic Synchronization" on page 14-6 for more information.

Simultaneous Analog Input and Output Not Synchronized Correctly

Do you have an external trigger? When you simultaneously acquire and generate analog signals in the same session with an external trigger, they may correctly synchronize.

Counter Single Scan Returns NaN

An input single scan on counter input channels may return a NaN. If this occurs:

- make sure that the signal voltage complies with TTL voltage specifications.
- Make sure that the channel frequency is within the specified frequency range.

External Clock Will Not Trigger Scan

Adding an external clock to your session may not trigger a scan unless you set the session's rate to match the expected external clock frequency.

Why Does My S/PDIF Device Time Out?

S/PDIF audio ports appear in the device list even when you have no devices plugged in.

- If you add this device (port) to your session and you have no device plugged into the port, the operation times out.
- If you have a device plugged into the S/PDIF port, you may need to match the session rate to the device scan rate to get accurate readings. Refer to your device documentation for information.

Audio Output Channels Display Incorrect ScansOutputByHardware Value

If you have downloaded the Windows Audio support package with R2014a, you may see incorrect values for the sessions ScansOutputByHardware property. The hardware outputs the scans as specified and the property may incorrectly report this number. To correct it, execute this code:

```
s = daq.createSession('directsound')
scansOutputByHardware_incorrect = s.ScansOutputByHardware;
correction = s.NotifyWhenScansQueuedBelow - 1;
scansOutputByHardware_corrected = scansOutputByHardware_incorrect + correction;
```

MOTU Device Not Working Correctly

MOTU devices Ultralight-mk3 and Traveler-mk3 may not work with DirectSound and Data Acquisition Toolbox versions R2014a and R2014b. If you have these devices, specify the device to use stereo pairs:

- In your MOTU Audio Console check "Use Stereo Pairs for Windows Audio" check box.
- Specify desired sample rate in the Sample Rate field.

Contacting MathWorks

If you need support from MathWorks, visit our website at `https://www.mathworks.com/support/`.

Before contacting MathWorks, you should run the `daqsupport` function. This function returns diagnostic information such as:

- The versions of MathWorks products you are using
- Your MATLAB software path
- The characteristics of your hardware

The output from `daqsupport` is automatically saved to a text file, which you can use to help troubleshoot your problem. For example, to have the MATLAB software generate this file for you, type

```
daqsupport
```

Hardware Limitations by Vendor

This topic describes limitations of using hardware in the Data Acquisition Toolbox based on limitations places by the hardware vendor:

Limitations by Vendor

For some vendors, there are limitations in the toolbox support for their functionality. See the following topics for each vendor.

- "Digilent Analog Discovery Hardware Limitations" on page B-4
- "Measurement Computing Hardware Limitations" on page B-5
- "National Instruments Hardware Limitations" on page B-3
- "Analog Devices ADALM1000 Limitations" on page B-6

National Instruments Hardware Limitations

- Required hardware drivers and any other device-specific software is described in the documentation provided by your hardware vendor. For more information, see NI-DAQmx Support from Data Acquisition Toolbox.
- You can use PXI_STAR with `addTriggerConnection` and `addClockConnection` functions. All supported PXI modules automatically use the reference Clock PXI_CLK10.
- Objects created for National Instruments devices, and used with the NI-DAQmx adaptor have the following behavior when you use the `inputSingleScan` or `outputSingleScan` function in the session-based interface:
  - The first time the command is used with the object, the corresponding subsystem of the device is reserved by the MATLAB session.
  - If you then try to access that subsystem in a different session of the MATLAB software, or any other application from the same computer, you might receive an error message informing you that the subsystem is reserved. Use `release` to unreserve the subsystem.
- You cannot acquire and generate synchronous data using myDAQ devices because they do not share a hardware clock. If you have both input and output channels in a session and you start the session, you will see near-simultaneous acquisition and generation. See “Automatic Synchronization” on page 14-6 for more information.
- NI USB devices that have their own power supply can shut down if the driver does not set the USB power correctly.

Note The Traditional NI-DAQ adaptor will be deprecated in a future version of the toolbox. If you create a Data Acquisition Toolbox™ object for Traditional NI-DAQ adaptor beginning in R2008b, you will receive a warning stating that this adaptor will be removed in a future release. For more information, see the supported hardware page at `https://www.mathworks.com/hardware-support/data-acquistion-software.html`.

Digilent Analog Discovery Hardware Limitations

- You cannot use multiple Digilent devices in the same session. If you need to use multiple devices, add one device per session and start the sessions sequentially.
- Digilent devices limit the minimum and maximum allowable rate of sampling based on channel types:

  - Analog input only: 0.1 – 1,000,000
  - Analog output only: 4,096 – 1,000,000
  - Input and output: 8,192 – 300,000

  Data Acquisition Toolbox conforms to the Digilent Player Mode for the Arbitrary Waveform Generator.
- You cannot use background operations with Digilent devices. You can only perform foreground operations using `startForeground`
- You cannot perform synchronous and triggered operations using a Digilent device in the session-based interface.
- You cannot access the digital input and output capabilities of a Digilent device.

Measurement Computing Hardware Limitations

- For your Measurement Computing device to appear in the output of the `daq.getDevices` function, you must first detect it in InstaCal.
- MCC devices are not supported by the Simulink blocks of the Data Acquisition Toolbox block library.
- MCC devices are not supported by the Analog Input Recorder.
- External clocking and triggering of MCC devices is not supported.
- Support for MCC devices is limited to analog output voltage and analog input voltage measurements.
- MCC `DEMO-BOARD` devices simulated in Instacal are not supported.

Analog Devices ADALM1000 Limitations

The following restrictions and limitations apply when programming the Analog Devices ADALM1000. Some are restrictions of the hardware, some are restrictions imposed by Data Acquisition Toolbox.

- You cannot add channels from multiple ADALM1000 modules in the same session object. To recover from attempting this, you might need to execute `daqreset`.
- You cannot simultaneously source and measure voltage on the same channel, nor simultaneously source and measure current on the same channel.
- You cannot execute a single-scan operation that performs both source and measurement simultaneously.
- You cannot use AC coupling, nor differential terminal configurations.
- You cannot use triggers or digital pins.
- You cannot measure current without generating an output voltage.
- When specified output ranges are exceeded, the device might reset itself. Any measurements taken during this time might be unreliable until the reset is complete.

Not all data acquisition session background operations are supported. Use foreground operation for full generation and acquisition functionality.

Examples by Vendor

See the following topics for examples of each hardware vendor.

- “Analog Devices ADALM1000 Examples” on page B-8
- “Digilent Analog Discovery Hardware Examples” on page B-9
- “Measurement Computing Hardware Examples” on page B-10
- “National Instruments Hardware Examples” on page B-11
- “Windows Sound Card Examples” on page B-14

Analog Devices ADALM1000 Examples

“Characterize an LED with ADALM1000”

“Estimate the Transfer Function of a Circuit with ADALM1000”

See Also

More About

- “Digilent Analog Discovery Hardware Examples” on page B-9
- “Measurement Computing Hardware Examples” on page B-10
- “National Instruments Hardware Examples” on page B-11
- “Windows Sound Card Examples” on page B-14

Digilent Analog Discovery Hardware Examples

"Getting Started Acquiring Data with Digilent® Analog Discovery™"

"Getting Started Generating Data with Digilent® Analog Discovery™"

"Acquiring and Generating Data at the Same Time with Digilent® Analog Discovery™"

"Simultaneously Acquire and Generate Periodic Waveforms using Digilent® Analog Discovery™"

"Simultaneously Acquire and Generate Arbitrary Periodic Waveforms using Digilent® Analog Discovery™"

See Also

More About

- "Analog Devices ADALM1000 Examples" on page B-8
- "Measurement Computing Hardware Examples" on page B-10
- "National Instruments Hardware Examples" on page B-11
- "Windows Sound Card Examples" on page B-14

Measurement Computing Hardware Examples

“Getting Started with MCC Devices”

“Discover MCC Devices Using the Session-Based Interface”

“Acquire Data from Multiple Channels using MCC Devices”

See Also

More About

- “Analog Devices ADALM1000 Examples” on page B-8
- “Digilent Analog Discovery Hardware Examples” on page B-9
- “National Instruments Hardware Examples” on page B-11
- “Windows Sound Card Examples” on page B-14

National Instruments Hardware Examples

Getting Started and Device Discovery

"Getting Started with NI Devices"

"Discover NI Devices Using the Session-Based Interface"

Analog Input and Output

"Acquire Data Using NI Devices"

"Acquire Continuous and Background Data Using NI Devices"

"Acquire Data from an Accelerometer"

"Measure Strain using an Analog Bridge Sensor"

"Acquire Temperature Data from a Thermocouple"

"Acquire Temperature Data from an RTD"

"Acquire and Analyze Sound Pressure Data from an IEPE Microphone"

"Acquire and Analyze Noisy Clock Signals"

"Generate Signals on NI devices that Output Voltage"

"Generate Signals on NI devices that Output Current"

"Generate Continuous and Background Signals Using NI Devices"

"Acquire Data and Generate Signals at the Same Time"

"Log Analog Input Data to a File Using NI Devices"

"Software-Analog Triggered Data Capture"

"Analog Trigger App"

"Live Data Acquisition App"

Digital Input and Output

“Control Stepper Motor using Digital Outputs”

“Communicate with I2C devices and analyze bus signals using digital IO”

Counters and Timers

“Count Pulses on a Digital Signal Using NI Devices”

“Measure Frequency Using NI Devices”

“Measure Pulse Width Using NI Devices”

“Generate Pulse Width Modulated Signals Using NI Devices”

“Measure Angular Position with an Incremental Rotary Encoder”

Simultaneous and Synchronized Operations

“Synchronize NI PCI devices using RTSI”

“Start a Multi-Trigger Acquisition on an External Event”

“Acquire Data from Two Devices at Different Rates”

Simulink Data Acquisition

“Perform Live Acquisition, Signal Processing, and Generation”

“Perform Spectral Analysis on Live Data”

See Also

More About

- “Analog Devices ADALM1000 Examples” on page B-8
- “Digilent Analog Discovery Hardware Examples” on page B-9

- “Measurement Computing Hardware Examples” on page B-10
- “Windows Sound Card Examples” on page B-14

Windows Sound Card Examples

“Acquire Continuous Audio Data”

“Generate Audio Signals”

“Generating Multichannel-Audio”

“Analog Trigger App by Using Stateflow Charts”

See Also

More About

- “Analog Devices ADALM1000 Examples” on page B-8
- “Digilent Analog Discovery Hardware Examples” on page B-9
- “Measurement Computing Hardware Examples” on page B-10
- “National Instruments Hardware Examples” on page B-11

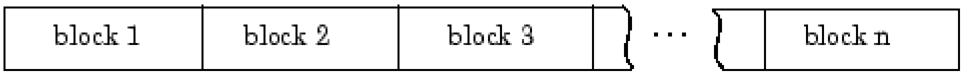
Managing Your Memory Resources

Manage memory allocation on your system to temporarily store data that is used by an analog input or output subsystem. This topic tells you:

What is Memory Allocation

When data is acquired from an analog input subsystem or output to an analog output subsystem, it must be temporarily stored in computer memory.

Data Acquisition Toolbox software allocates memory in terms of *data blocks.* A data block is defined as the smallest “slice” of memory that the data acquisition engine can usefully manipulate. For example, acquired data is logged to a disk file using an integral number of data blocks. A representation of allocated memory using *n* data blocks is shown below.



Data Acquisition Toolbox software strives to make memory allocation as simple as possible. For this reason, the data block size and number of blocks are automatically calculated by the engine. This calculation is based on the parameters of your acquisition such as the sampling rate, and is meant to apply to most common data acquisition applications. Additionally, as data is acquired, the number of blocks dynamically increases up to a predetermined limit. However, the engine cannot guarantee that the appropriate block size, number of blocks, or total memory is allocated under these conditions:

- You select certain property values. For example, if the samples to acquire per trigger are significantly less than the FIFO buffer of your hardware.
- You acquire data at the limits of your hardware, your computer, or the toolbox. In particular, if you are acquiring data at very high sampling rates, then the allocated memory must be carefully evaluated to guarantee that samples are not lost.

You are free to override the memory allocation rules used by the engine and manually change the block size and number of blocks, provided the device object is not running. However, you should do so only after careful consideration, as system performance might be adversely affected, which can result in lost data.

How Much Memory Do You Need?

The memory (in bytes) required for data storage depends on these factors:

- The number of hardware channels you use
- The number of samples you need to store in the engine
- The data type size of each sample

The memory required for data storage is given by the formula:

$$memoryrequired = samplesstored \times channelnumber \times datatype$$

Of course, the number of samples you need to store in the engine at any time depends on your particular needs. The memory used by a device object is given by the formula:

$$memoryused = blocksize \times blocknumber \times channelnumber \times datatype$$