Data Acquisition Toolbox™

Reference

R2013**b**

MATLAB®&SIMULINK®

MathWorks®

How to Contact MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Data Acquisition Toolbox™ Reference

Revision History

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)

Contents

1

Base Properties — Alphabetical List

ActiveEdge

Purpose Rising or falling edges of EdgeCount signals

Description When working with the session-based interface, use the `ActiveEdge` property to represent rising or falling edges of a EdgeCount signal.

Values You can set the Active edge of a counter input channel to `Rising` or `Falling`.

Examples

```
s = daq.createSession('ni');
ch = s.addCounterInputChannel ('cDAQ1Mod5', 0, 'EdgeCount')

ch =

Data acquisition counter input edge count channel 'ctr0' on device 'Dev2'

      ActiveEdge: Rising
  CountDirection: Increment
    InitialCount: 0
        Terminal: 'PFI8'
            Name: empty
              ID: 'ctr0'
          Device: [1x1 daq.ni.DeviceInfo]
 MeasurementType: 'EdgeCount'
```

Change the `Active Edge` property to `'Falling'`:

```
ch.ActiveEdge = 'Falling'

ch =

Data acquisition counter input edge count channel 'ctr0' on device 'Dev2':

      ActiveEdge: Falling
  CountDirection: Increment
    InitialCount: 0
        Terminal: 'PFI8'
            Name: empty
```

```
              ID: 'ctr0'
          Device: [1x1 daq.ni.DeviceInfo]
 MeasurementType: 'EdgeCount'
```

See Also

Methods

daq.Session.addCounterInputChannel, daq.Session.addCounterOutputChannel

Class

daq.Session

ActivePulse

Purpose

Active pulse measurement of PulseWidth counter channel

Description

When working with the session-based interface , the `ActivePulse` property displays the pulse width measurement in seconds of your counter channel, with `PulseWidth` measurement type.

Values

Active pulse measurement values include:

- `'High'`
- `'Low'`

Examples

Create a session object, add a counter input channel, with the `'EdgeCount' MeasurementType`.

```
s = daq.createSession('ni');
ch = s.addCounterInputChannel ('cDAQ1Mod5', 0, 'PulseWidth')

ch =

Data acquisition counter input pulse width channel 'ctr0' on device 'cDAQ1Mod5':

       ActivePulse: High
          Terminal: 'PFI4'
              Name: empty
                ID: 'ctr1'
            Device: [1x1 daq.ni.DeviceInfo]
   MeasurementType: 'PulseWidth
```

Change the `ActiveEdge` property to `Low`.

```
ch.ActivePulse = 'Low'

ch =

Data acquisition counter input pulse width channel 'ctr0' on device 'cDAQ

       ActivePulse: Low
```

```
         Terminal: 'PFI4'
             Name: empty
               ID: 'ctr1'
           Device: [1x1 daq.ni.DeviceInfo]
  MeasurementType: 'PulseWidth'
```

See Also

Class

`daq.Session`, `daq.Session.addCounterInputChannel`

ADCTimingMode

Purpose

Set channel timing mode

Description

When working with the session-based interface, use the `ADCTimingMode` property to specify if the timing mode in of all channels in the device is high resolution or high speed.

Note The `ADCTimingMode` must be the same for all channels on the device.

Values

You can set the `ADCTimingMode` to:

- `'HighResolution'`
- `'HighSpeed'`
- `'Best50HzRejection'`
- `'Best60HzRejection'`

Examples

Create a session and add an analog input channel:

```
s = daq.createSession('ni');
s.addAnalogInputChannel('cDAQ1Mod1','ai1','Voltage');
s.Channels

ans =

Data acquisition analog input voltage channel 'ai1' on device 'cDAQ1Mod1'

       Coupling: DC
 TerminalConfig: SingleEnded
          Range: -10 to +10 Volts
           Name: ''
             ID: 'ai1'
         Device: [1x1 daq.ni.CompactDAQModule]
MeasurementType: 'Voltage'
```

```
ADCTimingMode: ''
```

Set the `ADCTimingMode` property to `'HighResolution'`:

```
s.Channels.ADCTimingMode = 'HighResolution';
```

See Also

Methods

`daq.Session.addAnalogInputChannel`

Class

`daq.Session`

AutoSyncDSA

Purpose

Automatically Synchronize DSA devices

Description

Use this property to enable or disable automatic synchronization between DSA (PXI or PCI) devices in the same session. By default the sessions automatic synchronization capability is disabled.

Examples

To enable automatic synchronization, create a session and add channels from a DSA device:

```
s=daq.createSession('ni')
s.addAnalogInputChannel('PXI1Slot2',0,'Voltage');
s.addAnalogInputChannel('PXI1Slot3',1,'Voltage');
```

Enable automatic synchronization and acquire data”

```
s.AutoSyncDSA=true;
s.startForeground;
```

See Also

`daq.Session.addAnalogInputChannel`

Purpose

Specify analog input device bridge mode

Description

Use this property in the session-based interface to specify the bridge mode, which represents the active gauge of the analog input channel.

The bridge mode is `'Unknown'` when you add a bridge channel to the session. Change this value to a valid mode to use the channel. Valid bridge modes are:

- `'Full'` — All four gauges are active.
- `'Half'`— Only two bridges are active.
- `'Quarter'`— Only one bridge is active.

See Also

Class

`daq.Session`

BufferingConfig

Purpose

Specify per-channel allocated memory

Description

Note You cannot use the legacy interface on 64–bit MATLAB®. See "Session-Based Interface" to acquire and generate data.

`BufferingConfig` is a two-element vector that specifies the per-channel allocated memory. The first element of the vector specifies the block size, while the second element of the vector specifies the number of blocks. The total allocated memory (in bytes) is given by

(block size).(number of blocks).(number of channels).(native data type)

You can determine the native data type with `daqhwinfo`.

You can allocate memory automatically or manually. If `BufferingMode` is `Auto`, the `BufferingConfig` values are automatically set by the engine. If `BufferingMode` is `Manual`, then you must manually set the `BufferingConfig` values. If you change the `BufferingConfig` values, `BufferingMode` is automatically set to `Manual`.

When memory is automatically allocated by the engine, the block-size value depends on the sampling rate and is typically a binary number. The number of blocks is initially set to a value of 30 but can dynamically increase to accommodate the memory requirements. In most cases, the number of blocks used results in a per-channel memory that is somewhat greater than the `SamplesPerTrigger` value. When you manually allocate memory, the number of blocks is not dynamic and care must be taken to ensure there is sufficient memory to store the acquired data. If the number of samples acquired or queued exceeds the allocated memory, then an error is returned.

You can easily determine the memory allocated and available memory for each device object with the `daqmem` function.

Characteristics

Usage	AI, AO, common to all channels
Access	Read/write
Data type	Two-element vector of doubles
Read-only when running	Yes

Values

The default value is determined by the engine, and is based on the number of channels contained by the device object and the sampling rate. The `BufferingMode` value determines whether the values are automatically updated as data is acquired. For analog output objects, the default number of blocks is two.

Note If you change the `BufferingConfig` property for an analog output object, all previously queued output data will get discarded.

Examples

Create the analog input object `ai` for a sound card and add two channels to it.

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

The block size and number of blocks are given by `BufferingConfig`, while the native data type for the sound card is given by `daqhwinfo`.

```
ai.BufferingConfig
ans =
   512    30
out = daqhwinfo(ai);
out.NativeDataType
ans =
int16
```

With this information, the total allocated memory is calculated to be 61,440 bytes. This number is stored by `daqmem`.

```
out = daqmem(ai);
out.UsedBytes
ans =
      61440
```

The allocated memory is more than sufficient to store 8000 two-byte samples for two channels. If more memory was required, then the number of blocks would dynamically grow because `BufferingMode` is set to `Auto`.

See Also

Functions

`daqhwinfo`, `daqmem`

Properties

`BufferingMode`, `SampleRate`, `SamplesPerTrigger`

Purpose

Specify how memory is allocated

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`BufferingMode` can be set to `Auto` or `Manual`. If `BufferingMode` is set to `Auto`, the data acquisition engine automatically allocates the required memory. If `BufferingMode` is set to `Manual`, you must manually allocate memory with the `BufferingConfig` property.

If `BufferingMode` is set to `Auto` and the `SampleRate` value is changed, then the `BufferingConfig` values might be recalculated by the engine. Specifically, you can increase (decrease) the block size if `SampleRate` is increased (decreased). If `BufferingMode` is set to `Auto` and you change the `BufferingConfig` values, then `BufferingMode` is automatically set to `Manual`. If `BufferingMode` is set to `Manual`, then you cannot set the number of blocks to a value less than three.

For most data acquisition applications, you should set `BufferingMode` to `Auto` and have memory allocated by the engine because this minimizes the chance of an out-of-memory condition.

Characteristics

Usage	AI, AO, common to all channels
Access	Read/write
Data type	String
Read-only when running	Yes

Values

`{Auto}`	Memory is allocated by the data acquisition engine.
`Manual`	Memory is allocated manually.

See Also

Functions

daqmem

Properties

BufferingConfig

Purpose

Contain hardware channels added to device object

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`Channel` is a vector of all the hardware channels contained by an analog input (AI) or analog output (AO) object. Because a newly created AI or AO object does not contain hardware channels, `Channel` is initially an empty vector. The size of `Channel` increases as channels are added with the `addchannel` function, and decreases as channels are removed using the `delete` function.

`Channel` is used to reference one or more individual channels. To reference a channel, you must know its MATLAB index, which is given by the `Index` property. For example, you must use `Channel` with the appropriate indices when configuring channel property values.

For scanning hardware, the scan order follows the MATLAB index. Therefore, the hardware channel associated with index 1 is sampled first, the hardware channel associated with index 2 is sampled second, and so on. To change the scan order, you can specify a permutation of the indices with `Channel`.

Characteristics

Usage	AI, AO
Access	Read/write
Data type	Vector of channels
Read-only when running	Yes

Values

Values are automatically defined when channels are added to the device object with the `addchannel` function. The default value is an empty column vector.

Examples

Create the analog input object ai for a National Instruments® card and add three hardware channels to it.

```
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:2);
```

To set a property value for the first channel added (ID = 0), you must reference the channel by its index using the Channel property.

```
chans = ai.Channel(1);
set(chans,'InputRange',[-10 10])
```

Based on the current configuration, the hardware channels are scanned in order from 0 to 2. To swap the scan order of channels 0 and 1, you can specify the appropriate permutation of the MATLAB indices with Channel.

```
ai.Channel([1 2 3]) = ai.Channel([2 1 3]);
```

See Also

Functions

addchannel, delete

Properties

HwChannel, Index

Purpose

Specify descriptive channel name

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`ChannelName` specifies a descriptive name for a hardware channel. If a channel name is defined, then you can reference that channel by its name. If a channel name is not defined, then the channel must be referenced by its index. Channel names are not required to be unique.

You can also define descriptive channel names when channels are added to a device object with the `addchannel` function.

Characteristics

Usage	AI, AO, per channel
Access	Read/write
Data type	String
Read-only when running	Yes

Values

The default value is an empty string. To reference a channel by name, it must contain only letters, numbers, and underscores and must begin with a letter.

Examples

Create the analog input object `ai` for a sound card and add two channels to it.

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

To assign a descriptive name to the first channel contained by `ai`:

```
Chan1 = ai.Channel(1)
set(Chan1,'ChannelName','Joe')
```

You can now reference this channel by name instead of by index.

```
set(ai.Joe,'Units','Decibels')
```

See Also

Functions

addchannel

Purpose

Array of channel objects associated with session object

Description

This session object property contains and displays an array of channels added to the session. For more information on the session-based interface, see "Session-Based Interface".

Tip You cannot directly add or remove channels using the `Channels` object properties. Use `daq.Session.addAnalogInputChannel` and `daq.Session.addAnalogOutputChannel` to add channels. Use `daq.Session.removeChannel` to remove channels.

Values

The value is determined by the channels you add to the session object.

Examples

Create a session object, add an analog input channel, and display the session `Channels` property.

```
s = daq.createSession('ni');
s.addAnalogInputChannel('cDAQ1Mod1','ai1','Voltage');
s.Channels

ans =

Data acquisition analog input channel 'ai1' on device 'cDAQ1Mod1':

     Coupling: DC
    InputType: Differential
        Range: -10 to +10 Volts
         Name: empty
           ID: 'ai1'
       Device: [1x1 daq.ni.CompactDAQModule]
ADCTimingMode: empty
```

Add an analog output channel and view the `Channels` property:

```
s.addAnalogOutputChannel('cDAQ1Mod2', 'ao1', 'Voltage');
```

```
s.Channels


ans =


Number of channels: 2
   index Type  Device   Channel InputType      Range       Name
   ----- ---- --------- ------- --------- ---------------- ----
   1     ai   cDAQ1Mod1 ai1     Diff      -10 to +10 Volts
   2     ao   cDAQ1Mod2 ao1     n/a       -10 to +10 Volts
```

Change the `InputType` property of the input channel to `SingleEnded`:

```
s.Channels(1).InputType = 'SingleEnded';
```

See Also

Methods

`daq.Session.addAnalogInputChannel`,
`daq.Session.addAnalogOutputChannel`

Class

`daq.Session`

Purpose

Specify time between consecutive scanned hardware channels

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`ChannelSkew` applies only to scanning hardware and not to simultaneous sample and hold (SS/H) hardware.

If `ChannelSkewMode` is set to `Minimum` or `Equisample`, then `ChannelSkew` is automatically set to the appropriate device-specific read-only value. For SS/H hardware, the only valid `ChannelSkew` value is zero. For some vendors, `ChannelSkewMode` is automatically set to `Manual` if you first set `ChannelSkew` to a valid value.

Characteristics

Usage	AI, common to all channels
Access	Read/write (depends on `ChannelSkewMode` value)
Data type	Double
Read-only when running	Yes

Values

For SS/H hardware, the only valid value is zero. For scanning hardware, the value depends on `ChannelSkewMode`. `ChannelSkew` is specified in seconds.

See Also

Properties

`ChannelSkewMode`

ChannelSkewMode

Purpose

Specify how channel skew is determined

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

For simultaneous sample and hold (SS/H) hardware, `ChannelSkewMode` is `None`. For scanning hardware, `ChannelSkewMode` can be `Minimum`, `Equisample`, or `Manual` (National Instruments only). SS/H hardware includes sound cards, while scanning hardware includes most Measurement Computing™ and NI boards. Note that some supported boards from these vendors are SS/H, such as Measurement Computing's PCI-DAS4020/12.

If `ChannelSkewMode` is `Minimum`, then the minimum channel skew supported by the hardware is used. Some vendors refer to this as burst mode. If `ChannelSkewMode` is `Equisample`, the channel skew is given by $[(\text{sampling rate})(\text{number of channels})]^{-1}$. If `ChannelSkewMode` is `Manual`, then you must specify the channel skew with the `ChannelSkew` property. For some vendors, `ChannelSkewMode` is automatically set to `Manual` if you first set `ChannelSkew` to a valid value.

Notes If you want to use the maximum sampling rate of your hardware, you should set `ChannelSkewMode` to `Equisample`.

Large loads on the input device, especially if you are using multiple channels with scanning hardware, can increase the settling time. To improve the settling time, set `ChannelSkewMode` to `Equisample` and lower your sample rate.

Characteristics

Usage	AI, common to all channels
Access	Read/write

Data type	String
Read-only when running	Yes

Values

Advantech®

`{Equisample}`	The channel skew is given by [(sampling rate)(number of channels)]$^{-1}$.

Measurement Computing

`{Minimum}`	The channel skew is set to the minimum supported value.
`Equisample`	The channel skew is given by [(sampling rate)(number of channels)]$^{-1}$.

National Instruments

`{Minimum}`	The channel skew is set to the minimum supported value.
`Equisample`	The channel skew is given by [(sampling rate)(number of channels)]$^{-1}$.
`Manual`	The channel skew is given by `ChannelSkew`.

Sound Cards

`{None}`	This is the only supported value for SS/H hardware.

Examples

Create an analog input object for an `MCC` device and add eight channels.

```
ai = analoginput('mcc',1);
addchannel(ai,0:7);
```

Using the default `ChannelSkewMode` value of `Min` and the default `SampleRate` value of `1000`, the corresponding `ChannelSkew` value is

```
ai.ChannelSkew
ans =
  1.0000e-005
```

To use the maximum sampling rate, set `ChannelSkewMode` to `Equisample`.

```
ai.ChannelskewMode = 'Equisample';
ai.Samplerate = 100000/8;
```

See Also

Properties

`ChannelSkew`, `SampleRate`

Purpose

Specify clock that governs hardware conversion rate

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

For all supported hardware except Measurement Computing analog output subsystems, `ClockSource` can be set to `Internal`, which specifies that the acquisition rate is governed by the internal hardware clock.

Use this table to map to the National Instruments terminology.

Data Acquisition Toolbox™	NI_DAQmx
Scan Clock	Sample Clock
Sample Clock	Convert Clock

For subsystems without a hardware clock, you must use software clocking to govern the sampling rate. Software clocking allows a maximum sampling rate of 500 Hz and a minimum sampling rate of 0.0002 Hz. An error is returned if more than 1 sample of jitter is detected. Note that you might not be able to attain rates over 100 Hz on all systems.

Characteristics

Usage	AI, AO, common to all channels
Access	Read/write
Data type	String
Read-only when running	Yes

Values

Advantech

`{Internal}`	The internal hardware clock is used (AI only).
`External`	Externally control the channel clock (AI only).
`Software`	The computer clock is used.

Measurement Computing

`{Internal}`	The internal hardware clock is used.
`External`	Externally control the channel clock.
`Software`	The computer clock is used.

National Instruments

`{Internal}`	The internal hardware clock is used.
`External`	Externally control the channel clock (AO only).
`ExternalSampleCtrl`	Externally control the channel clock. This value overrides the `ChannelSkew` property value (AI only). This value does not apply to cards with simultaneous sample and hold.

Note If you set `ClockSource` to `ExternalSampleCtrl` then the value of `ExternalSampleClockSource` specifies the pin whose signal is used as the channel clock for conversions on each channel.

`ExternalScanCtrl`	Externally control the scan clock. This value overrides the `SampleRate` property value (AI only). **Note** If you set `ClockSource` to `ExternalScanCtrl` then the value of `ExternalScanClockSource` specifies the pin whose signal is used as the scan clock to initiate conversions across a group of channels.
`ExternalSampleAndScanCtrl`	Externally control the channel and scan clocks. This value overrides the `ChannelSkew` and `SampleRate` property values (AI only). This value does not apply to cards with simultaneous sample and hold. **Note** If you set `ClockSource` to `ExternalSampleAndScanCtrl` then the value of `ExternalSampleClockSource` specifies the pin whose signal is used as the channel clock for conversions on each channel, and the value of `ExternalScanClockSource` specifies the pin whose signal is used as the scan clock to initiate conversions across a group of channels.

Note If you set the `ClockSource` property to one of the `External` options, you must also set the `SampleRate` property to a value close to the external clock rate. `SampleRate` does not directly affect the external device, and the device will not use `SampleRate` if you have set an external clock rate, but this ensures that the toolbox configures itself correctly for expected data rates.

Sound Cards

{Internal}	The internal hardware clock is used.

See Also

Properties

ChannelSkew, SampleRate

Purpose

Array of connections in a session

Description

This session property contains and displays all connections added to the session. .

Tip You cannot directly add or remove connections using the `Connections` object properties. Use `daq.Session.addTriggerConnection` and `daq.Session.addClockConnection` to add connections. Use `daq.Session.removeConnection` to remove connections.

Values

The value is determined by the connections you add to the session.

Examples

Remove Synchronization Connection

This example shows you how to remove a synchronization connection.

Create a session and add analog input channels and trigger and clock connections.

```
s = daq.createSession('ni')
s.addAnalogInputChannel('Dev1', 0, 'voltage');
s.addAnalogInputChannel('Dev2', 0, 'voltage');
s.addAnalogInputChannel('Dev3', 0, 'voltage');
s.addTriggerConnection('Dev1/PFI4','Dev2/PFI0','StartTrigger');
s.addTriggerConnection('Dev1/PFI4','Dev3/PFI0','StartTrigger');
s.addClockConnection('Dev1/PFI5','Dev2/PFI1','ScanClock');
```

Examine the session Connections property.

```
s.Connections
```

```
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
```

Remove the last clock connection at index 4 and display the session connections.

```
s.removeConnection(4)
s.Connections

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'

   index     Type      Source   Destination
   ----- ------------ --------- -----------
   1     StartTrigger Dev1/PFI4 Dev2/PFI0
   2     StartTrigger Dev1/PFI4 Dev3/PFI0
   3     ScanClock    Dev1/PFI5 Dev2/PFI1
```

See Also

Methods

`daq.Session.addTriggerConnection`
`daq.Session.addClockConnection`

Class

`daq.Session`

CountDirection

Purpose

Specify direction of counter channel

Description

When working with the session-based interface, use the `CountDirection` property to set the direction of the counter. Count direction can be 'Increment', in which case the counter operates in incremental order, or 'Decrement', in which the counter operates in decremental order.

Examples

Create a session object, add a counter input channel, and change the `CountDirection`.

```
s = daq.createSession('ni');
ch = s.addCounterInputChannel ('cDAQ1Mod5', 0, 'EdgeCount')

ch =

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

      ActiveEdge: Rising
  CountDirection: Increment
    InitialCount: 0
        Terminal: 'PFI8'
            Name: empty
              ID: 'ctr0'
          Device: [1x1 daq.ni.DeviceInfo]
 MeasurementType: 'EdgeCount'
```

Change `CountDirection` to `'Decrement'`:

```
ch.CountDirection = 'Decrement'

ch =

Data acquisition counter input edge count channel 'ctr0' on device 'cDAQ1

      ActiveEdge: Rising
  CountDirection: Decrement
    InitialCount: 0
```

```
          Terminal: 'PFI8'
              Name: empty
                ID: 'ctr0'
            Device: [1x1 daq.ni.DeviceInfo]
   MeasurementType: 'EdgeCount'
```

See Also

Class

daq.Session, daq.Session.addCounterInputChannel

DataMissedFcn

Purpose

Specify callback function to execute when data is missed

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

A data missed event is generated immediately after acquired data is missed. This event executes the callback function specified for `DataMissedFcn`. The default value for `DataMissedFcn` is `daqcallback`, which displays the event type and the device object name.

In most cases, data is missed because:

- The engine cannot keep up with the rate of acquisition.
- The driver wrote new data into the hardware’s FIFO buffer before the previously acquired data was read. You can usually avoid this problem by increasing the size of the memory block with the `BufferingConfig` property.

Data missed event information is stored in the `Type` and `Data` fields of the `EventLog` property. The `Type` field value is `DataMissed`. The `Data` field values are given below.

Data Field Value	Description
`AbsTime`	The absolute time (as a `clock` vector) the event occurred.
`RelSample`	The acquired sample number when the event occurred.

When a data missed event occurs, the analog input object is automatically stopped.

Characteristics

Usage	AI, common to all channels
Access	Read/write
Data type	String
Read-only when running	No

Values

The default value is `daqcallback`.

See Also

Functions

`daqcallback`

Properties

`EventLog`

DefaultChannelValue

Purpose

Specify value held by analog output subsystem

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

`DefaultChannelValue` specifies the value to write to the analog output (AO) subsystem when data is finished being output from the engine.

`DefaultChannelValue` is used only when `OutOfDataMode` is set to `DefaultValue`. This property guarantees that a known value is held by the AO subsystem if a run-time error occurs. Note that sound cards do not have an `OutOfDataMode` property.

Characteristics

Usage	AO, per channel
Access	Read/write
Data type	Double
Read-only when running	Yes

Values

The default value is zero.

Examples

Create the analog output object `ao` and add two channels to it.

```
ao = analogoutput('nidaq','Dev1');
addchannel(ao,0:1);
```

You can configure `ao` so that when it stops outputting data, a value of 1 volt is held for both channels.

```
ao.OutOfDataMode = 'DefaultValue';
ao.Channel.DefaultChannelValue = 1.0;
```

See Also

Properties

`OutOfDataMode`

Destination

Purpose

Indicates trigger destination terminal

Description

When working with the session-based interface, the `Source` property indicates the device and terminal to which you connect a trigger.

See Also

`Sourcedaq.Session.addTriggerConnection`

Purpose

Channel device information

Description

When working with the session-based interface, the read-only `Device` property displays device information for the channel.

Examples

Create a session object, add a counter input channel, and view the `Device` property.

```
s = daq.createSession('ni');
 ch = s.addCounterInputChannel('cDAQ1Mod5', 0, 'EdgeCount');
ch.Device

ans =

ni cDAQ1Mod5: National Instruments NI 9402
   Counter input subsystem supports:
      Rates from 0.1 to 80000000.0 scans/sec
      2 channels
      'EdgeCount','PulseWidth','Frequency','Position' measurement types

   Counter output subsystem supports:
      Rates from 0.1 to 80000000.0 scans/sec
      3 channels
      'PulseGeneration' measurement type

This module is in chassis 'cDAQ1', slot 5
```

See Also

Class

`daq.Session`, `daq.Session.addCounterInputChannel`, `daq.Session.addCounterOutputChannel`

Direction property

Purpose

Specify digital channel direction

Description

When you add a digital channel or a group to a session, you can specify the measurement type to be:

- `Input`
- `Output`
- `Unknown`

When you specify the `MeasurementType` as `Bidirectional`, you can use the channel to input and output messges. By default the channel is set to `Unknown`. Change the direction to output singnal on the channel.

Example

To change the direction of a bidirectional signal on a digital channel in the session `s`, type:

```
s.Channels(1).Direction='Output';
```

Purpose

Specify whether line is for input or output

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

When adding hardware lines to a digital I/O object with `addline`, you must configure the line direction. The line direction can be `In` or `Out`, and is automatically stored in `Direction`. If a line direction is `In`, you can only read a value from that line. If a line direction is `Out`, you can write or read a line value.

For line-configurable devices, you can change individual line directions using `Direction`. For port-configurable devices, you cannot change individual line directions.

Characteristics

Usage	DIO, per line
Access	Read/write
Data type	String
Read-only when running	Yes

Values

`{In}`	The line can be read from.
`Out`	The line can be read from or written to.

Examples

Create the digital I/O object `dio` and add two input lines and two output lines to it.

```
dio = digitalio('nidaq','Dev1');
addline(dio,0:3,{'In','In','Out','Out'});
```

To configure all lines for output:

```
dio.Line(1:2).Direction = 'Out';
```

See Also

Functions

addline

Purpose

Specify duration of acquisition

Description

When working with the session-based interface, use the `DurationInSeconds` property to change the duration of an acquisition. When the session contains output channels, `DurationInSeconds` becomes a read only property and its value is determined by

$$\frac{s.ScansQueued}{s.Rate}.$$

Values

In a session with only input channels, you can enter a value in seconds for the length of the acquisition. Changing the duration changes the number of scans accordingly. By default, the `DurationInSeconds` is set to 1 second.

Examples

Create a session object, add an analog input channel, and change the duration:

```
s = daq.createSession('ni');
s.addAnalogInputChannel('cDAQ1Mod1','ai0','voltage');
s.DurationInSeconds = 2

s =

Data acquisition session using National Instruments hardware:
   Will run for 2 seconds (2000 scans) at 1000 scans/second.
   Operation starts immediately.
      Number of channels: 1
      index Type  Device   Channel InputType      Range       Name
      ----- ---- --------- ------- --------- ---------------- ----
      1     ai   cDAQ1Mod1 ai0     Diff      -10 to +10 Volts
```

See Also

Properties

`NumberOfScans`, `Rate`

Class

`daq.Session`, `daq.Session.addCounterInputChannel`

Purpose

Duty cycle of counter output channel

Description

When working with the session-based interface, use the `DutyCycle` property to specify the fraction of time that the generated pulse is in active state.

Duty cycle is the ratio between the duration of the pulse and the pulse period. For example, if a pulse duration is 1 microsecond and the pulse period is 4 microseconds, the duty cycle is 0.25. In a square wave, you will see that the time the signal is high is equal to the time the signal is low.

Examples

Create a session object and add a `'PulseGeneration'` counter output channel:

```
s = daq.createSession('ni');
s.addCounterOutputChannel('cDAQ1Mod5', 'ctr0', 'PulseGeneration')
```

Change the `DutyCycle` to `0.25` and display the channel:

```
s.Channels.Frequency = 200;

s.Channels

ans =

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

       IdleState: Low
    InitialDelay: 2.5e-008
       Frequency: 100
       DutyCycle: 0.25
        Terminal: 'PFI12'
            Name: empty
              ID: 'ctr0'
          Device: [1x1 daq.ni.DeviceInfo]
 MeasurementType: 'PulseGeneration'
```

See Also

Class

`daq.Session`, `daq.Session.addCounterOutputChannel`

EncoderType

Purpose

Encoding type of counter channel

Description

When working with the session-based interface, use the `EncoderType` property to specify the encoding type of the counter input `'Position'` channel.

Encoder types include:

- `'X1'`
- `'X2'`
- `'X4'`
- `'TwoPulse'`

See Also

Class

`daq.Session`, `daq.Session.addCounterInputChannel`

Purpose

Set enhanced alias rejection mode

Description

Enable or disable the enhanced alias rejection on your DSA device's analog channel. See "Synchronize DSA Devices" for more information. Enhanced alias reject is disabled by default. This property only takes logical values. To enable it, type:

```
s.Channels(1).EnhancedAliasRejectionEnable = 1
```

You cannot modify enhanced rejection mode if you are synchronizing your DSA device using `AutoSyncDSA`.

See Also

`AutoSyncDSA`

Purpose

Store information for specific events

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`Eventlog` is a structure array that stores information related to specific analog input (AI) or analog output (AO) events. Event information is stored in the `Type` and `Data` fields of `EventLog`. `Type` stores the event type. The logged event types are shown below.

Event Type	Description	AI	AO
Data missed	Data is missed by the engine.	✓	
Input overrange	A signal exceeds the hardware input range.	✓	
Run-time error	A run-time error is encountered. Run-time errors include timeouts and hardware errors.	✓	✓
Start	The `start` function is issued.	✓	✓
Stop	The device object stops executing.	✓	✓
Trigger	A trigger executes.	✓	✓

Timer events, samples available events (AI), and samples output events (AO) are not logged.

`Data` stores event-specific information associated with the event type in several fields. For all stored events, `Data` contains the `RelSample` field, which returns the input or output sample number at the time the event occurred. For the start, stop, run-time error, and trigger events, `Data` contains the `AbsTime` field, which returns the absolute time (as a `clock` vector) the event occurred. Other event-specific fields are included in `Data`. For a description of these fields, refer to "Events and Callbacks" for analog input objects and analog output objects, or the appropriate reference pages in this chapter.

`EventLog` can store a maximum of 1000 events. If this value is exceeded, then the most recent 1000 events are stored. You can use the `showdaqevents` function to easily display stored event information.

Characteristics

Usage	AI, AO, common to all channels
Access	Read-only
Data type	Structure array
Read-only when running	N/A

Values

Values are automatically added as events occur. The default value is an empty structure array.

Examples

Create the analog input object `ai` and add four channels to it.

```
ai = analoginput('nidaq','Dev1');
chans = addchannel(ai,0:3);
```

Acquire 1 second of data and display the logged event types.

```
start(ai)
events = ai.EventLog;
{events.Type}
ans =
    'Start'    'Trigger'    'Stop
```

To examine the data associated with the trigger event:

```
events(2).Data
ans =
      AbsTime: [1999 2 12 14 54 52.5456]
    RelSample: 0
```

```
Channel: []
Trigger: 1
```

See Also

Functions

showdaqevents

Purpose Voltage of external source of excitation

Description When working with the session-based interface, the `ExcitationCurrent` property indicates the current in ams that you use to excite an IEPE accelerometer, IEPE microphone, generic IEPE sensors, and RTDs.

The default `ExcitationCurrent` is typically determined by the device. If the device supports an range of excitation currents, the default will be the lowest available value in the range.

See Also

Properties

`ExcitationSource`

Class

`daq.Session` `daq.Session.addAnalogInputChannel`

ExcitationSource

Purpose

External source of excitation

Description

When working with the session-based interface, the `ExcitationSource` property indicates the source of `ExcitationVoltage` for bridge measurements or `ExcitationCurrent` for IEPE sensors and RTDs. Excitation source can be:

- `Internal`
- `External`
- `None`
- `Unknown`

By default, `ExcitationSource` is set to `Unknown`.

See Also

Properties

`ExcitationCurrent`

`ExcitationVoltage`

Class

`daq.Session` `daq.Session.addAnalogInputChannel`

Purpose

Voltage of excitation source

Description

When working with RTD measurements in the session-based interface, the `ExcitationVoltage` property indicates the excitation voltage value to apply to bridge measurements.

The default `ExcitationVoltage` is typically determined by the device. If the device supports a range of excitation voltages, the default will be the lowest avaiulable value in the range.

See Also

Properties

`ExcitationSource`

Class

`daq.Session`

ExternalTriggerTimeout

Purpose Indicate if external trigger timed out

Description When working with the session-based interface, the `ExternalTriggerTimeout` property indicates if the if an external trigger timed out.

See Also `daq.Session.addTriggerConnection`

Purpose

Frequency of generated pulses on counter output channel

Description

When working with the session-based interface, use the `Frequency` property to set the pulse repetition rate of a counter input channel .

Values

Specify the frequency in hertz.

Examples

Create a session object and add a `'PulseGeneration'` counter output channel:

```
s = daq.createSession('ni');
s.addCounterOutputChannel('cDAQ1Mod5', 'ctr0', 'PulseGeneration')
```

Change the `Frequency` to `200` and display the channel:

```
s.Channels.Frequency = 200;

s.Channels

ans =

Data acquisition counter output pulse generation channel 'ctr0' on dev

       IdleState: Low
    InitialDelay: 2.5e-008
       Frequency: 200
       DutyCycle: 0.5
        Terminal: 'PFI12'
            Name: empty
              ID: 'ctr0'
          Device: [1x1 daq.ni.DeviceInfo]
 MeasurementType: 'PulseGeneration'
```

See Also

Class

`daq.Session`, `daq.Session.addCounterInputChannel`

HwChannel

Purpose

Specify hardware channel ID

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

All channels contained by a device object have a hardware channel ID and an associated MATLAB index. The channel ID is given by `HwChannel` and the MATLAB index is given by the `Index` property. The `HwChannel` value is defined when hardware channels are added to a device object with the `addchannel` function.

The beginning channel ID value depends on the hardware device. For National Instruments hardware, channel IDs are zero-based (begin at zero). For sound cards, channel IDs are one-based (begin at one).

For scanning hardware, the scan order follows the MATLAB index. Therefore, the hardware channel associated with index 1 is sampled first, the hardware channel associated with index 2 is sampled second, and so on. To change the scan order, you can assign the channel IDs to different indices using `HwChannel`.

Characteristics

Usage	AI, AO, per channel
Access	Read/write
Data type	Double
Read-only when running	Yes

Values

Values are automatically defined when channels are added to the device object with the `addchannel` function. The default value is one.

Examples

Create the analog input object `ai` for a National Instruments board and add the first three hardware channels to it.

```
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:2);
```

Based on the current configuration, the hardware channels are scanned in order from 0 to 2. To swap the scan order of channels 0 and 1, you can assign these channels to the appropriate indices using `HwChannel`.

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

See Also

Functions

`addchannel`

Properties

`Channel`, `Index`

Purpose

Specify hardware line ID

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

All lines contained by a digital I/O object have a hardware ID and an associated MATLAB index. The hardware ID is given by `HwLine` and the MATLAB index is given by the `Index` property. The `HwLine` value is defined when hardware lines are added to a digital I/O object with the `addline` function.

The beginning line ID value depends on the hardware device. For National Instruments hardware, line IDs are zero-based (begin at zero).

Characteristics

Usage	DIO, per line
Access	Read/write
Data type	Double
Read-only when running	Yes

Values

Values are automatically defined when lines are added to the digital I/O object with the `addline` function. The default value is one.

Examples

Suppose you create the digital I/O object `dio` and add four hardware lines to it.

```
dio = digitalio('nidaq','Dev1');
addline(dio,0:3,'out');
```

`addline` automatically assigns the indices 1-4 to these hardware lines. You can swap the hardware lines associated with index 1 and index 2 with `HwLine`.

```
dio.Line(1).HwLine = 1;
```

```
dio.Line(2).HwLine = 0;
```

See Also

Functions

`addline`

Properties

`Line`, `Index`

ID

Purpose

ID of channel in session

Description

When working with the session-based interface, the `ID` property displays the ID of the channel. You set the channel ID when you add the channel to a session object.

Values

Examples

Create a session object, add a counter input channel, with the ID `'ctr0'`.

```
s = daq.createSession('ni');
ch = s.addCounterInputChannel ('cDAQ1Mod5', 'ctr0', 'EdgeCount')

ch =

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

      ActiveEdge: Rising
  CountDirection: Increment
    InitialCount: 0
        Terminal: 'PFI8'
            Name: empty
              ID: 'ctr0'
          Device: [1x1 daq.ni.DeviceInfo]
 MeasurementType: 'EdgeCount'
```

Change `CountDirection` to `'Decrement'`:

```
ch.CountDirection = 'Decrement'

ch =

Data acquisition counter input edge count channel 'ctr0' on device 'cDAQ1

      ActiveEdge: Rising
  CountDirection: Decrement
```

```
    InitialCount: 0
        Terminal: 'PFI8'
            Name: empty
              ID: 'ctr0'
          Device: [1x1 daq.ni.DeviceInfo]
 MeasurementType: 'EdgeCount'
```

See Also

Class

daq.Session

IdleState

Purpose

Default state of counter output channel

Description

When working with the session-based interface, the `IdleState` property indicates the default state of the counter output channel with a `'PulseGeneration'` measurement type when the counter is not running.

Values

IdleState is either 'High' or 'Low'.

Examples

Create a session object and add a `'PulseGeneration'` counter output channel:

```
s = daq.createSession('ni');
s.addCounterOutputChannel('cDAQ1Mod5', 'ctr0', 'PulseGeneration')
```

Change the `IdleState` property to `'High'` and display the channel:

```
s.Channels.IdleState = 'High';

s.Channels

ans =

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

       IdleState: High
    InitialDelay: 2.5e-008
       Frequency: 100
       DutyCycle: 0.5
        Terminal: 'PFI12'
            Name: empty
              ID: 'ctr0'
          Device: [1x1 daq.ni.DeviceInfo]
 MeasurementType: 'PulseGeneration'
```

See Also

Class

daq.Session, daq.Session.addCounterOutputChannel

Index

Purpose

MATLAB index of hardware channel or line

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Every hardware channel (line) contained by a device object has an associated MATLAB index that is used to reference that channel (line). For example, to configure property values for an individual channel, you must reference the channel through the `Channel` property using the appropriate `Index` value. Likewise, to configure property values for an individual line, you must reference the line through the `Line` property using the appropriate `Index` value.

For channels (lines), you can assign indices automatically with the `addchannel` (`addline`) function. Channel (line) indices always begin at 1 and increase monotonically up to the number of channels (lines) contained by the device object. For channels, index assignments can also be made manually with the `addchannel` function.

For scanning hardware, the scan order follows the MATLAB index. Therefore, the hardware channel associated with index 1 is sampled first, the hardware channel associated with index 2 is sampled second, and so on. To change the scan order, you can assign the channel IDs to different indices using the `HwChannel` or `Channel` property.

`Index` provides a convenient way to access channels and lines programmatically.

Characteristics

Usage	AI, AO, per channel; DIO, per line
Access	Read-only
Data type	Double
Read-only when running	N/A

Values

Values are automatically defined when channels (lines) are added to the device object with the `addchannel` (`addline`) function. The default value is one.

Examples

Create the analog input object `ai` for a sound card and add two hardware channels to it.

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

You can access the MATLAB indices for these channels with `Index`.

```
Index1 = chans(1).Index;
Index2 = chans(2).Index;
```

See Also

Functions

`addchannel`, `addline`

Properties

`Channel`, `HwChannel`, `HwLine`, `Line`

InitialDelay

Purpose

Delay until output channel generates pulses

Description

When working with the session-based interface, use the `InitialDelay` property to set an initial delay on the counter output channel in which the counter is running but does not generate any pulse.

See Also

Class

`daq.Session`

Purpose Specify initial count point

Description When working with the session-based interface, use the `InitialCount` property to set the point from which the device starts the counter.

Values

Examples Create a session object, add counter input channel, and change the `InitialCount`.

```
s = daq.createSession('ni');
ch = s.addCounterInputChannel ('cDAQ1Mod5', 0, 'EdgeCount')

ch =

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

      ActiveEdge: Rising
  CountDirection: Increment
    InitialCount: 0
        Terminal: 'PFI8'
            Name: empty
              ID: 'ctr0'
          Device: [1x1 daq.ni.DeviceInfo]
 MeasurementType: 'EdgeCount'
```

Change `InitalCount` to `15`:

```
 ch.InitialCount = 15

ch =

Data acquisition counter input edge count channel 'ctr0' on device 'cD

      ActiveEdge: Rising
   CountDirection: Increment
    InitialCount: 15
```

```
          Terminal: 'PFI8'
              Name: empty
                ID: 'ctr0'
            Device: [1x1 daq.ni.DeviceInfo]
   MeasurementType: 'EdgeCount'
```

See Also

Properties

Class

daq.Session

Purpose

Absolute time of first trigger

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

For all trigger types, `InitialTriggerTime` records the time when `Logging` or `Sending` is set to `On`. The absolute time is recorded as a `clock` vector.

You can return the `InitialTriggerTime` value with the `getdata` function, or with the `Data.AbsTime` field of the `EventLog` property.

If you synchronize multiple `analoginput` and `analogoutput` objects by setting `TriggerType` to `HwDigitalTrigger` and use the same digital trigger signal for all of the subsystems, the `InitialTriggerTime` property will not show the exact identical time for all subsystems.

Although the actual trigger events occurred simultaneously across all subsystems, the `InitialTriggerTime` events are recorded serially on a single thread. This causes the discrepancy of a few milliseconds. The time difference between `InitialTriggerTime` for multiple Data Acquisition Toolbox objects will not be consistent due to operating system process scheduling algorithms.

Characteristics

Usage	AI, AO, common to all channels
Access	Read-only
Data type	Six-element vector of doubles
Read-only when running	N/A

Values

The value is automatically updated when the trigger executes. The default value is a vector of zeros.

Examples

Create the analog input object ai for a sound card and add two hardware channels to it.

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

After starting ai, the trigger immediately executes and the trigger time is recorded.

```
start(ai)
abstime = ai.InitialTriggerTime
abstime =
1.0e+003 *
    1.9990    0.0020    0.0190    0.0130    0.0260    0.0208
```

To convert the clock vector to a more convenient form:

```
t = fix(abstime);
sprintf('%d:%d:%d', t(4),t(5),t(6))
ans =
13:26:20
```

See Also

Functions

getdata

Properties

EventLog, Logging, Sending

Purpose

Specify callback function to execute when acquired data exceeds valid hardware range

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

An input overrange event is generated immediately after an overrange condition is detected for any channel group member. This event executes the callback function specified for `InputOverRangeFcn`.

An overrange condition occurs when an input signal exceeds the range specified by the `SensorRange` property. Overrange detection is enabled only if the analog input object is running and a callback function is specified for `InputOverRangeFcn`.

Input overrange event information is stored in the `Type` and `Data` fields of the `EventLog` property. The `Type` field value is `OverRange`. The `Data` field values are given below.

Note The input overrange event is not generated if a signal begins outside the range and then goes into the range.

Data Field Value	Description
`AbsTime`	The absolute time (as a `clock` vector) the event occurred.
`Channel`	The index of the channel that experienced an overrange signal.

Data Field Value	Description
OverRange	The OverRange value, Off indicates that the channel went from overrange to in range, and On indicates that it went from in range to overrange.
RelSample	The acquired sample immediately before the moment when the overrange transition occurs.

Note The input signal values will not exceed the values set by the InputRange property. If you set InputRange and SensorRange to the same value, the OverRange event is never received. To receive OverRange events set the value of SensorRange within, and not equal to, the InputRange value.

Characteristics

Usage	AI, common to all channels
Access	Read/write
Data type	String
Read-only when running	No

Values

The default value is an empty string.

See Also

Properties

EventLog, SensorRange

Purpose

Specify range of analog input subsystem

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`InputRange` is a two-element vector that specifies the range of voltages that can be accepted by the analog input (AI) subsystem. You should configure `InputRange` so that the maximum dynamic range of your hardware is utilized.

If an input signal exceeds the `InputRange` value, then an overrange condition occurs. Overrange detection is enabled only if the analog input object is running and a value is specified for the `InputOverRangeFcn` property. For many devices, the input range is expressed in terms of the gain and polarity.

AI subsystems have a finite number of `InputRange` values that you can set. If an input range is specified but does not match a valid range, then the next highest supported range is automatically selected by the engine. If `InputRange` exceeds the range of valid values, then an error is returned. Use the `daqhwinfo` function to return the input ranges supported by your board.

Because the engine can set the input range to a value that differs from the value you specify, you should return the actual input range for each channel using the `get` function or the device object display summary. Alternatively, you can use the `setverify` function, which sets the `InputRange` value and then returns the actual value that is set.

Note If your hardware supports a channel gain list, then you can configure `InputRange` for individual channels. Otherwise, `InputRange` must have the same value for all channels contained by the analog input object.

You should use `InputRange` in conjunction with the `SensorRange` property. These two properties should be configured such that the maximum precision is obtained and the full dynamic range of the sensor signal is covered.

Characteristics

Usage	AI, per channel
Access	Read/write
Data type	Two-element vector of doubles
Read-only when running	Yes

Values

The default value is supplied by the hardware driver.

Examples

Create the analog input object `ai` for a National Instruments board, and add two hardware channels to it.

```
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:1);
```

You can return the input ranges supported by the board with the `InputRanges` field of the `daqhwinfo` function.

```
out = daqhwinfo(ai);
out.InputRanges
ans =
   -0.0500    0.0500
   -0.5000    0.5000
   -5.0000    5.0000
  -10.0000   10.0000
```

To configure both channels contained by `ai` to accept input signals between -10 volts and 10 volts:

```
ai.Channel.InputRange = [-10 10];
```

Some devices allow you to set each channel's `InputRange` property independently:

```
ai.Channel(1).InputRange = [-0.05    0.05];
ai.Channel(2).InputRange = [-10 10];
```

Alternatively, you can use the `setverify` function.

```
ActualRange = setverify(ai.Channel,'InputRange',[-10 10]);
```

See Also

Functions

`daqhwinfo`, `setverify`

Properties

`InputOverRangeFcn`, `SensorRange`, `Units`, `UnitsRange`

InputType

Purpose

Specify analog input hardware channel configuration

Description

For National Instruments devices, `InputType` can be `SingleEnded`, `Differential`, `NonReferencedSingleEnded`, or `PseudoDifferential`. For Measurement Computing devices, `InputType` can be `SingleEnded`, or `Differential`. For sound cards, `InputType` can only be `AC-Coupled`.

If channels have been added to a National Instruments or Measurement Computing analog input object and you change the `InputType` value, then the channels are automatically deleted if the hardware reduces the number of available channels.

Characteristics

Usage	AI, common to all channels
Access	Read/write
Data type	String
Read-only when running	Yes

Values

Advantech and Measurement Computing

`Differential`	Channels are configured for differential input.
`SingleEnded`	Channels are configured for single-ended input.

The value for `InputType` on Advantech and MCC boards is always read-only in MATLAB. For Advantech boards, the setting is made in the Advantech Device Manager. For Measurement Computing boards, the setting is made in InstaCal.

National Instruments

`{Differential}`	Channels are configured for differential input.
`SingleEnded`	Channels are configured for single-ended input.
`NonReferencedSingleEnded`	This channel configuration is used when the input signal has its own ground reference, which is tied to the negative input of the instrumentation amplifier.
`PseudoDifferential`	Channels are configured for pseudodifferential input, which are all referred to a common ground but this ground is not connected to the computer ground.

Sound Cards

`{AC-Coupled}`	The input is coupled so that constant (DC) signal levels are suppressed.

IsContinuous

Purpose

Specify if operation continues until manually stopped

Description

When working with the session-based interface, use `IsContinuous` to specify that the session operation runs until you execute `daq.Session.stop`. When set to `true`, the session will run continuously, acquiring or generating data until stopped.

Values

`{false}`
: Set the `IsContinuous` property to `false` to make the session operation stop automatically. This property is set to `false` by default.

`true`
: Set the `IsContinuous` property to `true` to make the session operation run until you execute `daq.Session.stop`.

Examples

Create a session object, add an analog input channel, and set the session to run until manually stopped:

```
s = daq.createSession('ni');
s.addAnalogInputChannel('cDAQ1Mod1','ai0','voltage');
s.IsContinuous = true


s =


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

See Also

Properties

`IsDone`

Methods

`daq.Session.stop`, `daq.Session.startBackground`

Class

`daq.Session`

IsDone

Purpose

Indicate if operation is complete

Description

When working with the session-based interface, the read-only `IsDone` property indicates if the session operation is complete.

Tip `IsDone` indicates if the session object has completed acquiring or generating data. `IsRunning` indicates if the operation is in progress, but the hardware may not be acquiring or generating data. `IsLogging` indicates that the hardware is acquiring or generating data.

Values

`true`
: Value is `true` if the operation is complete.

`false`
: Value is `false` if the operation is not complete.

Examples

Create an acquisition session and see if the operation is complete:

```
s = daq.createSession('ni');
s.addAnalogOutputChannel('cDAQ1Mod2', 'ao1', 'vVoltage');
s.queueOutputData (linspace(-1, 1, 1000)');
s.startBackground();
s.IsDone

ans =

     0
```

Issue a wait and see if the operation is complete:

```
s.wait()
s.IsDone

ans =
```

```
1
```

See Also

Methods

`daq.Session.startBackground`

Class

`daq.Session`

IsLogging

Purpose

Indicate if hardware is acquiring or generating data

Description

When working with the session-based interface, the status of the read-only `IsLogging` property indicates if the hardware is acquiring or generating data.

Tip `IsLogging` indicates that the hardware is acquiring or generating data. `IsRunning` indicates if the operation is in progress, but the hardware may not be acquiring or generating data. `IsDone` indicates if the session object has completed acquiring or generating data.

Values

`true`
: Value is `true` if the device is acquiring or generating data.

`false`
: Value is `false` if the device is not acquiring or generating data.

Examples

Create a session and see if the operation is logging:

```
 s = daq.createSession('ni');
 s.addAnalogOutputChannel('cDAQ1Mod2', 'ao1', 'Voltage');
 s.queueOutputData (linspace(-1, 1, 1000)');
 s.startBackground();
 s.IsLogging

ans =

     1
```

Wait until the operation is complete:

```
s.wait()
s.IsLogging

ans =
```

```
0
```

See Also

Properties

`IsRunning`, `IsDone`

Methods

`daq.Session.startBackground`

Class

`daq.Session`

IsNotifyWhenDataAvailableExceedsAuto

Purpose

Control if `NotifyWhenDataAvailableExceeds` is set automatically

Description

When working with the session-based interface, the `IsNotifyWhenDataAvailableExceedsAuto` property indicates if the `NotifyWhenDataAvailableExceeds` property is set automatically, or you have set a specific value.

Tip This property is typically used to set `NotifyWhenDataAvailableExceeds` back to its default behavior.

Values

`{true}`
: When the value is `true`, then the `NotifyWhenDataAvailableExceeds` property is set automatically.

`false`
: When the value is `false`, when you have set the `NotifyWhenDataAvailableExceeds` property to a specific value.

See Also

Properties

`NotifyWhenDataAvailableExceeds`

Events

`DataAvailable`

Class

`daq.Session`

Purpose

Control if `NotifyWhenScansQueuedBelow` is set automatically

Description

When working with the session-based interface, the `IsNotifyWhenScansQueuedBelowAuto` property indicates if the `NotifyWhenScansQueuedBelow` property is set automatically, or you have set a specific value.

Values

`{true}`
: When the value is `true`, then `NotifyWhenScansQueuedBelow` is set automatically.

`false`
: When the value is `false`, you have set `NotifyWhenScansQueuedBelow` property to a specific value.

See Also

Properties

`NotifyWhenScansQueuedBelow`, `ScansQueued`

Events

`DataRequired`

Class

`daq.Session`

IsRunning

Purpose

Indicate if operation is still in progress

Description

When working with the session-based interface, the `IsRunning` status indicates if the operation is still in progress.

Tip `IsRunning` indicates if the operation is in progress, but the hardware may not be acquiring or generating data. `IsLogging` indicates that the hardware is acquiring or generating data. `IsDone` indicate is if the session object has completed acquiring or generating.

Values

`true`
: When the value is `true` if the operation is in progress.

`false`
: When the value is `false` if the operation is not in progress.

Examples

Create an acquisition session, add a `DataAvailable` event listener and start the acquisition.

```
s = daq.createSession('ni');
s.addAnalogInputChannel('cDAQ1Mod1','ai0','voltage');
lh = s.addlistener('DataAvailable', @plotData);

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

s.startBackground();
```

See if the session is in progress.

```
s.IsRunning

ans =

    1
```

Wait until operation completes and see if session is in progress:

```
 s.wait()
s.IsRunning

ans =

     0
```

See Also

Properties

`IsLogging`, `IsDone`

Methods

`daq.Session.startBackground`

Class

`daq.Session`

IsSimulated

Purpose Indicate if device is simulated

Description When working with the session-based interface, the `IsSimulated` property indicates if the session is using a simulated device.

Values

`true`
: When the value is `true` if the operation is in progress.

`false`
: When the value is `false` if the operation is not in progress.

Examples Discover available devices.

```
>> d = daq.getDevices

d =

Data acquisition devices:

index Vendor Device ID           Description
----- ------ --------- -----------------------------
1     ni     cDAQ1Mod1 National Instruments NI 9201
2     ni     cDAQ2Mod1 National Instruments NI 9201
3     ni     Dev1      National Instruments USB-6211
4     ni     Dev2      National Instruments USB-6218
5     ni     Dev3      National Instruments USB-6255
6     ni     Dev4      National Instruments USB-6363
7     ni     PXI1Slot2 National Instruments PXI-4461
8     ni     PXI1Slot3 National Instruments PXI-4461
```

Examine properties of NI 9201, with the device id cDAQ1Mod1 with the index 1.

```
>> d(1)

ans =
```

```
ni: National Instruments NI 9201 (Device ID: 'cDAQ1Mod1')
   Analog input subsystem supports:
      -10 to +10 Volts range
      Rates from 0.1 to 800000.0 scans/sec
      8 channels ('ai0','ai1','ai2','ai3','ai4','ai5','ai6','ai7')
      'Voltage' measurement type

This module is in slot 4 of the 'cDAQ-9178' chassis with the name 'cDA

Properties, Methods, Events
```

Click the `Properties` link to see the properties of the device.

```
  ChassisName: 'cDAQ1'
    ChassisModel: 'cDAQ-9178'
      SlotNumber: 4
     IsSimulated: true
       Terminals: [48x1 cell]
          Vendor: National Instruments
              ID: 'cDAQ1Mod1'
           Model: 'NI 9201'
      Subsystems: [1x1 daq.ni.AnalogInputInfo]
     Description: 'National Instruments NI 9201'
RecognizedDevice: true
```

Note that the `IsSimulated` value is `true`, indicating that this device is simulated.

See Also

Properties

`IsLogging`, `IsDone`

Methods

`daq.Session.startBackground`

Class

`daq.Session`

IsWaitingForExternalTrigger

Purpose

Indicates if synchronization is waiting for an external trigger

Description

When working with the session-based interface, the read-only`IsWaitingForExternalTrigger` property indicates if the acquisition or generation session is waiting for a trigger from an external device. If you have added an external trigger, this property displays `true`, if not, it displays `false`.

See Also

`daq.Session.addTriggerConnection`

Purpose

Contain hardware lines added to device object

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`Line` is a vector of all the hardware lines contained by a digital I/O (DIO) object. Because a newly created DIO object does not contain hardware lines, `Line` is initially an empty vector. The size of `Line` increases as lines are added with the `addline` function, and decreases as lines are removed with the `delete` function.

You can use `Line` to reference one or more individual lines. To reference a line, you must know its MATLAB index and hardware ID. The MATLAB index is given by the `Index` property, while the hardware ID is given by the `HwLine` property.

Characteristics

Usage	DIO
Access	Read/write
Data type	Vector of lines
Read-only when running	Yes

Values

Values are automatically defined when lines are added to the DIO object with the `addline` function. The default value is an empty column vector.

Examples

Create the digital I/O object `dio` and add four input lines to it.

```
dio = digitalio('nidaq','Dev1');
addline(dio,0:3,'In');
```

To set a property value for the first line added (ID = 0), you can reference the line by its index using the `Line` property.

```
line1 = dio.Line(1);
set(line1,'Direction','Out')
```

See Also

Functions

`addline`, `delete`

Properties

`HwLine`, `Index`

Purpose

Specify descriptive line name

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`LineName` specifies a descriptive name for a hardware line. If a line name is defined, then you can reference that line by its name. If a line name is not defined, then the line must be referenced by its index. Line names are not required to be unique.

You can also define descriptive line names when lines are added to a digital I/O object with the `addline` function.

Characteristics

Usage	DIO, per line
Access	Read/write
Data type	String
Read-only when running	Yes

Values

The default value is an empty string. To reference a line by name, it must contain only letters, numbers, and underscores and must begin with a letter.

Examples

Create the digital I/O object `dio` and add four hardware lines to it.

```
dio = digitalio('nidaq','Dev1');
addline(dio,0:3,'out');
```

To assign a descriptive name to the first line contained by `dio`:

```
line1 = dio.Line(1);
set(line1,'LineName','Joe')
```

You can now reference this line by name instead of index.

```
set(dio.Joe,'Direction','In')
```

See Also

Functions

addline

Purpose

Specify name of disk file information is logged to

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

You can log acquired data, device object property values and event information, and hardware information to a disk file by setting the `LoggingMode` property to `Disk` or `Disk&Memory`.

You can specify any value for `LogFileName` as long as it conforms to the MATLAB software naming conventions: the name cannot start with a number and cannot contain spaces. If no extension is specified as part of `LogFileName`, then `daq` is used. The default value for `LogFileName` is `logfile.daq`.

You can choose whether an output file is overwritten or if multiple log files are created with the `LogToDiskMode` property. Setting `LogToDiskMode` to `Overwrite` causes the output file to be overwritten. Setting `LogToDiskMode` to `Index` causes new data files to be created, each with an indexed name based on the value of Log`FileName`.

Characteristics

Usage	AI, common to all channels
Access	Read/write
Data type	String
Read-only when running	Yes

Values

The default value is `logfile.daq`.

See Also

Properties

`Logging`, `LoggingMode`, `LogToDiskMode`

Logging

Purpose

Indicate whether data is being logged to memory or disk file

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

Along with the `Running` property, `Logging` reflects the state of an analog input object. `Logging` can be `On` or `Off`.

`Logging` is automatically set to `On` when a trigger occurs. When `Logging` is `On`, acquired data is being stored in memory or to a disk file.

`Logging` is automatically set to `Off` when the requested samples are acquired, an error occurs, or a `stop` function is issued. When `Logging` is `Off`, you can still preview data with the `peekdata` function provided `Running` is `On`. However, `peekdata` does not guarantee that all the requested data is returned.

To guarantee that acquired data contains no gaps, is must be logged to memory or to a disk file. Data stored in memory is extracted with the `getdata` function, while data stored to disk is returned with the `daqread` function. The destination for logged data is controlled with the `LoggingMode` property.

Characteristics

Usage	AI, common to all channels
Access	Read-only
Data type	String
Read-only when running	N/A

Values

`{Off}`	Data is not logged to memory or a disk file.
`On`	Data is logged to memory or a disk file.

See Also

Functions

`daqread`, `getdata`, `peekdata`, `stop`

Properties

`LoggingMode`, `Running`

LoggingMode

Purpose

Specify destination for acquired data

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

You can set `LoggingMode` to `Disk`, `Memory`, or `Disk&Memory`. When you set `LoggingMode` to `Disk`, then acquired data (as well as device object and hardware information) is streamed to a disk file. If `LoggingMode` is set to `Memory`, then acquired data is stored in the data acquisition engine. If `LoggingMode` is set to `Disk&Memory`, then acquired data is stored in the data acquisition engine and is streamed to a disk file.

When logging to the engine, you must extract the data with the `getdata` function. If you do not extract this data, and the amount of data stored in memory reaches the limit for the data acquisition object (see `daqmem`(obj)), a **DataMissed** event occurs. At this point, the acquisition stops.

When logging to disk, you can specify the log filename with the `LogFileName` property, and you can control the number of log files created with the `LogToDiskMode` property. You can return data stored in a disk file to the MATLAB workspace with the `daqread` function.

Characteristics

Usage	AI, common to all channels
Access	Read/write
Data type	String
Read-only when running	Yes

Values

`Disk`	Acquired data is logged to a disk file.
`{Memory}`	Acquired data is logged to memory.
`Disk&Memory`	Acquired data is logged to a disk file and to memory.

See Also

Functions

`daqread`, `getdata`

Properties

`LogFileName`, `LogToDiskMode`

LogToDiskMode

Purpose

Specify whether data, events, and hardware information are saved to one or more disk files

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`LogToDiskMode` can be set to `Overwrite` or `Index`. If `LogToDiskMode` is set to `Overwrite`, then the log file is overwritten each time `start` is issued. If `LogToDiskMode` is set to `Index`, a different disk file is created each time `start` is issued and these rules are followed:

- The first log filename is specified by the initial value of `LogFileName`.
- If the specified file already exists, it is overwritten and no warning is issued.
- `LogFileName` is automatically updated with a numeric identifier after each file is written. For example, if `LogFileName` is initially specified as `data.daq`, then `data.daq` is the first filename, `data01.daq` is the second filename, and so on.

Separate analog input objects are logged to separate files. You can return data stored in a disk file to the MATLAB workspace with the `daqread` function. If an error occurs during data logging, an error message is returned and data logging is stopped.

Characteristics

Usage	AI, common to all channels
Access	Read/write
Data type	String
Read-only when running	Yes

Values

`Index`	Multiple log files are written, each with an indexed filename based on the `LogFileName` property.
`{Overwrite}`	The log file is overwritten.

See Also

Functions

`daqread`

Properties

`LogFileName`, `LoggingMode`

ManualTriggerHwOn

Purpose

Specify hardware device starts at manual trigger

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

You can set `ManualTriggerHwOn` to `Start` or `Trigger`, and it has an effect only when the `TriggerType` property value is `Manual`. If `ManualTriggerHwOn` is `Start`, then the hardware device associated with your device object starts running after you issue the `start` function. If `ManualTriggerHwOn` is `Trigger`, then the hardware device associated with your device object starts acquiring after you issue both the `start` function and you execute a manual trigger with the `trigger` function. You can use `trigger` only when you configure the `TriggerType` property to `Manual`.

You should configure `ManualTriggerHwOn` to `Trigger` when you want to synchronize the input and output of data, or you require more control over when your hardware starts. Note that you cannot use `peekdata` or acquire pretrigger data when you use this value. Additionally, you should not use this value with repeated triggers because the subsequent behavior is undefined.

Characteristics

Usage	AI, common to all channels
Access	Read/write
Data type	String
Read-only when running	Yes

Values

{Start}	Start the hardware after the `start` function is issued.
Trigger	Start the hardware after the `trigger` function is issued.

Examples

Create the analog input object `ai` and the analog output object ao for a sound card and add two channels to each device object.

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

To operate the sound card in full duplex mode, and to minimize the time between when `ai` starts and `ao` starts, you configure `ManualTriggerHwOn` to `Trigger` for `ai` and `TriggerType` to `Manual` for both `ai` and `ao`.

```
set([ai ao],'TriggerType','Manual')
ai.ManualTriggerHwOn = 'Trigger';
```

The analog input and analog output hardware devices will both start after you issue the `trigger` function. For a detailed example that uses `ManualTriggerHwOn`, refer to “Start Multiple Device Objects”.

See Also

Functions

`peekdata`, `start`, `trigger`

Properties

`TriggerType`

MaxSamplesQueued

Purpose

Indicate maximum number of samples that can be queued in engine

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`MaxSamplesQueued` indicates the maximum number of samples allowed in the analog output queue.

If the `BufferingMode` is set to `Auto`, the default value is calculated by the engine, and is based on the memory resources of your system. You can override the default value of `MaxSamplesQueued` with the `daqmem` function.

If the `BufferingMode` is set to `Manual`, `MaxSamplesQueued` is updated to indicate the maximum number of samples allowed in the analog output queue based on the number of buffers selected in `BufferingConfig`.

The value of `MaxSamplesQueued` can affect the behavior of `putdata`. For example, if the queued data exceeds the value of `MaxSamplesQueued`, then `putdata` becomes a blocking function until there is enough space in the queue to add the additional data.

Characteristics

Usage	AO, common to all channels
Access	Read-only
Data type	Double
Read-only when running	N/A

Values

The value is calculated by the data acquisition engine.

See Also

Functions

`daqmem`, `putdata`

MaxSoundPressureLevel

Purpose

Sound pressure level for microphone channels

Description

When working with the session-based interface, use the `MaxSoundPressureLevel` set the maximum sound pressure of the microphone channel in decibels.

Values

The maximum sound pressure level is based on the sensitivity and the voltage range of your device. When you sent your device `Sensitivity`, the `MaxSoundPressure` value is automatically corrected to match the specified sensitivity value and the device voltage range. You can also specify any acceptable pressure level in decibels. Refer to your microphone specifications for more information.

MeasurementType

Purpose

Type counter channel measurement

Description

When working with the session-based interface, the `MeasurementType` property displays the selected measurement type for your channel.

Values

Counter measurement types include:

- `'EdgeCount'` (input)
- `'PulseWidth'` (input)
- `'Frequency'`(input)
- `'Position'`(input)
- `'PulseGeneration'` (output)

Analog measurement types include:

- `'Voltage'` (input and output)
- `'Thermocouple'` (input)
- `'Current'` (input and output)
- `'Accelerometer'` (input)
- `'RTD'` (input)
- `'Bridge'` (input)
- `'Microphone'` (input)
- `'IEPE'` (input)

Examples

Create a session object, add a counter input channel, with the `'EdgeCount' MeasurementType`.

```
s = daq.createSession('ni');
ch = s.addCounterInputChannel ('cDAQ1Mod5', 0, 'EdgeCount')

ch =
```

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

      ActiveEdge: Rising
  CountDirection: Increment
    InitialCount: 0
        Terminal: 'PFI8'
            Name: empty
              ID: 'ctr0'
          Device: [1x1 daq.ni.DeviceInfo]
 MeasurementType: 'EdgeCount'
```

See Also

daq.Session.addAnalogInputChannel,
daq.Session.addAnalogOutputChannel,
daq.Session.addCounterInputChannel,
daq.Session.addCounterOutputChannel

Class

daq.Session

Purpose Specify descriptive name for the channel

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

When you add a channel , a descriptive name is automatically generated and stored in `Name`. The name is a concatenation the name of the adaptor, the device ID, and the device object type. You can change the value of `Name` at any time.

Values The value is defined when you add the channel.

Examples Create the analog input object `ai` for a sound card.

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

The descriptive name for `ai` is given by

```
ai.Name

ans =
winsound0-AI
```

Change the name to `WindowsSoundChannel` and access the name

```
ai.Name='WindowsSoundChannel'
```

Purpose

Specify descriptive name for the channel

Description

When you add a channel , a descriptive name is stored in `Name`. By default there is no name assigned to the channel. You can change the value of `Name` at any time.

Values

You can specify a string value for the name.

Examples

Change the name of an analog input channel

Create a session and add an analog input channel.

```
s = daq.createSession('ni');
s.addAnalogInputChannel('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: 1
      index Type Device Channel MeasurementType      Range       Name
      ----- ---- ------ ------- -------------- --------------- ----
      1     ai   Dev1   ai0     Voltage (Diff)  -10 to +10 Volts
```

Create a session object, add a counter input channel, and change the `Name`.

```
s = daq.createSession('ni');
ch = s.addCounterInputChannel ('cDAQ1Mod5', 0, 'EdgeCount')

ch =

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

      ActiveEdge: Rising
  CountDirection: Increment
    InitialCount: 0
```

```
      Terminal: 'PFI8'
          Name: empty
            ID: 'ctr0'
        Device: [1x1 daq.ni.DeviceInfo]
MeasurementType: 'EdgeCount'
```

Change `Name` to `'AI-Voltage'`:

```
s.Channels(1).Name='AI-Voltage'
```

```
s =

Data acquisition session using National Instruments hardware:
   Will run for 1 second (1000 scans) at 1000 scans/second.
   Number of channels: 1
      index Type Device Channel MeasurementType      Range           Name
      ----- ---- ------ ------- --------------- ---------------- --------
      1     ai   Dev1   ai0     Voltage (Diff)  -10 to +10 Volts AI-Volta
```

See Also

Class

`daq.Session`

Purpose

Indicate offset to use when converting between native data format and doubles

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`NativeOffset`, along with `NativeScaling`, is used to convert data between the native hardware format and doubles.

For analog input objects, you return native data from the engine with the `getdata` function. Additionally, if you log native data to a `.daq` file, then you can read back that data using the `daqread` function. The formula for converting from native data to doubles is

doubles data = (native data)(native scaling) + native offset

For analog output objects, you queue native data in the engine with the `putdata` function. The formula for converting from doubles to native data is

native data = (doubles data)(native scaling) + native offset

You return the native data type of your hardware device with the `daqhwinfo` function. Note that the `NativeScaling` value for a given channel might change if you change its `InputRange` (AI) or `OutputRange` (AO) property value.

You might want to return or queue data in native format to conserve memory and to increase data acquisition or data output speed.

Characteristics

Usage	AI, AO, per channel
Access	Read-only

Data type	Double
Read-only when running	N/A

Values

The default value is device-specific.

Examples

Create the analog input object ai for a National Instruments board, and add eight channels to it.

```
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:7);
```

Start ai, collect one second of data for each channel, and extract the data from the engine using the native format of the device.

```
start(ai)
nativedata = getdata(ai,1000,'native');
```

You can return the native data type of the board with the daqhwinfo function.

```
out = daqhwinfo(ai);
out.NativeDataType
ans =
double
```

Convert the data to doubles using the NativeScaling and NativeOffset properties.

```
scaling = get(ai.Channel(1),'NativeScaling');
offset = get(ai.Channel(1),'NativeOffset');
data = double(nativedata)*scaling + offset;
```

See Also

Functions

daqhwinfo, daqread, getdata, putdata

Properties

`InputRange`, `NativeScaling`, `OutputRange`

NativeScaling

Purpose

Indicate scaling to use when converting between native data format and doubles

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`NativeScaling`, along with `NativeOffset`, is used to convert data between the native hardware format and doubles.

For analog input objects, you return native data from the engine with the `getdata` function. Additionally, if you log native data to a `.daq` file, then you can read back that data using the `daqread` function. The formula for converting from native data to doubles is

doubles data = (native data)(native scaling) + native offset

For analog output objects, you queue native data in the engine with the `putdata` function. The formula for converting from doubles to native data is

native data = (doubles data)(native scaling) + native offset

You return the native data type of your hardware device with the `daqhwinfo` function. Note that the `NativeScaling` value for a given channel might change if you change its `InputRange` (AI) or `OutputRange` (AO) property value.

You might want to return or queue data in native format to conserve memory and to increase data acquisition or data output speed.

Characteristics

Usage	AI, AO, per channel
Access	Read-only

Data type	Double
Read-only when running	N/A

Values

The default value is device-specific.

See Also

Functions

`daqhwinfo`, `daqread`, `getdata`, `putdata`

Properties

`InputRange`, `NativeOffset`, `OutputRange`

NominalBridgeResistance

Purpose Resistance of sensor

Description When working with the session-based interface, the `NominalBridgeResistance` property displays the resistance of a bridge– based sensor in ohms. This value is used to calculate voltage.

You can specify any accepted positive value in ohms. The default value is `0` until you change it. You must set the resistance to use the channel.

See Also

Class

`daq.Session`

Purpose

Control firing of `DataAvailable` event

Description

When working with the session-based interface the `DataAvailable` event is fired when the scans available to the session object exceeds the value specified in the `NotifyWhenDataAvailableExceeds` property.

Values

By default the `DataAvailable` event fires when 1/10 second worth of data is available for analysis. To specify a different threshold change this property to control when `DataAvailable` fires.

Examples

Create the session and add an analog input voltage channel. Add an event listener to display the total number of scans acquired for this operation:

```
s = daq.createSession('ni');
s.addAnalogInputChannel('cDAQ1Mod4', 1, 'Voltage');
lh = s.addlistener('DataAvailable', ...
   @(src, event) disp(s.ScansAcquired));
```

The default the Rate is 1000 scans per second. The session is automatically configured to fire the `DataAvailable` notification 10 times per second.

```
s.Rate
ans =
          1000
s.NotifyWhenDataAvailableExceeds
ans =
                       100
s.IsNotifyWhenDataAvailableExceedsAuto
ans =
      1
```

Increase the Rate to 800,000 scans per second and the `DataAvailable` notification automatically fires 10 times per second:

```
s.Rate = 800000;
```

```
s.NotifyWhenDataAvailableExceeds
ans =
                80000
```

Running the acquisition causes the number of scans acquired to be displayed by the callback 10 times:

```
data = s.startForeground;
                80000
               160000
               240000
               320000
               400000
               480000
               560000
               640000
               720000
               800000
```

Increase `NotifyWhenDataAvailableExceeds` to 160,000. `NotifyWhenDataAvailableExceeds`is no longer configured automatically when the `Rate` changes.

```
s.NotifyWhenDataAvailableExceeds = 160000;
s.IsNotifyWhenDataAvailableExceedsAuto
ans =
     0
data = s.startForeground;
               160000
               320000
               480000
               640000
               800000
```

The `DataAvailable` event is fired only five times per second.

Set `IsNotifyWhenDataAvailableExceedsAuto` back to `true`. This causes `NotifyWhenDataAvailableExceeds` to set automatically when `Rate` changes.

```
s.IsNotifyWhenDataAvailableExceedsAuto = true;
s.NotifyWhenDataAvailableExceeds
ans =
                80000
s.Rate = 50000;
s.NotifyWhenDataAvailableExceeds
ans =
                 5000
```

See Also

Properties

`IsNotifyWhenDataAvailableExceedsAuto`

Events

`DataAvailable`

Class

`daq.Session`

NotifyWhenScansQueuedBelow

Purpose

Control firing of `DataRequired` event

Description

When working with the session-based interface to generate output signals continuously, the `DataRequired` event is fired when you need to queue more data. This occurs when the `ScansQueued` property drops below the value specified in the `NotifyWhenScansQueuedBelow` property.

Values

By default the `DataRequired` event fires when 1/2 second worth of data remains in the queue. To specify a different threshold, change the this property to control when `DataRequired` is fired.

See Also

Properties

`ScansQueued`, `IsNotifyWhenScansQueuedBelowAuto`

Events

`DataRequired`

Class

`daq.Session`

Purpose

Number of scans for operation when starting

Description

When working with the session-based interface, use the `NumberOfScans` property to specify the number of scans the session will acquire during the operation. Changing the number of scans changes the duration of an acquisition. When the session contains output channels, `NumberOfScans` becomes a read only property and the number of scans in a session is determined by the amount of data queued.

Tips

- To specify length of the acquisition, use `DurationInSeconds`.
- To control length of the output operation, use `daq.Session.queueOutputData`.

Values

You can change the value only when you use input channels.

Examples

Create an acquisition session, add an analog input channel, and display the `NumberOfScans`.

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

ans =

                    1000
```

Change the `NumberOfScans` property.

```
s.NumberOfScans = 2000

s =
```

```
Data acquisition session using National Instruments hardware:
   Will run for 2000 scans (2 seconds) at 1000 scans/second.
   Operation starts immediately.
      Number of channels: 1
      index Type  Device    Channel InputType      Range       Name
      ----- ---- --------- ------- --------- ---------------- ----
      1     ai   cDAQ1Mod1 ai0     Diff      -10 to +10 Volts
```

Changing `NumberOfScans` changes the duration.

See Also

Properties

`ScansQueued`, `DurationInSeconds`

Methods

`daq.Session.startForeground`, `daq.Session.startBackground`, `daq.Session.queueOutputData`

Class

`daq.Session`

Purpose

Specify range of analog output hardware subsystem

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`OutputRange` is a two-element vector that specifies the range of voltages that can be output by the analog output (AO) subsystem. You should configure `OutputRange` so that the maximum dynamic range of your hardware is utilized. For many devices, the output range is expressed in terms of the gain and polarity.

AO subsystems have a finite number of `OutputRange` values that you can set. If an output range is specified but does not match a valid range, then the next highest supported range is automatically selected by the engine. If `OutputRange` exceeds the range of valid values, then an error is returned. Use the `daqhwinfo` function to return the output ranges supported by your board.

Because the engine can set the output range to a value that differs from the value you specify, you should return the actual output range for each channel using the `get` function or the device object display summary. Alternatively, you can use the `setverify` function, which sets the `OutputRange` value and then returns the actual value that is set.

Characteristics

Usage	AO, per channel
Access	Read/write
Data type	Two-element vector of doubles
Read-only when running	Yes

Values

The default value is determined by the hardware driver.

Examples

Create the analog output object `ao` for a National Instruments board and add two hardware channels to it.

```
ao = analogoutput('nidaq','Dev1');
addchannel(ao,0:1);
```

You can return the output ranges supported by the board with the `OutputRanges` field of the `daqhwinfo` function.

```
out = daqhwinfo(ao);
out.OutputRanges
ans =
    0.0000    10.0000
  -10.0000    10.0000
```

To configure both channels contained by `ao` to output signals between -10 volts and 10 volts:

```
ao.Channel.OutputRange = [-10 10];
```

Alternatively, you can use the `setverify` function to configure and return the `OutputRange` value.

```
ActualRange = setverify(ao.Channel,'OutputRange',[-10 10]);
```

See Also

Functions

`daqhwinfo`, `setverify`

Properties

`Units`, `UnitsRange`

Purpose

Indicate parent (device object) of channel or line

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

The parent of a channel (line) is defined as the device object that contains the channel (line).

You can create a copy of the device object containing a particular channel or line by returning the value of `Parent`. You can treat this copy like any other device object. For example, you can configure property values, add channels or lines to it, and so on.

Characteristics

Usage	AI, AO, per channel; DIO, per line
Access	Read-only
Data type	Device object
Read-only when running	N/A

Values

The value is defined when channels or lines are added to the device object.

Examples

Create the analog input object `ai` for a National Instruments board and add three hardware channels to it.

```
ai = analoginput('nidaq','Dev1');
chans = addchannel(ai,0:2);
```

To return the parent for channel 2:

```
parent = ai.Channel(2).Parent;
```

`parent` is an exact copy of `ai`.

```
isequal(ai,parent)
ans =
      1
```

Purpose

Specify port ID

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Hardware lines are often grouped together as a port. Digital I/O subsystems can consist of multiple ports and typically have eight lines per port. When adding hardware lines to a digital I/O object with `addline`, you can specify the port ID. The port ID is stored in the `Port` property. If the port ID is not specified, then the smallest port ID value is automatically used.

Characteristics

Usage	DIO, per line
Access	Read-only
Data type	Double
Read-only when running	N/A

Values

The port ID is defined when line are added to the digital I/O object with `addline`.

Examples

Create the digital I/O object `dio` and add two hardware channels to it.

```
dio = digitalio('nidaq','Dev1');
addline(dio,0:1,'In');
```

You can use `Port` property to return the port IDs associated with the lines contained by `dio`.

```
dio.Line.Port
ans =
    [0]
```

```
[0]
```

See Also

Functions

`addline`

Purpose

Specify resistance value

Description

Use this property to specify the resistance of the device.

You can specify any acceptable value in ohms. When you add an RTD Channel, the resistance is unknown and the `RO` property displays `Unknown`. You must change this value to set the resistance of this device.

See Also

Class

`daq.Session`

Properties

`RTDConfiguration`, `RTDType`

Range

Purpose

Specify channel measurement range

Description

When working with the session-based interface, use the `Range` to indicate the measurement range of a channel.

Values

Range is not applicable for counter channels. For analog channels, value is dependent on the measurement type. This property is read-only for all measurement types except `'Voltage'`. You can specify a range in volts for analog channels.

Purpose

Rate of operation in scans per second

Description

When working with the session-based interface, use the `Rate` property to set the number of scans per second.

Note Many hardware devices accept fractional rates.

Tip On most devices, the hardware limits the exact rates that you can set. When you set the rate, Data Acquisition Toolbox sets the rate to the next higher rate supported by the hardware. If the exact rate affects your analysis of the acquired data, obtain the actual rate after you set it, and then use that in your analysis.

Values

You can set the rate to any positive nonzero scalar value supported by the hardware in its current configuration.

Examples

Create a session object and add an analog input channel and change the session rate.

```
s = daq.createSession('ni');
s.addAnalogInputChannel('cDAQ1Mod1','ai1','Voltage');
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.
      Number of channels: 1
      index Type  Device    Channel InputType      Range       Name
      ----- ---- --------- ------- --------- --------------- ----
      1     ai   cDAQ1Mod1 ai1     Diff      -10 to +10 Volts
```

Properties, Methods, Events

See Also

Properties

DurationInSeconds, NumberOfScans, RateLimit

Class

daq.Session

Purpose

Limit of rate of operation based on hardware configuration

Description

In the session-based interface, the read-only `RateLimit` property displays the minimum and maximum rates that the session supports, based on the device configuration for the session.

Tip `RateLimit` changes dynamically as the session configuration changes.

See Also

Properties

`Rate`

Class

`daq.Session`

Purpose

Specify number of additional times queued data is output

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

To send data to an analog output subsystem, it must first be queued in the data acquisition engine with the `putdata` function. If you want to continuously output the same data, you can use multiple calls to `putdata`. However, because each `putdata` call consumes memory, a long output sequence can quickly bring your system to halt.

As an alternative to `putdata`, you can continuously output previously queued data using `RepeatOutput`. Because `RepeatOutput` requeues the data, additional memory resources are not consumed. While the data is being output, you cannot add additional data to the queue.

Characteristics

Usage	AO, common to all channels
Access	Read/write
Data type	Double
Read-only when running	Yes

Values

The default value is zero.

Examples

Create the analog output object `ao` for a sound card and add one channel to it.

```
ao = analogoutput('winsound');
chans = addchannel(ao,1);
```

To queue one second of data:

```
data = sin(linspace(0,10,8000))';
```

```
putdata(ao,data)
```

To continuously output `data` for 10 seconds:

```
set(ao,'RepeatOutput',9)
```

See Also

Functions

`putdata`

RTDConfiguration

Purpose

Specify wiring configuration of RTD device

Description

Use this property to specify the wiring configuration for measuring resistance.

When you create an RTD channel, the wiring configuration is unknown and the `RTDConfiguration` property displays `Unknown`. You must change this to one of the following valid configurations:

- `TwoWire`
- `ThreeWire`
- `FourWire`

See Also

Class

`daq.Session`

Properties

`R0`, `RTDType`

Purpose

Specify sensor sensitivity

Description

Use this property to specify the sensitivity of a standard RTD sensor in the session-based interface. A standard RTD sensor is defined as a 100–ohm platinum sensor.

When you create an RTD channel, the sensitivity is unknown and the `RTDType` property displays `Unknown`. You must change this to one of these valid values:

- `Pt3750`
- `Pt3851`
- `Pt3911`
- `Pt3916`
- `Pt3920`
- `Pt3928`

See Also

`daq.Session.addAnalogInputChannel`

Class

`daq.Session`

Properties

`RTDConfiguration`, `RTDType`

Running

Purpose

Indicate whether device object is running

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Along with the `Logging` or `Sending` property, `Running` reflects the state of an analog input or analog output object. `Running` can be `On` or `Off`.

`Running` is automatically set to `On` once the `start` function is issued. When `Running` is `On`, you can acquire data from an analog input device or send data to an analog output device after the trigger occurs. For digital I/O objects, `Running` is typically used to indicate if time-based events are being generated.

`Running` is automatically set to `Off` once the `stop` function is issued, the specified data is acquired or sent, or a run-time error occurs. When `Running` is `Off`, you cannot acquire or send data. However, you can acquire one sample with the `getsample` function, or send one sample with the `putsample` function.

Characteristics

Usage	AI, AO, DIO, common to all channels and lines
Access	Read-only
Data type	String
Read-only when running	N/A

Values

`{Off}`	The device object is not running.
`On`	The device object is running.

See Also

Functions

`getsample`, `putsample`, `start`

Properties

`Logging`, `Sending`

Purpose

Specify callback function to execute when run-time error occurs

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

A run-time error event is generated immediately after a run-time error occurs. This event executes the callback function specified for `RuntimeErrorFcn`. Additionally, a toolbox error message is automatically displayed to the MATLAB workspace. If an error occurs that is not explicitly handled by the toolbox, then the hardware-specific error message is displayed.

The default value for `RunTimeErrorFcn` is `daqcallback`, which displays the event type, the time the event occurred, and the device object name along with the error message.

Run-time error event information is stored in the `Type` and `Data` fields of the `EventLog` property. The `Type` field value is `Error`. The `Data` field values are given below.

Data Field Value	Description
`AbsTime`	The absolute time (as a `clock` vector) the event occurred.
`RelSample`	The acquired (AI) or output (AO) sample number when the event occurred.
`String`	The descriptive error message.

Run-time errors include hardware errors and timeouts. Run-time errors do not include configuration errors such as setting an invalid property value.

Characteristics

Usage	AI, AO, common to all channels
Access	Read/write
Data type	String
Read-only when running	No

Values

The default value is `daqcallback`.

See Also

Functions

`daqcallback`

Properties

`EventLog`, `Timeout`

Purpose

Specify per-channel rate at which analog data is converted to digital data, or vice versa

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`SampleRate` specifies the per-channel rate (in samples/second) that an analog input (AI) or analog output (AO) subsystem converts data. AI subsystems convert analog data to digital data, while AO subsystems convert digital data to analog data.

AI and AO subsystems have a finite (though often large) number of valid sampling rates. If you specify a sampling rate that does not match one of the valid values, the data acquisition engine automatically selects the nearest available sampling rate. In most data acquisition hardware, some valid sample rates can be non integers. See The Sampling Rate for more info about valid sample rates.

Because the engine can set the sampling rate to a value that differs from the value you specify, you should return the actual sampling rate using the `get` function or the device object display summary. Alternatively, you can use the `setverify` function, which sets the `SampleRate` value and then returns the actual value that is set. To find out the range of sampling rates supported by your board, use the `propinfo` function. Additionally, because the actual sampling rate depends on the number of channels contained by the device object and the `ChannelSkew` property value (AI only), `SampleRate` should be the last property you set before starting the device object.

Characteristics

Usage	AI, AO, common to all channels
Access	Read/write

Data type	Double
Read-only when running	Yes

Values

The default value is obtained from the hardware driver.

Examples

Create the analog input object `ai` for a sound card and add two channels to it.

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

You can find out the range of valid sampling rates with the `ConstraintValue` field of the `propinfo` function.

```
rates = propinfo(ai,'SampleRate');
rates.ConstraintValue
ans =
        8000        48000
```

To configure the per-channel sampling rate to 48 kHz:

```
set(ai,'SampleRate',48000)
```

Alternatively, you can use the `setverify` function to configure and return the `SampleRate` value.

```
ActualRate = setverify(ai,'SampleRate',48000);
```

See Also

Functions

`propinfo`, `setverify`

Properties

`ChannelSkew`

SamplesAcquired

Purpose

Indicate number of samples acquired per channel

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`SamplesAcquired` is continuously updated to reflect the current number of samples acquired by an analog input object. It is reset to zero after a `start` function is issued.

Use the `SamplesAvailable` property to find out how many samples are available to be extracted from the engine.

Characteristics

Usage	AI, common to all channels
Access	Read-only
Data type	Double
Read-only when running	N/A

Values

The value is continuously updated to reflect the current number of samples acquired. The default value is zero.

See Also

Functions

`start`

Properties

`SamplesAvailable`

Purpose

Specify callback function to execute when predefined number of samples is acquired for each channel group member

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

A samples acquired event is generated immediately after the number of samples specified by the `SamplesAcquiredFcnCount` property is acquired for each channel group member. This event executes the callback function specified for `SamplesAcquiredFcn`.

The samples acquired event is executed regardless of its waiting time in the queue.

Use `SamplesAcquiredFcn` to trigger an event each time a specified number of samples is acquired. To process samples at regular time intervals, use the `TimerFcn` property.

Samples acquired event information is not stored in the `EventLog` property. When the callback function is executed, the second argument is a structure containing two fields. The `Type` field value is set to the string `'SamplesAcquired'`, and the `Data` field values are given below.

Data Field Value	Description
`AbsTime`	The absolute time (as a `clock` vector) the event occurred.
`RelSample`	The acquired sample number when the event occurred.

Characteristics

Usage	AI, common to all channels
Access	Read/write

Data type	String
Read-only when running	No

Values

The default value is an empty string.

See Also

Properties

`EventLog`, `SamplesAcquiredFcnCount`, `TimerFcn`

Purpose

Specify number of samples to acquire for each channel group member before samples acquired event is generated

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

A samples acquired event is generated immediately after the number of samples specified by `SamplesAcquiredFcnCount` is acquired for each channel group member. This event executes the callback function specified by the `SamplesAcquiredFcn` property.

Characteristics

Usage	AI, common to all channels
Access	Read/write
Data type	Double
Read-only when running	Yes

Values

The default value is 1024.

See Also

Properties

`SamplesAcquiredFcn`

SamplesAvailable

Purpose

Indicate number of samples available per channel in engine

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

For analog input (AI) objects, `SamplesAvailable` indicates the number of samples that can be extracted from the engine for each channel group member with the `getdata` function. For analog output (AO) objects, `SamplesAvailable` indicates the number of samples that have been queued with the `putdata` function, and can be sent (output) to each channel group member.

After data has been extracted (AI) or output (AO), the `SamplesAvailable` value is reduced by the appropriate number of samples. For AI objects, `SamplesAvailable` is reset to zero after a `start` function is issued.

For AI objects, use the `SamplesAcquired` property to find out how many samples have been acquired since the `start` function was issued. For AO objects, use the `SamplesOutput` property to find out how many samples have been output since the `start` function was issued.

Characteristics

Usage	AI, AO, common to all channels
Access	Read-only
Data type	Double
Read-only when running	N/A

Values

The value is automatically updated based on the number of samples acquired (analog input) or sent (analog output). The default value is zero.

See Also

Functions

`start`

Properties

`SamplesAcquired`, `SamplesOutput`

SamplesOutput

Purpose

Indicate number of samples output per channel from engine

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

`SamplesOutput` is continuously updated to reflect the current number of samples output by an analog output object. It is reset to zero after the device objects stops and data has been queued with the `putdata` function.

Use the `SamplesAvailable` property to find out how many samples are available to be output from the engine.

Characteristics

Usage	AO, common to all channels
Access	Read-only
Data type	Double
Read-only when running	N/A

Values

The value is continuously updated to reflect the current number of samples output. The default value is zero.

See Also

Functions

`putdata`

Properties

`SamplesAvailable`

Purpose

Specify callback function to execute when predefined number of samples is output for each channel group member

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

A samples output event is generated immediately after the number of samples specified by the `SamplesOutputFcnCount` property is output for each channel group member. This event executes the callback function specified for `SamplesOutputFcn`.

Use `SamplesOutputFcn` to trigger an event each time a specified number of samples is output. To process samples at regular time intervals, use the `TimerFcn` property.

Samples output event information is not stored in the `EventLog` property. When the callback function is executed, the second argument is a structure containing two fields. The `Type` field value is set to the string `'SamplesOutput'`, and the event `Data` field values are given below.

Data Field Value	Description
`AbsTime`	The absolute time (as a `clock` vector) the event occurred.
`RelSample`	The output sample number when the event occurred.

Characteristics

Usage	AO, common to all channels
Access	Read/write

Data type	String
Read-only when running	No

Values

The default value is an empty string.

See Also

Properties

`EventLog`, `SamplesOutputFcnCount`

Purpose

Specify number of samples to output for each channel group member before samples output event is generated

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

A samples output event is generated immediately after the number of samples specified by `SamplesOutputFcnCount` is output for each channel group member. This event executes the callback function specified by the `SamplesOutputFcn` property.

Characteristics

Usage	AO, common to all channels
Access	Read/write
Data type	Double
Read-only when running	Yes

Values

The default value is 1024.

See Also

Properties

`SamplesOutputFcn`

SamplesPerTrigger

Purpose

Specify number of samples to acquire for each channel group member for each trigger that occurs

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`SamplesPerTrigger` specifies the number of samples to acquire for each analog input channel group member for each trigger that occurs. If `SamplesPerTrigger` is set to `Inf`, then the analog input object continually acquires data until a `stop` function is issued or an error occurs.

The default value of `SamplesPerTrigger` is calculated by the data acquisition engine such that one second of data is acquired. This calculation is based on the value of `SampleRate`.

Characteristics

Usage	AI, common to all channels
Access	Read/write
Data type	Double
Read-only when running	Yes

Values

The default value is set by the engine such that one second of data is acquired.

Examples

Create the analog input object `ai` for a sound card and add two channels to it.

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

By default, a one second acquisition in which 8000 samples are acquired for each channel is defined. To define a two second acquisition at the same sampling rate:

```
set(ai,'SamplesPerTrigger',16000)
```

See Also

Functions

`stop`

Properties

`SampleRate`

ScansAcquired

Purpose

Number of scans acquired during operation

Description

In the session-based interface, the `ScansAcquired` property displays the number of scans acquired after you start the operation using `daq.Session.startBackground`.

Values

The read-only value represents the number of scans acquired by the hardware. This value is reset each time you call `daq.Session.startBackground`.

See Also

Properties

`NumberOfScans`, `ScansOutputByHardware`

Methods

`daq.Session.startBackground`

Class

`daq.Session`

ScansOutputByHardware

Purpose

Indicate number of scans output by hardware

Description

In the session-based interface, the `ScansOutputByHardware` property displays the number of scans output by the hardware after you start the operation using `daq.Session.startBackground`.

Tip The value depends on information from the hardware.

Values

This read-only value is based on the output of the hardware configured for your session.

See Also

Properties

`ScansAcquired`, `ScansQueued`

Methods

`daq.Session.queueOutputData`, `daq.Session.startBackground`

Class

`daq.Session`

ScansQueued

Purpose

Indicate number of scans queued for output

Description

In the session-based interface, the `ScansQueued` property displays the number of scans queued for output `daq.Session.queueOutputData`. The `ScansQueued` property increases when you successfully call `daq.Session.queueOutputData`. The `ScansQueued` property decreases when the hardware reports that it has successfully output data.

Values

This read-only value is based on the number of scans queued.

See Also

Properties

`ScansOutputByHardware`

Methods

`daq.Session.queueOutputData`

Class

`daq.Session`

Purpose

Indicate whether data is being sent to hardware device

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Along with the `Running` property, `Sending` reflects the state of an analog output object. `Sending` can be `On` or `Off`.

`Sending` is automatically set to `On` when a trigger occurs. When `Sending` is `On`, queued data is being output to the analog output subsystem.

`Sending` is automatically set to `Off` when the queued data has been output, an error occurs, or a `stop` function is issued. When `Sending` is `Off`, data is not being output to the analog output subsystem although you can output a single sample with the `putsample` function.

Characteristics

Usage	AO, common to all channels
Access	Read-only
Data type	String
Read-only when running	N/A

Values

`{Off}`	Data is not being sent to the analog output hardware.
`On`	Data is being sent to the analog output hardware.

See Also

Functions

`putsample`

Properties

`Running`

Sensitivity

Purpose

Sensitivity of an analog channel

Description

When working with the session-based interface, the `Sensitivity` property to set the accelerometer or microphone sensor channel.

Sensitivity in an accelerometer channel is expressed as $\frac{v}{g}$, or volts per gravity.

Sensitivity in a microphone channel is expressed as $\frac{v}{pa}$, or volts per pascal.

Examples

Create a session object, add an analog input channel, with the `'accelerometer'` `MeasurementType`.

```
s = daq.createSession('ni');
s.addAnalogInputChannel('Dev4', 'ai0', 'accelerometer')

Data acquisition session using National Instruments hardware:
   Will run for 1 second (2000 scans) at 2000 scans/second.
   Number of channels: 1
      index Type Device Channel      MeasurementType             Range        Name
      ----- ---- ------ ------- ------------------------- ----------------- ----
      1     ai   Dev4   ai0     Accelerometer (PseudoDiff) -5.0 to +5.0 Volts
```

Change the `Sensitivity` to 10.2e-3 V/G:

```
s.Channels(1).Sensitivity = 10.2e-3

s =

Data acquisition session using National Instruments hardware:
   Will run for 1 second (2000 scans) at 2000 scans/second.
   Number of channels: 1
      index Type Device Channel      MeasurementType               Range          Name
      ----- ---- ------ ------- ------------------------- --------------------- ----
```

```
1      ai    Dev4    ai0     Accelerometer (PseudoDiff) -490 to +490 Gravities
```

See Also

Class

`daq.Session`, `daq.Session.addAnalogInputChannel`

SensorRange

Purpose

Specify range of data expected from sensor

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

You use `SensorRange` to scale your data to reflect the range you expect from your sensor. You can find the appropriate sensor range from your sensor's specification sheet.

The data is scaled while it is extracted from the engine with the `getdata` function according to the formula

$$\textit{scaled value} = \frac{(A\ /\ D\ \textit{value})(\textit{units range})}{(\textit{sensor range})}$$

The A/D value is constrained by the `InputRange` property, which reflects the gain and polarity of your hardware channels. The units range is given by the `UnitsRange` property.

Characteristics

Usage	AI, per channel
Access	Read/write
Data type	Two-element vector of doubles
Read-only when running	No

Values

The default value is determined by the default value of the `InputRange` property.

See Also

Functions

`getdata`

Properties

`InputRange`, `Units`, `UnitsRange`

ShuntLocation property

Purpose

Indicate location of channel's shunt resistor

Description

When working with the session-based interface, `ShuntLocation` on the analog input current channel indicates if the shunt resistor is located internally on the device or externally. Values are:

- `'Internal'`: when the shunt resistor is located internally.
- `'External'`: when the shunt resistor is located externally.

If your device supports an internal shunt resistor, this property is set to `Internal` by default. If the shunt location is external, you must specify the shunt resistance value.

See Also

`ShuntResistance` |

Purpose

Resistance value of channel's shunt resistor

Description

When working with the session-based interface, the analog input current channel's `ShuntResistance` property indicates resistance in ohms. This value is automatically set if the shunt resistor is located internally on the device and is read only.

Note Before starting an analog output channel with an external shunt resistor, specify the shunt resistance value.

See Also

`ShuntLocation` |

Source

Purpose

Indicates trigger source terminal

Description

When working with the session-based interface, the `Source` property indicates the device and terminal to which you added a trigger.

See Also

`Destinationdaq.Session.addTriggerConnection`

Purpose

Specify callback function to execute before device object runs

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

A start event is generated immediately after the `start` function is issued. This event executes the callback function specified for `StartFcn`. When the callback function has finished executing, `Running` is automatically set to `On` and the device object and hardware device begin executing. Note that the device object is not started if an error occurs while executing the callback function.

Start event information is stored in the `Type` and `Data` fields of the `EventLog` property. The `Type` field value is `Start`. The `Data` field values are given below.

Data Field Value	Description
`AbsTime`	The absolute time (as a `clock` vector) the event occurred.
`RelSample`	The acquired (AI) or output (AO) sample number when the event occurred.

Characteristics

Usage	AI, AO, common to all channels
Access	Read/write
Data type	String
Read-only when running	No

Values

The default value is an empty string.

See Also

Functions

`start`

Properties

`EventLog`, `Running`

Purpose

Specify callback function to execute after device object runs

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

A stop event is generated immediately after the device object and hardware device stop executing. This occurs when

- A `stop` function is issued.
- For analog input (AI) objects, the requested number of samples to acquire was reached or data was missed. For analog output (AO) objects, the requested number of samples to output was reached.
- A run-time error occurred.

A stop event executes the callback function specified for `StopFcn`. Under most circumstances, the callback function is not guaranteed to complete execution until sometime after the device object and hardware device stop, and the `Running` property is set to `Off`.

Stop event information is stored in the `Type` and `Data` fields of the `EventLog` property. The `Type` field value is `Stop`. The `Data` field values are given below.

Data Field Value	Description
`AbsTime`	The absolute time (as a `clock` vector) the event occurred.
`RelSample`	The acquired (AI) or output (AO) sample number when the event occurred.

Characteristics

Usage	AI, AO, common to all channels
Access	Read/write
Data type	String
Read-only when running	No

Values

The default value is an empty string.

See Also

Functions

`stop`

Properties

`EventLog`, `Running`

Purpose

Specify device object label

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`Tag` provides a means to identify device objects with a label. Using the `daqfind` function and the `Tag` value, you can identify and retrieve a device object that was cleared from the MATLAB workspace.

Characteristics

Usage	AI, AO, DIO, common to all channels and lines
Access	Read/write
Data type	String
Read-only when running	No

Values

The default value is an empty string.

Examples

Create the analog input object `ai` for a sound card and add two channels to it.

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

Assign `ai` a label using `Tag`.

```
set(ai,'Tag','Sound')
```

If `ai` is cleared from the workspace, you can use `daqfind` and the `Tag` value to identify and retrieve the device object.

```
clear ai
aicell = daqfind('Tag','Sound');
ai = aicell{1};
```

See Also

Functions

`daqfind`

Purpose

PFI terminal of counter subsystem

Description

When working with the session-based interface, the `Terminal` property indicates the counter subsystem's corresponding PFI terminal.

See Also

Class

`daq.Session`, `daq.Session.AddCounterInputChannel`, `daq.Session.addCounterOutputChannel`,

TerminalConfig

Purpose

Specify terminal configuration

Description

Use the `TerminalConfig` to change the configuration of your analog channel. The property displays the hardware default configuration. You can change this to

- `SingleEnded`
- `NonReferencedSingleEnded`
- `Differential`
- `PseudoDifferential`

See Also

`TerminalConfigsAvailable` | `daq.Session.addAnalogInputChannel` | `daq_ref.Session.addAnalogOutputChannel`

Purpose

Terminals available on device or CompactDAQ chassis

Description

When working with the session-based interface, the `Terminals` on the device or the CompactDAQ chassis lists all available terminals. The list includes terminals available for trigger and clock connections. When you access the `Terminals` property on modules on a CompactDAQ chassis, the terminals are on the chassis, not on the module.

Examples

Discover available devices:

```
d=daq.getDevices
```

```
d =

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
```

Access the `Terminals` property of NI 9205 with index `1`:

```
 d(1).Terminals
```

```
ans =

    'cDAQ1/PFI0'
    'cDAQ1/PFI1'
    'cDAQ1/20MHzTimebase'
    'cDAQ1/80MHzTimebase'
```

```
'cDAQ1/ChangeDetectionEvent'
'cDAQ1/AnalogComparisonEvent'
'cDAQ1/100kHzTimebase'
'cDAQ1/SyncPulse0'
'cDAQ1/SyncPulse1'
.
.
.
```

See Also

Class

`daq.Session`

Functions

`daq.getDevices`, `daq.Session.addTriggerConnection`, `daq.Session.addClockConnection`

ThermocoupleType

Purpose

Select thermocouple type

Description

When working with the session-based interface, use the `ThermocoupleType` property to select the type of thermocouple you will use to make your measurements. Select the type based on the temperature range and sensitivity you need.

Values

You can set the `ThermocoupleType` to:

- `'J'`
- `'K'`
- `'N'`
- `'R'`
- `'S'`
- `'T'`
- `'B'`
- `'E'`

By default the thermocouple type is `'Unknown'`.

Examples

Create a session and add an analog input channel with `'Thermocouple'` measurement type:

```
s = daq.createSession('ni');
s.addAnalogInputChannel('cDAQ1Mod6','ai1','Thermocouple');
ans =

Data acquisition analog input voltage channel 'ai0' on device 'cDAQ1Mo

           Units: Celsius
ThermocoupleType: Unknown
        Coupling: DC
  TerminalConfig: Differential
```

```
           Range: 0 to +750 Celsius
            Name: ''
              ID: 'ai0'
          Device: [1x1 daq.ni.CompactDAQModule]
 MeasurementType: 'Voltage'
```

Set the `ThermocoupleType` property to `'J'`:

```
s.Channels.Thermocoupletype = 'J';
```

See Also

Methods

`daq.Session.addAnalogInputChannel`

Class

`daq.Session`

Purpose

Specify additional waiting time to extract or queue data

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

The `Timeout` value (in seconds) is added to the time required to extract data from the engine or queue data to the engine. Because data is extracted with the `getdata` function, and queued with the `putdata` function, `Timeout` is associated only with these two "blocking" functions.

If the requested data is not extracted or queued after waiting the required time, then a time-out condition occurs and control is immediately returned to the MATLAB workspace. A time-out is one of the conditions for stopping an acquisition. When a time-out occurs, the callback function specified by `RuntimeErrorFcn` is called.

`Timeout` is not associated with hardware time-out conditions. Possible hardware time-out conditions include

- Triggering on a voltage level and that level never occurs
- Externally clocking an acquisition and the external clock signal never occurs
- Losing the hardware connection

To check for hardware timeouts, you might need to poll the appropriate property value.

Characteristics

Usage	AI, AO, common to all channels
Access	Read/write
Data type	Double
Read-only when running	Yes

Values

The default value is one second.

See Also

Functions

getdata, putdata

Properties

RuntimeErrorFcn

Purpose

Specify callback function to execute when predefined time period passes

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

A timer event is generated whenever the time specified by the `TimerPeriod` property passes. This event executes the callback function specified for `TimerFcn`. Time is measured relative to when the device object starts running.

Some timer events might not be processed if your system is significantly slow or if the `TimerPeriod` value is too small. The time taken to process an event depends on the sample rate, the performance of your system, and the data itself.

There can only be one timer event waiting in the queue at a given time. The callback function must process all available data to ensure that it keeps up with the inflow of data. Alternatively, you can use the `SamplesAcquiredFcn` (analog input) or `SamplesOutputFcn` (analog output) property to process the data when a specified number of samples is acquired or output.

Note For analog input objects, use the `SamplesAvailable` property inside a callback function to determine the number of samples available in the queue.

For digital I/O objects, timer events are typically used to update and display the state of the device object.

Timer event information is not stored in the `EventLog` property. When the callback function is executed, the second argument is a structure containing two fields. The `Type` field value is set to the string `'Timer'`, and the event `Data` field value is given below.

Data Field Value	Description
`AbsTime`	The absolute time (as a `clock` vector) the event occurred.

Characteristics

Usage	AI, AO, DIO, common to all channels and lines
Access	Read/write
Data type	String
Read-only when running	No

Values

The default value is an empty string.

See Also

Properties

`EventLog`, `SamplesAcquiredFcn`, `SamplesOutputFcn`, `TimerPeriod`

Purpose

Specify time period between timer events

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`TimerPeriod` specifies the time, in seconds, that must pass before the callback function specified for `TimerFcn` is called. Time is measured relative to when the hardware device starts running.

Some timer events might not be processed if your system is significantly slowed or if the `TimerPeriod` value is too small. For example, a common application for timer events is to display data. However, because displaying data is a CPU-intensive task, some of these events might be dropped.

Characteristics

Usage	AI, AO, DIO, common to all channels and lines
Access	Read/write
Data type	Double
Read-only when running	No

Values

The default value is 0.1 second.

See Also

Properties

`TimerFcn`

TriggerChannel

Purpose

Specify channel serving as trigger source

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

`TriggerChannel` specifies the channel serving as the trigger source. The trigger channel must be specified before the trigger type. You might need to configure the `TriggerCondition` and `TriggerConditionValue` properties in conjunction with `TriggerChannel`.

For all supported vendors, if `TriggerType` is `Software`, then you must acquire data from the channel being used for the trigger source. For National Instruments hardware, if `TriggerType` is `HwAnalogChannel`, then `TriggerChannel` must be the first element of the channel group. The exception is if you are using simultaneous acquisition devices such as the S-series boards, with which you can specify any channel for the `TriggerChannel` value.

Characteristics

Usage	AI, common to all channels
Access	Read/write
Data type	Vector or scalar
Read-only when running	Yes

Values

The data type can be either vector or scalar, representing one channel. The default value is an empty vector.

Examples

Create the analog input object `ai`, add two channels, and define the trigger source as channel 2.

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

```
set(ai,'TriggerChannel',ch(2))
set(ai,'TriggerType','Software')
```

See Also

Properties

`TriggerCondition`, `TriggerConditionValue`, `TriggerType`

TriggerCondition

Purpose

Specify condition that must be satisfied before trigger executes

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

When working with the session-based interface, use the `TriggerCondition` property to specify the signal condition that executes the trigger, which synchronizes operations on devices in a session. For more information, see .

The available trigger conditions depend on the value of `TriggerType`. If `TriggerType` is `Immediate` or `Manual`, the only available `TriggerCondition` is `None`. If `TriggerType` is `Software`, then `TriggerCondition` can be `Rising`, `Falling`, `Leaving`, or `Entering`. These trigger conditions require one or more voltage values to be specified for the `TriggerConditionValue` property.

Based on the hardware you are using, additional trigger conditions might be available. Refer to the values listed below.

Values

All Supported Hardware

The following trigger condition is used when `TriggerType` is `Immediate` or `Manual`.

`{None}`	No trigger condition is required.

The following trigger conditions are available when `TriggerType` is `Software`.

`{Rising}`	The trigger occurs when the signal has a positive slope when passing through the specified value.
`Falling`	The trigger occurs when the signal has a negative slope when passing through the specified value.
`Leaving`	The trigger occurs when the signal leaves the specified range of values.
`Entering`	The trigger occurs when the signal enters the specified range of values.

Measurement Computing

The following trigger conditions are available when `TriggerType` is `HwDigital`.

`GateHigh`	The trigger occurs as long as the digital signal is high.
`GateLow`	The trigger occurs as long as the digital signal is low.
`TrigHigh`	The trigger occurs when the digital signal is high.
`TrigLow`	The trigger occurs when the digital signal is low.
`TrigPosEdge`	The trigger occurs when the positive (rising) edge of the digital signal is detected.
`{TrigNegEdge}`	The trigger occurs when the negative (falling) edge of the digital signal is detected.

The following trigger conditions are available when `TriggerType` is `HwAnalog`.

`{TrigAbove}`	The trigger occurs when the analog signal makes a transition from below the specified value to above.
`TrigBelow`	The trigger occurs when the analog signal makes a transition from above the specified value to below.
`GateNegHys`	The trigger occurs when the analog signal is more than the specified high value. The acquisition stops if the analog signal is less than the specified low value.
`GatePosHys`	The trigger occurs when the analog signal is less than the specified low value. The acquisition stops if the analog signal is more than the specified high value.
`GateAbove`	The trigger occurs as long as the analog signal is more than the specified value.
`GateBelow`	The trigger occurs as long as the analog signal is less than the specified value.
`GateInWindow`	The trigger occurs as long as the analog signal is within the specified range of values.
`GateOutWindow`	The trigger occurs as long as the analog signal is outside the specified range of values.

National Instruments

The following trigger conditions are available for AI objects when `TriggerType` is `HwDigital`.

`PositiveEdge`	The trigger occurs when the positive (rising) edge of a digital signal is detected.
`{NegativeEdge}`	The trigger occurs when the negative (falling) edge of a digital signal is detected.

The following trigger conditions are available for AO objects on NI-DAQmx devices when `TriggerType` is `HwDigital`.

`PositiveEdge`	The trigger occurs when the positive (rising) edge of a digital signal is detected.
`{NegativeEdge}`	The trigger occurs when the negative (falling) edge of a digital signal is detected.

The following trigger conditions are available when `TriggerType` is `HwAnalogChannel` or `HwAnalogPin`.

`{AboveHighLevel}`	The trigger occurs when the analog signal is above the specified value.
`BelowLowLevel`	The trigger occurs when the analog signal is below the specified value.
`InsideRegion`	The trigger occurs when the analog signal is inside the specified region.
`LowHysteresis`	The trigger occurs when the analog signal is less than the specified low value with hysteresis given by the specified high value.
`HighHysteresis`	The trigger occurs when the analog signal is greater than the specified high value with hysteresis given by the specified low value.

See Also

Properties

`TriggerChannel`, `TriggerConditionValue`, `TriggerType`

TriggerCondition

Purpose

Specify condition that must be satisfied before trigger executes

Description

When working with the session-based interface, use the `TriggerCondition` property to specify the signal condition that executes the trigger, which synchronizes operations on devices in a session. For more information, see .

Values

Set the trigger condition to `RisingEdge` or `FallingEdge`.

Examples

Create a session and add channels and trigger to the session.

```
s = daq.createSession('ni');
s.addAnalogInputChannel('Dev1', 0, 'voltage');
s.addAnalogInputChannel('Dev2', 0, 'voltage');
s.addTriggerConnection('Dev1/PFI4','Dev2/PFI0','StartTrigger');
```

Change the trigger condition to `FallingEdge`.

```
 s.Connections(1).TriggerCondition='FallingEdge'


s =


Data acquisition session using National Instruments hardware:
   Will run for 1 second (1000 scans) at 1000 scans/second.


   Trigger Connection added. (Details)


   Number of channels: 2
      index Type Device Channel MeasurementType      Range       Name
      ----- ---- ------ ------- --------------- ---------------- ----
      1     ai   Dev1   ai0     Voltage (Diff)  -10 to +10 Volts
      2     ai   Dev2   ai0     Voltage (Diff)  -10 to +10 Volts
```

See Also

`daq.Session.addTiggerConnection`

Properties

`TriggerType`

Purpose

Specify voltage value(s) that must be satisfied before trigger executes

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

For all hardware `TriggerConditionValue` is used when `TriggerType` is `Software`, and is ignored when `TriggerCondition` is `None`. For vendor specific triggers, refer to the `TriggerCondition` and the `TriggerType` properties.

To execute a software trigger, the values specified for `TriggerCondition` and `TriggerConditionValue` must be satisfied. When `TriggerCondition` is `Rising` or `Falling`, `TriggerConditionValue` accepts a single value. When `TriggerCondition` is `Entering` or `Leaving`, `TriggerConditionValue` accepts a two-element vector of values. For vendor specific values, refer to the `TriggerCondition` property.

Characteristics

Usage	AI, common to all channels
Access	Read/write
Data type	Double (or a two-element vector of doubles)
Read-only when running	Yes

Values

The default value is zero.

Examples

Create the analog input object `ai` and add one channel to it.

```
ai = analoginput('winsound');
ch = addchannel(ai,1);
```

The trigger executes when a signal with a negative slope passing through 0.2 volts is detected on channel 1.

```
set(ai,'TriggerChannel',ch)
set(ai,'TriggerType','Software')
set(ai,'TriggerCondition','Falling')
set(ai,'TriggerConditionValue',0.2)
```

Create the analog input object ai for a National Instruments device and add four channels to it.

```
ai = analoginput('nidaq', 'Dev1');
ch = addchannel(ai,0:3);
```

The trigger executes when a signal with a positive slope passing through 4.5 volts is detected on PFI2.

```
set(ai,'TriggerType','HwDigital')
set(ai,'HwDigitalTriggerSource', 'PFI2')
set(ai,'TriggerCondition','PositiveEdge')
set(ai,'TriggerConditionValue',4.5)
```

See Also

Properties

`TriggerCondition`, `TriggerType`

Purpose

Specify delay value for data logging

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

You can define both pretriggers and postriggers. Pretriggers are specified with a negative `TriggerDelay` value while postriggers are specified with a positive `TriggerDelay` value. You can delay a trigger in units of time or samples with the `TriggerDelayUnits` property. Pretriggers are not defined for hardware triggers or when `TriggerType` is `Immediate`.

Pretrigger samples are included as part of the total samples acquired per trigger as specified by the `SamplesPerTrigger` property. If sample-time pairs are returned to the workspace with the `getdata` function, then the pretrigger samples are identified with negative time values.

Characteristics

Usage	AI, common to all channels
Access	Read/write
Data type	Double
Read-only when running	Yes

Values

The default value is zero.

Examples

Create the analog input object `ai` and add one channel to it.

```
ai = analoginput('winsound');
ch = addchannel(ai,1);
```

Configure ai to acquire 44,100 samples per trigger with 11,025 samples (0.25 seconds) acquired as pretrigger data.

```
set(ai,'SampleRate',44100)
set(ai,'TriggerType','Manual')
set(ai,'SamplesPerTrigger',44100)
set(ai,'TriggerDelay',-0.25)
```

See Also

Properties

SamplesPerTrigger, TriggerDelayUnits

Purpose

Specify units in which trigger delay data is measured

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`TriggerDelayUnits` can be `Seconds` or `Samples`. If `TriggerDelayUnits` is `Seconds`, then data logging is delayed by the specified time for each channel group member. If `TriggerDelayUnits` is `Samples`, then data logging is delayed by the specified number of samples for each channel group member.

The trigger delay value is given by the `TriggerDelay` property.

Characteristics

Usage	AI, common to all channels
Access	Read/write
Data type	String
Read-only when running	Yes

Values

`{Seconds}`	The trigger is delayed by the specified number of seconds.
`Samples`	The trigger is delayed by the specified number of samples.

See Also

Properties

`TriggerDelay`

TriggerFcn

Purpose

Specify callback function to execute when trigger occurs

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

A trigger event is generated immediately after a trigger occurs. This event executes the callback function specified for `TriggerFcn`. Under most circumstances, the callback function is not guaranteed to complete execution until sometime after `Logging` is set to `On` for analog input (AI) objects, or `Sending` is set to `On` for analog output (AO) objects.

Trigger event information is stored in the `Type` and `Data` fields of the `EventLog` property. The `Type` field value is `Trigger`. The `Data` field values are given below.

Data Field Value	Description
`AbsTime`	The absolute time (as a `clock` vector) the event occurred.
`RelSample`	The acquired (AI) or output (AO) sample number when the event occurred.
`Channel`	The index number for each input channel serving as a trigger source (AI only).
`Trigger`	The trigger number.

Characteristics

Usage	AI, AO, common to all channels
Access	Read/write
Data type	String
Read-only when running	No

Values

The default value is an empty string.

See Also

Functions

`trigger`

Properties

`EventLog`, `Logging`

Purpose

Specify number of additional times trigger executes

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

You can configure a trigger to occur once (one-shot acquisition) or multiple times. If `TriggerRepeat` is set to its default value of zero, then the trigger executes once. If `TriggerRepeat` is set to a positive integer value, then the trigger executes once, and is repeated the specified number of times. For example, if the value is set to 2, you will get a total of 3 triggers. If `TriggerRepeat` is set to `inf` then the trigger executes continuously until a `stop` function is issued or an error occurs.

You can quickly evaluate how many triggers have executed by examining the `TriggersExecuted` property or by invoking the display summary for the device object. The display summary is invoked by typing the device object name at the MATLAB Command Window.

Note We have observed that National Instruments USB devices have a significant cycle time for the communications required to trigger the device. If you are using an NI USB device, we recommend that you set up longer acquisitions that use fewer triggers. That is, increase `SamplesPerTrigger` and decrease `TriggerRepeat`.

Characteristics

Usage	AI, common to all channels
Access	Read/write
Data type	Double
Read-only when running	Yes

Values

The default value is zero.

See Also

Functions

`disp`, `stop`

Properties

`SamplesPerTrigger`, `TriggersExecuted`, `TriggerType`

Purpose

Indicate number of triggers that execute

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

You can find out how many triggers executed by returning the value of `TriggersExecuted`. The trigger number for each trigger executed is also recorded by the `Data.Trigger` field of the `EventLog` property.

Characteristics

Usage	AI, AO, common to all channels
Access	Read-only
Data type	Double
Read-only when running	N/A

Values

The default value is zero.

Examples

Create the analog input object `ai` and add one channel to it.

```
ai = analoginput('winsound');
ch = addchannel(ai,1);
```

Configure `ai` to acquire 40,000 samples with five triggers using the default sampling rate of 8000 Hz.

```
set(ai,'TriggerRepeat',4)
start(ai)
```

`TriggersExecuted` returns the number of triggers executed.

```
ai.TriggersExecuted
ans =
      5
```

See Also

Properties

`EventLog`

TriggersPerRun

Purpose

Indicate the number of times the trigger executes in an operation

Description

When working with the session-based interface, the `TriggersPerRun` property indicates the number of times the specified trigger executes for one acquisition or generation session.

See Also

`daq.Session.addTriggerConnection`

Purpose Indicates the number of trigger to execute in an operation

Description When working with the session-based interface, the `TriggersRemaining` property indicates the number of trigger remaining for this acquisition or generation session. This value depends on the number of triggers set using `TriggersPerRun`.

See Also `daq.Session.addTriggerConnection`

TriggerType

Purpose

Specify type of trigger to execute

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`TriggerType` can be `Immediate`, `Manual`, or `Software`. If `TriggerType` is `Immediate`, the trigger occurs immediately after the `start` function is issued. If `TriggerType` is `Manual`, the trigger occurs immediately after the `trigger` function is issued. If `TriggerType` is `Software`, the trigger occurs when the associated trigger condition is satisfied (AI only).

For a given hardware device, additional trigger types might be available. Some trigger types require trigger conditions and trigger condition values. Trigger conditions are specified with the `TriggerCondition` property, while trigger condition values are specified with the `TriggerConditionValue` property.

When a trigger occurs for an analog input object, data logging is initiated and the `Logging` property is automatically set to `On`. When a trigger occurs for an analog output object, data sending is initiated and the `Sending` property is automatically set to `On`.

Characteristics

Usage	AI, AO, common to all channels
Access	Read/write
Data type	String
Read-only when running	Yes

Values

All Supported Hardware

`{Immediate}`	The trigger executes immediately after `start` is issued. Pretrigger data cannot be captured.
`Manual`	The trigger executes immediately after the `trigger` function is issued.
`Software`	The trigger executes when the associated trigger condition is satisfied. Trigger conditions are given by the `TriggerCondition` property. (AI only).

Measurement Computing

`HwDigital`	The trigger source is an external digital signal (AI only). Pretrigger data cannot be captured.
`HwAnalog`	The trigger source is an external analog signal (AI only).

National Instruments

`HwDigital`	The trigger source is an external digital signal. Pretrigger data cannot be captured. Control the trigger source with `HwDigitalTriggerSource` property. Specify the external digital signal with the `TriggerCondition` and `TriggerConditionValue` properties.
`HwAnalogChannel`	The trigger source is an external analog signal (AI only). To set the trigger source, see `TriggerChannel` property.
`HwAnalogPin`	The trigger source is a low-range external analog signal (AI only). Note that `HwAnalogPin` is supported only for Traditional NIDAQ devices. It is not supported for NIDAQmx devices.

For 1200 Series hardware, `HwDigital` is the only device-specific `TriggerType` value for analog input subsystems. Analog output subsystems do not support any device-specific `TriggerType` values.

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. See the supported hardware page at `www.mathworks.com/products/daq/supportedio.html` for more information.

See Also

Functions

`start`, `trigger`

Properties

`Logging`, `Sending`, `TriggerChannel`, `TriggerCondition`, `TriggerConditionValue`

Purpose

Type of trigger executed

Description

When working with the session-based interface, use this read only property displays the type of trigger that the source device executes to synchronize operations in the session. Currently all trigger types re `digital`.

See Also

Functions

`daq.Session.addTriggerConnection`

Properties

`TriggerCondition`

Purpose

Indicate device object type, channel, or line

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Type is associated with device objects, channels, and lines. For device objects, `Type` can be `Analog Input`, `Analog Output`, or `Digital I/O`. Once a device object is created, the value of `Type` is automatically defined.

For channels, the only value of `Type` is `Channel`. For lines, the only value of `Type` is `Line`. The value is automatically defined when channels or lines are added to the device object.

Characteristics

Usage	AI, AO, common to all channels and per channel; DIO, common to all lines and per line
Access	Read-only
Data type	String
Read-only when running	N/A

Values

Device Objects

For device objects, `Type` has these possible values:

`Analog Input`	The device object type is analog input.
`Analog Output`	The device object type is analog output.
`Digital IO`	The device object type is digital I/O.

The value is automatically defined after the device object is created.

Channels and Lines

For channels, the only value of `Type` is `Channel`. For lines, the only value of `Type` is `Line`. The value is automatically defined when channels or lines are added to the device object.

Type

Purpose

Display synchronization trigger type

Description

When working with the session-based interface, this property displays the trigger type

Characteristics

Usage	AI, AO, common to all channels and per channel; DIO, common to all lines and per line
Access	Read-only
Data type	String
Read-only when running	N/A

Values

Device Objects

For device objects, `Type` has these possible values:

`Analog Input`	The device object type is analog input.
`Analog Output`	The device object type is analog output.
`Digital IO`	The device object type is digital I/O.

The value is automatically defined after the device object is created.

Channels and Lines

For channels, the only value of `Type` is `Channel`. For lines, the only value of `Type` is `Line`. The value is automatically defined when channels or lines are added to the device object.

Purpose

Specify unit of RTD measurement

Description

Use this property to specify the temperature unit of the analog input channel with RTD measurement type in the session-based interface.

You can specify temperature values as:

- `Celsius` (Default)
- `Fahrenheit`
- `Kelvin`
- `Rankine`

See Also

Class

`daq.Session`

Units

Purpose

Specify engineering units label

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`Units` is a string that specifies the engineering units label to associate with your data. You should use `Units` in conjunction with the `UnitsRange` property.

Characteristics

Usage	AI, AO, per channel
Access	Read/write
Data type	String
Read-only when running	No

Values

The default value is `Volts`.

See Also

Properties

`UnitsRange`

Purpose

Specify range of data as engineering units

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

You use `UnitsRange` to scale your data to reflect particular engineering units.

For analog input objects, the data is scaled while it is extracted from the engine with the `getdata` function according to the formula

scaled value = (A/D value)(units range)/(sensor range)

The A/D value is constrained by the `InputRange` property, which reflects the gain and polarity of your analog input channels. The sensor range is given by the `SensorRange` property, which reflects the range of data you expect from your sensor.

For analog output objects, the data is scaled when it is queued in the engine with the `putdata` function according to the formula

scaled value = (original value)(output range)/(units range)

The output range is constrained by the `OutputRange` property, which specifies the gain and polarity of your analog output channels.

For both objects, you can also use the `Units` property to associate a meaningful label with your data.

Characteristics

Usage	AI, AO, per channel
Access	Read/write
Data type	Two-element vector of doubles
Read-only when running	No

Values

The default value is determined by the default value of the `InputRange` or the `OutputRange` property.

See Also

Functions

`getdata`, `putdata`

Properties

`InputRange`, `OutputRange`, `SensorRange`, `Units`

Purpose

Store data to associate with device object

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`UserData` stores data that you want to associate with the device object.

Note that if you return analog input object information to the MATLAB workspace using the `daqread` function, the `UserData` value is not restored.

Characteristics

Usage	AI, AO, DIO, common to all channels and lines
Access	Read/write
Data type	Any type
Read-only when running	No

Values

The default value is an empty vector.

Examples

Create the analog input object `ai` and add two channels to it.

```
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:1);
```

Suppose you want to access filter coefficients during the acquisition. You can create a structure to store these coefficients, which can then be stored in `UserData`.

```
coeff.a = 1.0;
coeff.b = -1.25;
set(ai,'UserData',coeff)
```

Vendor

Purpose

Vendor information associated with session object

Description

In the session-based interface, the `Vendor` property displays information about the vendor.

Examples

Use the `daq.getVendors` to get information about vendors.

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

v =

Data acquisition vendor 'National Instruments':

            ID: 'ni'
      FullName: 'National Instruments'
AdaptorVersion: '3.3 (R2013a)'
 DriverVersion: '9.2.3 NI-DAQmx'
 IsOperational: true


Properties, Methods, Events

Additional data acquisition vendors may be available as downloadable supp
Open the Support Package Installer to install additional vendors.
```

Values

a `daq.Vendor` object that represents the vendor associated with the session.

See Also

Properties

`ScansQueued`

Methods

`daq.createSession`

Class

`daq.Session`

ZResetCondition

Purpose

Reset condition for Z-indexing

Description

When working with the session-based interface, use the `ZResetCondition` property to specify reset conditions for Z-indexing of counter Input `'Position'` channels. Accepted values are:

- `'BothHigh'`
- `'BothLow'`
- `'AHigh'`
- `'BHigh'`

See Also

Class

`daq.Session`, `daq.Session.addCounterInputChannel`

ZResetEnable

Purpose

Enable reset for Z-indexing

Description

When working with the session-based interface, use the `ZResetEnable` property to specify if you will allow the Z-indexing to be reset on a counter input `'Position'` channel.

See Also

Class

`daq.Session`, `daq.Session.addCounterInputChannel`

ZResetValue

Purpose

Reset value for Z-indexing

Description

When working with the session-based interface, use the `ZResetValue` property to specify the reset value for Z-indexing on a counter input `'Position'` channel.

See Also

Class

`daq.Session`, `daq.Session.addCounterInputChannel`

2

Device-Specific Properties — Alphabetical List

BiDirectionalBit property

Purpose

Specify BIOS control register bit that determines bidirectional operation

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`BiDirectionalBit` can be `5`, `6`, or `7`. The default value is `5` because most parallel port hardware uses bit 5 of the BIOS control register to determine the direction (input or output) of port 0.

If port 0 is unable to input data, you need to configure the `BiDirectionalBit` value to `6` or `7`. Typically, you will not know the bit value required by your port, and some experimentation is required.

Note The Parallel Port adaptor will be deprecated in a future version of the toolbox. If you create a Data Acquisition Toolbox™ object for `'parallel'` beginning in R2008b, you will receive a warning stating that this adaptor will be removed in a future release. See the supported hardware page at `www.mathworks.com/products/daq/supportedio.html` for more information.

Characteristics

Vendor	Parallel port
Usage	DIO, common to all lines
Access	Read/write
Data type	Double
Read-only when running	Yes

Values

{5}, 6, or 7	The BIOS control register bit that determines bidirectional operation.

BitsPerSample property

Purpose

Specify number of bits sound card uses to represent samples

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

`BitsPerSample` can be `8`, `16`, or any value between `17` and `32`. The specified number of bits determines the number of unique values a sample can take on. For example, if `BitsPerSample` is `8`, the sound card represents each sample with 8 bits. This means that each sample is represented by a number from 0 through 255. If `BitsPerSample` is `16`, the sound card represents each sample with 16 bits. This means that each sample is represented by a number from 0 through 65,535.

For older Sound Blaster cards configured for full duplex operation, you might not be able to set `BitsPerSample` to 16 bits for both the analog input and analog output subsystems. Instead, you need to set one subsystem for 8 bits, and the other subsystem for 16 bits.

Note To use the high-resolution (greater than 16 bit) capabilities for some sound cards, you might need to configure `BitsPerSample` to either `24` or `32` even if your device does not use that number of bits.

Characteristics

Vendor	Sound cards
Usage	AI, AO, common to all channels
Access	Read/write
Data type	Double
Read-only when running	Yes

Values	8, {16}, or 17-32	Represent data with the specified number of bits.

Coupling property

Purpose

Specify input coupling mode

Description

The `Coupling` property is visible only if the device you are using supports coupling and the value can be changed. `Coupling` can be `DC` or `AC`. If `Coupling` is `DC`, the input is connected directly to the amplifier. If `Coupling` is `AC`, a series capacitor is inserted between the input connector and the amplifier.

When AC coupling is selected, the DC bias component of the measured signal is filtered out of the waveform by the hardware. This is typically used with dynamic signals such as audio. When DC coupling is selected, the complete signal including the DC bias component is measured. This is typically used with slowly changing signals such as temperature or voltage readings.

Values

`{AC}`	A series capacitor is inserted between the input connector and the amplifier.
`DC`	The input is connected directly to the amplifier.

The default is set to `AC` for

- National Instruments devices that use the NI-DAQmx interface and support AC coupling
- National Instruments DSA cards using the Traditional NI-DAQ interface

---

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. See the supported hardware page at `www.mathworks.com/products/daq/supportedio.html` for more information.

In all other cards, the default is set to DC.

Examples

In the session-based interface, create a session and add an analog input channel.

```
s = daq.createSession('ni');
ch = s.addAnalogInputChannel('Dev4', 'ai1', 'Voltage')
```

Change the coupling type to DC:

```
ch.Coupling = 'DC';
```

In the legacy interface, create the analog input object ai for a National Instruments board, and add a hardware channel to it.

```
ai = analoginput('nidaq','Dev1');
addchannel(ai,0);
```

You can return the coupling modes supported by the board with the Coupling field of the daqhwinfo function.

```
out = daqhwinfo(ai);
out.Coupling
ans =
   'AC,DC'
```

Configure the channel contained by ai to use dc-coupling:

```
ai.Channel.Coupling = 'DC';
ai.Channel.Coupling
ans=
DC
```

ExternalClockDriveLine property

Purpose

Specify which signal is driven by the clock indicating that an analog output update has occurred

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`ExternalClockDriveLine` defines which pin is pulsed when analog output channels are updated. You can use this property to synchronize the operations of multiple cards over the RTSI bus or via external PFI pins.

Note The National Instruments term for this clock is AO Sample Clock.

Characteristics

Vendor	National Instruments
Usage	AO
Access	Read/write
Data type	String
Read-only when running	Yes

Values

`PFI0` to `PFI15`	Use specified pin from `PFI0` through `PFI15`.
`RTSI0` to `RTSI6`	Use specified pin from `RTSI0` through `RTSI6`.

See Also

Properties

`ExternalClockSource`

Purpose

Specify which signal generates an analog output update across channels

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`ExternalClockSource` specifies the pin whose signal is used as the clock to update analog outputs across a group of channels. This property is in effect when the `ClockSource` property is set to **External**.

Note The National Instruments term for this clock is AO Sample Clock.

Characteristics

Vendor	National Instruments
Usage	AO
Access	Read/write
Data type	String
Read-only when running	Yes

Values

`PFI0` to `PFI15`	Use specified pin from `PFI0` through `PFI15`.
`RTSI0` to `RTSI6`	Use specified pin from `RTSI0` through `RTSI6`.

See Also

Properties

`ClockSource`

ExternalSampleClockDriveLine property

Purpose

Specify which signal line is driven by the clock for sample conversions on each channel

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`ExternalSampleClockDriveLine` defines which pin is pulsed when conversions occur on each channel. Data acquisition cards with simultaneous sample and hold ignore this property. You can use this property to synchronize the operations of multiple cards over the RTSI bus or via external PFI pins.

Note The National Instruments term for this clock is AI Convert Clock.

Characteristics

Vendor	National Instruments
Usage	AI
Access	Read/write
Data type	String
Read-only when running	Yes

Values

`PFI0` to `PFI15`	Use specified pin from `PFI0` through `PFI15`.
`RTSI0` to `RTSI6`	Use specified pin from `RTSI0` through `RTSI6`.

See Also

Properties

`ExternalSampleClockSource`

Purpose

Specify which signal provides clock for sample conversions across channels

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`ExternalSampleClockSource` specifies the pin whose signal is used as the channel clock for conversions on each channel. This property is in effect when the `ClockSource` property is set to `ExternalSampleCtrl` or `ExternalSampleAndScanCtrl`.

Data acquisition cards with simultaneous sample and hold ignore this property.

Note The National Instruments term for this clock is AI Convert Clock.

Characteristics

Vendor	National Instruments
Usage	AI
Access	Read/write
Data type	String
Read-only when running	Yes

Values

`PFI0` to `PFI15`	Use specified pin from `PFI0` through `PFI15`.
`RTSI0` to `RTSI6`	Use specified pin from `RTSI0` through `RTSI6`.

See Also

Properties

`ClockSource`, `ExternalScanClockSource`

Purpose

Specify which signal is driven by the clock indicating the start of a series of conversions across channels

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

`ExternalScanClockDriveLine` defines which pin is pulsed when a series of conversions across channels start. You can use this property to synchronize the operations of multiple cards over the RTSI bus or via external PFI pins.

Note The National Instruments term for this clock is AI Sample Clock.

Characteristics

Vendor	National Instruments
Usage	AI
Access	Read/write
Data type	String
Read-only when running	Yes

Values

`PFI0` to `PFI15`	Use specified pin from `PFI0` through `PFI15`.
`RTSI0` to `RTSI6`	Use specified pin from `RTSI0` through `RTSI6`.

See Also

Properties

`ExternalScanClockSource`

ExternalScanClockSource property

Purpose

Specify which signal starts series of conversions across channels

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`ExternalScanClockSource` specifies the pin whose signal is used as the scan clock to initiate conversions across a group of channels. This property is in effect when the `ClockSource` property is set to `ExternalScanCtrl` or `ExternalSampleAndScanCtrl`.

Note The National Instruments term for this clock is AI Sample Clock.

Characteristics

Vendor	National Instruments
Usage	AI
Access	Read/write
Data type	String
Read-only when running	Yes

Values

`PFI0` to `PFI15`	Use specified pin from `PFI0` through `PFI15`.
`RTSI0` to `RTSI6`	Use specified pin from `RTSI0` through `RTSI6`.

See Also

Properties

`ClockSource`, `ExternalSampleClockSource`

Purpose

Specify which signal line is driven with a pulse when data acquisition or generation starts

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`ExternalTriggerDriveLine` defines which pin is pulsed when a data acquisition or generation starts. You can use this property to synchronize the operations of multiple cards over the RTSI bus or via external PFI pins.

Characteristics

Vendor	National Instruments
Usage	AI
Access	Read/Write
Data type	String
Read-only when running	Yes

Values

`PFI0` to `PFI15`	Use specified pin from `PFI0` through `PFI15`.
`RTSI0` to `RTSI6`	Use specified pin from `RTSI0` through `RTSI6`.

See Also

Properties

`HwDigitalTriggerSource`

HwDigitalTriggerSource property

Purpose

Specify which signal initiates data acquisition

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`HwDigitalTriggerSource` defines which pin is used to initiate a data acquisition when the `TriggerType` property is set to `HwDigital`.

Characteristics

Vendor	National Instruments
Usage	AI, AO
Access	Read/write
Data type	String
Read-only when running	Yes

Values

`PFI0` to `PFI15`	Use specified pin from `PFI0` through `PFI15`.
`RTSI0` to `RTSI6`	Use specified pin from `RTSI0` through `RTSI6`.

See Also

Properties

`TriggerType`

Purpose

Specify number of external multiplexer devices connected

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`NumMuxBoards` specifies the number of AMUX-64T multiplexer devices connected to your hardware. `NumMuxBoards` can be `0`, `1`, `2`, or `4`. If you are using a 1200 Series board, then `NumMuxBoards` can only be `0`.

Characteristics

Vendor	National Instruments Traditional NI-DAQ devices
Usage	AI, common to all channels
Access	Read/write
Data type	Double
Read-only when running	No

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. See the supported hardware page at `www.mathworks.com/products/daq/supportedio.html` for more information.

Values

`{0}`, `1`, `2`, or `4`	The number of AMUX-64T multiplexer devices connected.

OutOfDataMode property

Purpose

Specify how value held by analog output subsystem is determined

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

When queued data is output to the analog output (AO) subsystem, the hardware typically holds a value. For National Instruments and Measurement Computing devices, the value held is determined by `OutOfDataMode`.

`OutOfDataMode` can be `Hold` or `DefaultValue`. If `OutOfDataMode` is `Hold`, then the last value output is held by the AO subsystem. If `OutOfDataMode` is `DefaultValue`, then the value specified by the `DefaultChannelValue` property is held by the AO subsystem.

Characteristics

Vendor	Measurement Computing, National Instruments
Usage	AO, common to all channels
Access	Read/write
Data type	String
Read-only when running	Yes

Values

`{Hold}`	Hold the last output value.
`DefaultValue`	Hold the value specified by `DefaultChannelValue`.

Examples

Create the analog output object ao and add two channels to it.

```
ao = analogoutput('nidaq','Dev1');
addchannel(ao,0:1);
```

You can configure ao so that when queued data is finished being output, a value of 1 volt is held for both channels.

```
ao.OutOfDataMode = 'DefaultValue';
ao.Channel.DefaultChannelValue = 1.0;
```

See Also

Properties

DefaultChannelValue

PortAddress property

Purpose

Indicate base address of parallel port

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

The PC supports up to three parallel ports that are assigned the labels LPT1, LPT2, and LPT3. You can use any of these standard ports as long as they use the usual base addresses, which are (in hex) 378, 278, and 3BC, respectively.

Additional ports, or standard ports not assigned the usual base addresses, are not accessible by the toolbox. Note that most PCs that support MATLAB will include a single parallel printer port with base address 378 (LPT1).

Note The Parallel Port adaptor will be deprecated in a future version of the toolbox. If you create a Data Acquisition Toolbox™ object for `'parallel'` beginning in R2008b, you will receive a warning stating that this adaptor will be removed in a future release. See the supported hardware page at `www.mathworks.com/products/daq/supportedio.html` for more information.

Characteristics

Vendor	Parallel port
Usage	DIO, common to all lines
Access	Read only
Data type	String
Read-only when running	Yes

Values

The value is automatically defined when the object is created.

Examples

Create a digital I/O object for parallel port LPT1 and return the PortAddress value.

```
dio = digitalio('parallel','LPT1');
get(dio,'PortAddress')

ans =
0x378
```

The returned value indicates that LPT1 uses the usual base address.

StandardSampleRates property

Purpose

Specify whether valid sample rates snap to small set of standard values, or if you can set sample rate to any allowed value

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

`StandardSampleRates` can be `On` of `Off`. If `StandardSampleRates` is `Off`, then it is possible to set the sample rate to any value within the bounds supported by the hardware. For most sound cards, the lower bound is 8.000 kHz, while the upper bound is 44.1 kHz. For newer sound cards, an upper bound of 96.0 kHz might be supported. The specified sample rate is rounded up to the next integer value.

If `StandardSampleRates` is `On`, then the available sample rates snap to a small set of standard values. The standard values are 8.000 kHz, 11.025 kHz, 22.050 kHz, and 44.100 kHz. If you specify a sampling rate that is within one percent of a standard value, then the sampling rate snaps to that standard value. If you specify a sampling rate that is not within one percent of a standard value, then the sampling rate rounds up to the closest standard value.

Regardless of the `StandardSampleRates` value, if you specify a sampling rate that is outside the allowed limits, then an error is returned.

Characteristics

Vendor	Sound cards
Usage	AI, AO, common to all channels
Access	Read/write
Data type	String
Read-only when running	Yes

Values

`On`	The sample rate can be set only to a small set of standard values.
`{Off}`	If supported by the hardware, the sample rate can be set to any value within the allowed bounds, up to a maximum of 96.0 kHz.

TransferMode property

Purpose

Specify how data is transferred from data acquisition device to system memory

Description

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

For National Instruments NI-DAQmx hardware, this property is ignored. The device driver automatically selects the most efficient transfer mode available.

For National Instruments Traditional NI-DAQ hardware, `TransferMode` can be `Interrupts` or `SingleDMA` for both analog input and analog output subsystems. If `TransferMode` is `Interrupts`, then data is transferred from the hardware first-in, first-out memory buffer (FIFO) to system memory using interrupts. If `TransferMode` is `SingleDMA`, then data is transferred from the hardware FIFO to system memory using a single direct memory access (DMA) channel. Some boards also support a `TransferMode` of `DualDMA` for analog input subsystems. For example, the AT-MIO-16E-1 board supports this transfer mode. If `TransferMode` is `DualDMA`, then data is transferred from the hardware FIFO to system memory using two DMA channels. Depending on your system resources, data transfer via interrupts can significantly degrade system performance.

For Measurement Computing hardware, `TransferMode` can be `Default`, `InterruptPerPoint`, `DMA`, `InterruptPerBlock`, or `InterruptPerScan`. If `TransferMode` is `Default`, the transfer mode is automatically selected by the driver based on the board type and the sampling rate. If `TransferMode` is `InterruptPerPoint`, a single conversion is transferred for each interrupt. You should use this property value if your sampling rate is less the 5 kHz or you specify a small block size for memory buffering (as defined by the `BufferingConfig` property). If `TransferMode` is `DMA`, data is transferred using a single DMA channel. If `TransferMode` is `InterruptPerBlock`, a block of data is transferred for each interrupt. You should use this property value if your sampling rate is greater than 5 kHz and you are using a board that has a fast maximum sampling rate. Note that a data block is defined by the board, and usually corresponds to half the FIFO size. If `TransferMode` is `InterruptPerScan`, data is not transferred until the entire scan is complete. This can only be used when the number of points acquired is less than or equal to the FIFO size. You should use this mode if your sampling rate is higher than the maximum continuous scan rate of the data acquisition device.

Note If your sampling rate is greater than ~5 kHz, you should avoid using interrupts if possible. The recommended `TransferMode` setting for your application will be described in your hardware documentation, and depends on the specific board you are using and your platform configuration.

Characteristics

Vendor	Measurement Computing, National Instruments
Usage	AI, AO, common to all channels
Access	Read/write
Data type	String
Read-only when running	Yes

Values

Advantech

`{InterruptPerPoint}`	Transfer single data points using interrupts.
`InterruptPerBlock`	Transfer a block of data using interrupts (AI only).

Measurement Computing

`{Default}`	The transfer mode is automatically selected by the driver based on the board type and the sampling rate.
`InterruptPerPoint`	Transfer single data points using interrupts.
`DMA`	Transfer data using a single DMA channel (AI only).
`InterruptPerBlock`	Transfer a block of data using interrupts (AI only).
`InterruptPerScan`	Transfer all data when the acquisition is complete (AI only).

National Instruments

`Interrupts`	Transfer data using interrupts.
`SingleDMA`	Transfer data using a single DMA channel.
`DualDMA`	Transfer data using two DMA channels.

This default property value is supplied by the driver. For most devices that support data transfer via interrupts and DMA, `SingleDMA` is the default value.

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. See the supported hardware page at `www.mathworks.com/products/daq/supportedio.html` for more information.

Examples

Set the `TransferMode` property for a National Instruments board before acquiring data.

```
ai = analoginput('nidaq', 1);
set(ai, 'TransferMode', 'SingleDMA');
addchannel(ai, 1:2);
softscope(ai)
```

3

Block Reference

Purpose Acquire data from multiple analog channels of data acquisition device

Library Data Acquisition Toolbox

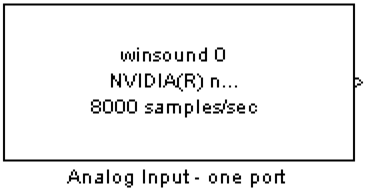
Note You cannot use certain devices with Data Acquisition Toolbox Simulink® blocks. Refer to the Supported Hardware page to see if your device supports Simulink use.

Description

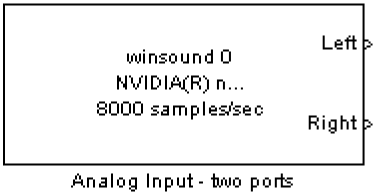
The Analog Input block opens, initializes, configures, and controls an analog data acquisition device. The opening, initialization, and configuration of the device occur once at the start of the model's execution. During the model's run time, the block acquires data either synchronously (deliver the current block of data the device is providing) or asynchronously (buffer incoming data).

Note You need a license for both Data Acquisition Toolbox and Simulink software to use this block.

The block has no input ports. It has one or more output ports, depending on the configuration you choose in the Source Block Parameters dialog box. The following diagram shows the block configured with one port for both channels and with one port for each channel, in the case of a device that has two channels.



Analog Input - one port



Analog Input - two ports

Use the Analog Input block to incorporate live measured data into Simulink for:

- System characterization
- Algorithm verification
- System and algorithm modeling
- Model and design validation
- design control

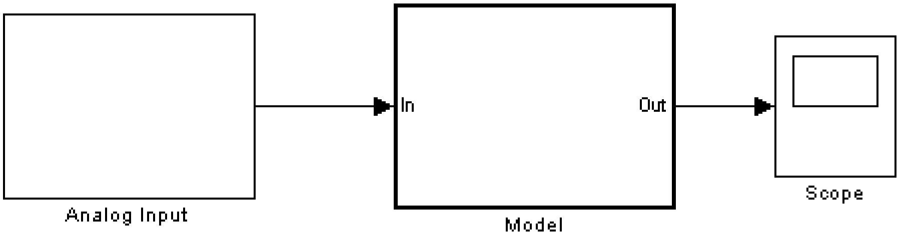
Note You can use the Analog Input block only with devices that support clocked acquisition. The block will error out when the model is run with a device that does not support clocking. To acquire data using devices that do not support clocking, use the Analog Input (Single Sample) block.

You can use this block for signal applications by using it with basic Simulink and DSP System Toolbox™.

You can use the Analog Input block either synchronously or asynchronously. Select the acquisition mode in the Source Block Parameters dialog box.

The following diagram shows the basic analog input usage scenario, in which you would:

- Acquire data at each time step or once per model execution.
- Analyze the data, or use it as input to a system in the model.
- Optionally display results.



For an example of creating a model using the Analog Input block, see Example: Bringing Analog Data into a Model.

Other Supported Features

The Analog Input block supports the use of Simulink Accelerator™ mode. This feature speeds up the execution of Simulink models.

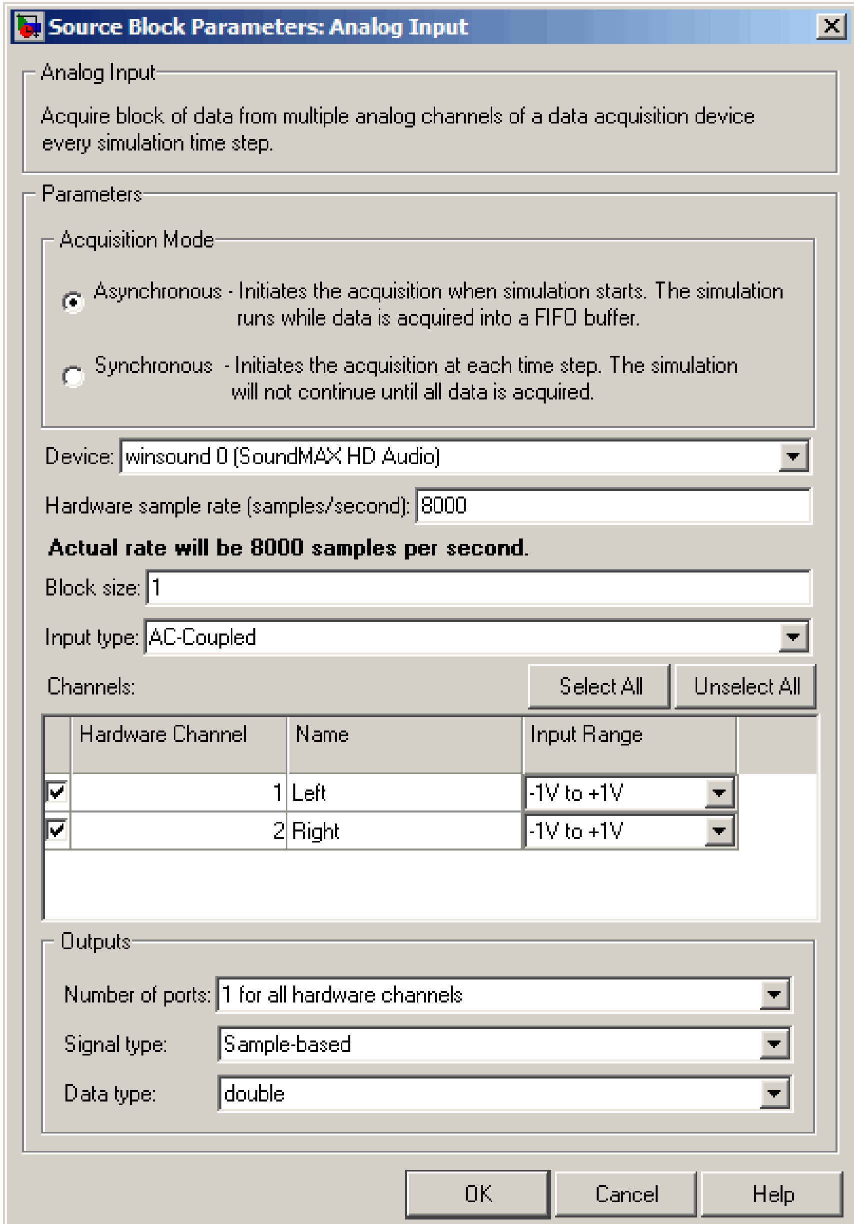
Note You need the C++ Compiler to use Simulink Accelerator mode.

The block supports the use of model referencing. This feature lets your model include other Simulink models as modular components.

For more information on these features, see the Simulink documentation.

Dialog Box

Use the Source Block Parameters dialog box to select your acquisition mode and to set other configuration options.



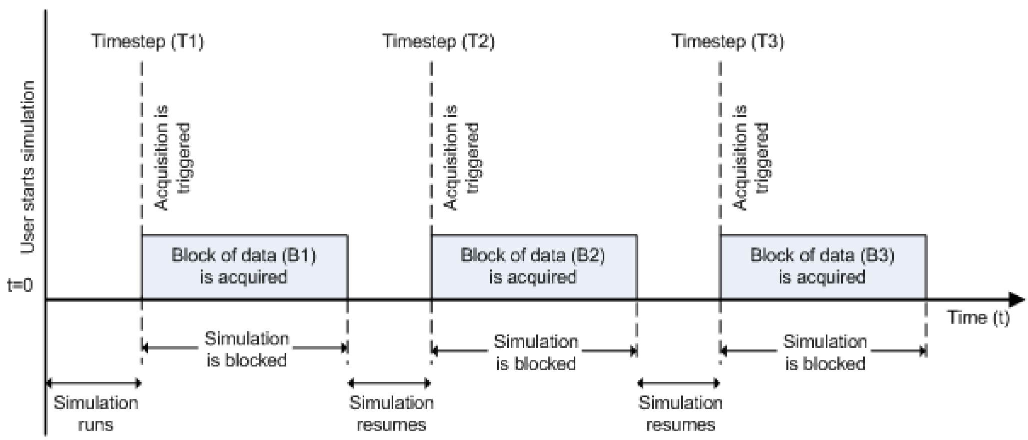
Acquisition Mode

Asynchronous
: Initiates the acquisition when the simulation starts. The simulation runs while data is acquired into a FIFO (First in, First out) buffer. The acquisition is continuous; the block buffers data while outputting a scan/frame of data at each time step.

Synchronous
: Initiates the acquisition at each time step. The simulation will not continue until the requested block of data is acquired. This is unbuffered input; the block will synchronously output the latest scan/frame of data at each time step.

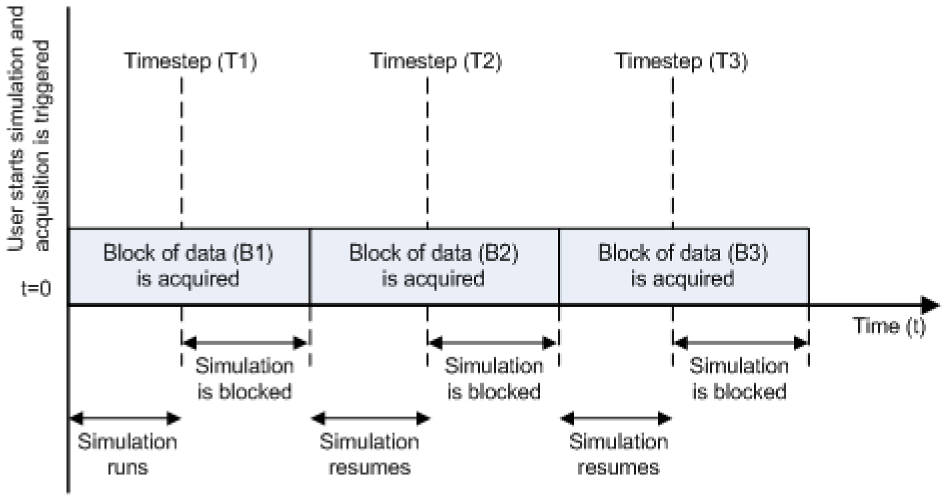
The following diagrams show the difference between synchronous and asynchronous modes for the Analog Input block.

Synchronous Analog Input



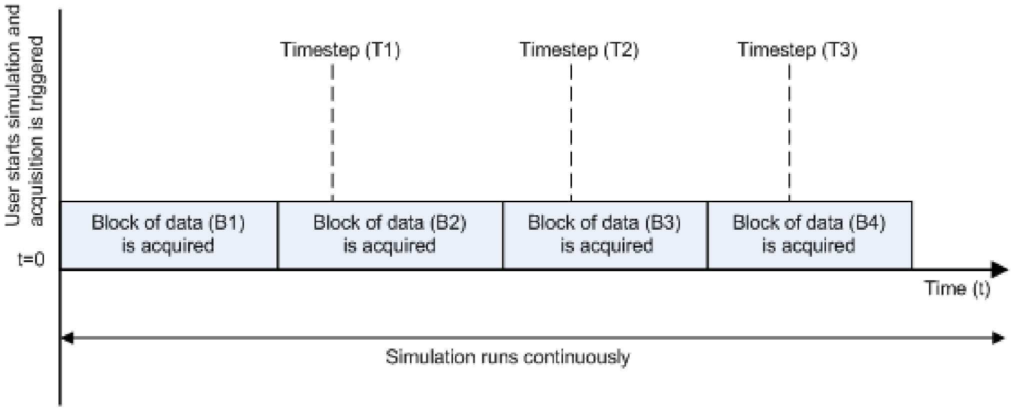
At the first time step (T1), the acquisition is initiated for the required block of data (B1). The simulation does not continue until B1 is completely acquired.

Asynchronous Analog Input – Scenario 1



Scenario 1 shows the case when simulation speed outpaces data acquisition speed. At the first time step (T1), the required block of data (B1) is still being acquired. Therefore, the simulation does not continue until B1 is completely acquired.

Asynchronous Analog Input – Scenario 2



Scenario 2 shows the case when data acquisition speed outpaces simulation speed. At the first time step (T1), the required block of data (B1) has been completely acquired. Therefore, the simulation runs continuously.

Note Several factors, including device hardware and model complexity, can affect the simulation speed, causing both scenarios 1 and 2 to occur within the same simulation.

Options

Device

The data acquisition device from which you want to acquire data. The items in the list vary, depending on which devices you have connected to your system. Devices in the list are specified by adaptor/vendor name and unique device ID, followed by the name of the device. The first available device is selected by default.

Hardware sample rate

The rate at which samples are acquired from the device, in samples per second. This is the sampling time for the hardware. The default is defined when a device is selected.

The sample rate must be a positive real number, and be within the range allowed for the selected hardware.

Block size

The desired number of data samples to output at each time step for each channel. Block size corresponds to the `SamplesPerTrigger` property for an analog input device. The default value for block size depends on the hardware selected. It must be a positive integer, and be within the range allowed for the selected hardware.

Input type

Specifies the hardware channel configuration, such as single-ended, differential, etc. The input type is defined by the capabilities of the selected device.

Channels

The channel configuration table lists your device's hardware channels and lets you configure them. Use the check boxes and selection buttons to specify which channels to acquire data from. These parameters are specified for each selected channel:

Hardware Channel — Displays the hardware channel ID specified by the device. The **Hardware Channel** column is read only and the parameters are defined when the device is selected.

The **Name** — Specifies the channel name. By default the table displays any names provided by the hardware, but you can edit the names. For example, if the device is a sound card with two channels, you can name them `Left` and `Right`.

Input Range — Specifies the input ranges available for each channel supported by the hardware, and is defined when a device is selected.

Outputs

Number of ports

Select **1 for all hardware channels** (default) or **1 per hardware channel**.

Using **1 for all hardware channels** outputs data from a single port as a matrix, with a size of Block size x Number of Channels selected.

Using **1 per hardware channel** outputs data from N ports, where N is equal to the number of selected channels. Each output port will be a column vector with a size of Block size x 1. For naming, each output port will use the channel name if one was specified, or otherwise use [HWChannel + channel ID], for example, `HWChannel2`.

Signal type

Select **Sample-based** or **Frame-based**. This option determines whether the signal type is sample-based or frame-based. **Sample-based** is the default.

Note The **Frame-based** option works only if you have the DSP System Toolbox software installed.

Data type

Select your data type to output from the block. The Analog Input block supports double and native data types, as supported by the hardware. `double` is the default. Native data types will be dynamically populated in this list based on the hardware that is selected. For example, if int16 is a native data type of a specific hardware device, then one of the entries for **Data type** will be `int16 (native)`.

See Also

Analog Input (Single Sample), Analog Output, Analog Output (Single Sample), Digital Input, Digital Output

Analog Input (Single Sample)

Purpose

Acquire single sample from multiple analog channels of data acquisition device

Library

Data Acquisition Toolbox

Note You cannot use certain devices with Data Acquisition Toolbox Simulink blocks. Refer to the Supported Hardware page to see if your device supports Simulink use.

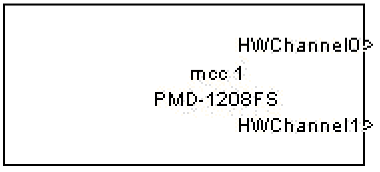
Description

The Analog Input (Single Sample) block opens, initializes, configures, and controls an analog data acquisition device. The opening, initialization, and configuration of the device occur once at the start of the model's execution. The block acquires a single sample every sample time, synchronously from the device, during the model's run time.

Note You need a license for both Data Acquisition Toolbox and Simulink software to use this block.

The block has no input ports. It has one or more output ports, depending on the configuration you choose in the Source Block Parameters dialog box. The following diagram shows the block configured with one port for both channels and with one port for each channel, in the case of a device that has two channels.

Analog Input
(Single Sample) - one port



Analog Input
(Single Sample) - two ports

Use the Analog Input (Single Sample) block to incorporate live measured data into Simulink for:

- System characterization
- Algorithm verification
- System and algorithm modeling
- Model and design validation
- Controls design

Note You can use Analog Input (Single Sample) block only with devices that support single sample acquisition. The block will error out when the model is run with a device that does not support single sample acquisition. To acquire data from devices that do not support acquisition of a single sample (like devices designed for sound and vibration), use the Analog Input block.

You can use the Analog Input (Single Sample) block for signal applications by using it with basic Simulink and DSP System Toolbox.

Other Supported Features

The Analog Input (Single Sample) block supports the use of Simulink Accelerator mode. This feature speeds up the execution of Simulink models.

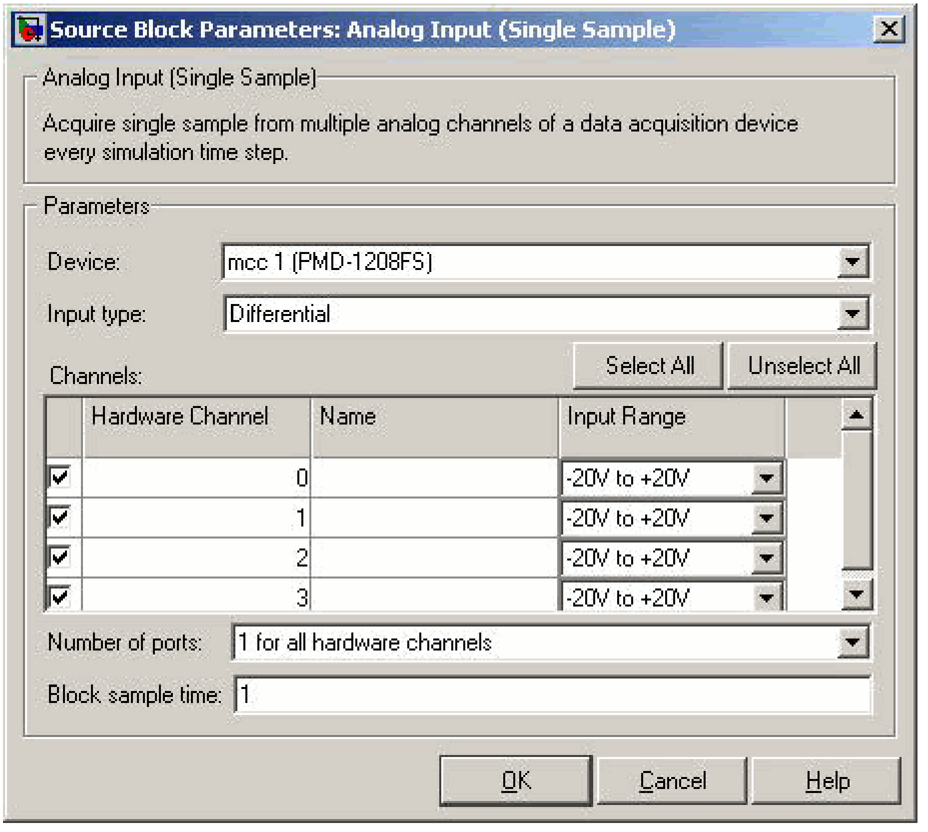
Note You need the C++ Compiler to use Simulink Accelerator mode.

This block supports the use of model referencing. This feature lets your model include other Simulink models as modular components.

For more information on these features, see the Simulink documentation.

Dialog Box

Use the Source Block Parameters dialog box to select your device and to set other configuration options.



Device

The data acquisition device from which you want to acquire data. The items in the list vary, depending on which devices you have connected to your system. Devices in the list are specified by adaptor/vendor name and unique device ID, followed by the name of the device. The first available device is selected by default.

Input type

Specifies the hardware channel configuration, such as single-ended, differential, etc. When you select a device, the device capability defines the available values for input type.

Channels

The channel configuration table lists your device's hardware channels and lets you configure them. Use the check boxes and selection buttons to specify which channels to acquire data from. These parameters are specified for each selected channel:

Hardware Channel — Displays the hardware channel ID specified by the device. The **Hardware Channel** column is read-only and the parameters are defined when the device is selected.

Name — Specifies the channel name. By default the table will display any names provided by the hardware, but you can edit the names. For example, if you are using a device to acquire indoor and outdoor temperature from two channels, you can name them `IndoorTemp` and `OutdoorTemp`.

Input Range — Specifies the input ranges available for each channel supported by the hardware, and the available values are defined when a device is selected.

Number of ports

Select **1 for all hardware channels** (default) or **1 per hardware channel**.

Using **1 for all hardware channels**, outputs data from a single port as a matrix, with a size of [1 x Number of Channels selected].

Using **1 per hardware channel**, outputs data from N ports, where N is equal to the number of selected channels. Each output port will be a scalar value. For naming, each output port will use the channel name if one was specified, or otherwise use ["HWChannel" + channel ID], for example, `HWChannel2`.

Block sample time

Specifies the sample time of the block during the simulation. This is the rate at which the block is executed during simulation. The default value is 0.01 (seconds).

See Also

Analog Input, Analog Output, Analog Output (Single Sample), Digital Input, Digital Output

Purpose

Output data to multiple analog channels of data acquisition device

Library

Data Acquisition Toolbox

Note You cannot use certain devices with Data Acquisition Toolbox Simulink blocks. Refer to the Supported Hardware page to see if your device supports Simulink use.

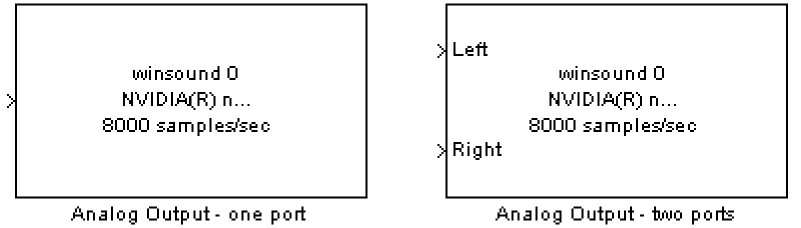
Description

The Analog Output block opens, initializes, configures, and controls an analog data acquisition device. The opening, initialization, and configuration of the device occur once at the start of the model's execution. During the model's run time, the block outputs data to the hardware either synchronously (outputs the block of data as it is provided) or asynchronously (buffers output data).

Note You need a license for both Data Acquisition Toolbox and Simulink software to use this block.

The block has one or more input ports, depending on the option you choose in the Sink Block Parameters dialog box. It has no output ports. The following diagram shows the block configured with one port for both channels and with one port for each channel, in the case of a device that has two channels selected.

Analog Output



Note You can use the Analog Output block only with devices that support clocked generation. The block will error out when the model is run with a device that does not support clocking. To send data using devices that do not support clocking, use the Analog Output (Single Sample) block.

The Analog Output block inherits the sample time from the driving block connected to the input port. The valid data types of the signal at the input port are double or native data types supported by the hardware.

Other Supported Features

The Analog Output block supports the use of Simulink Accelerator mode. This feature speeds up the execution of Simulink models.

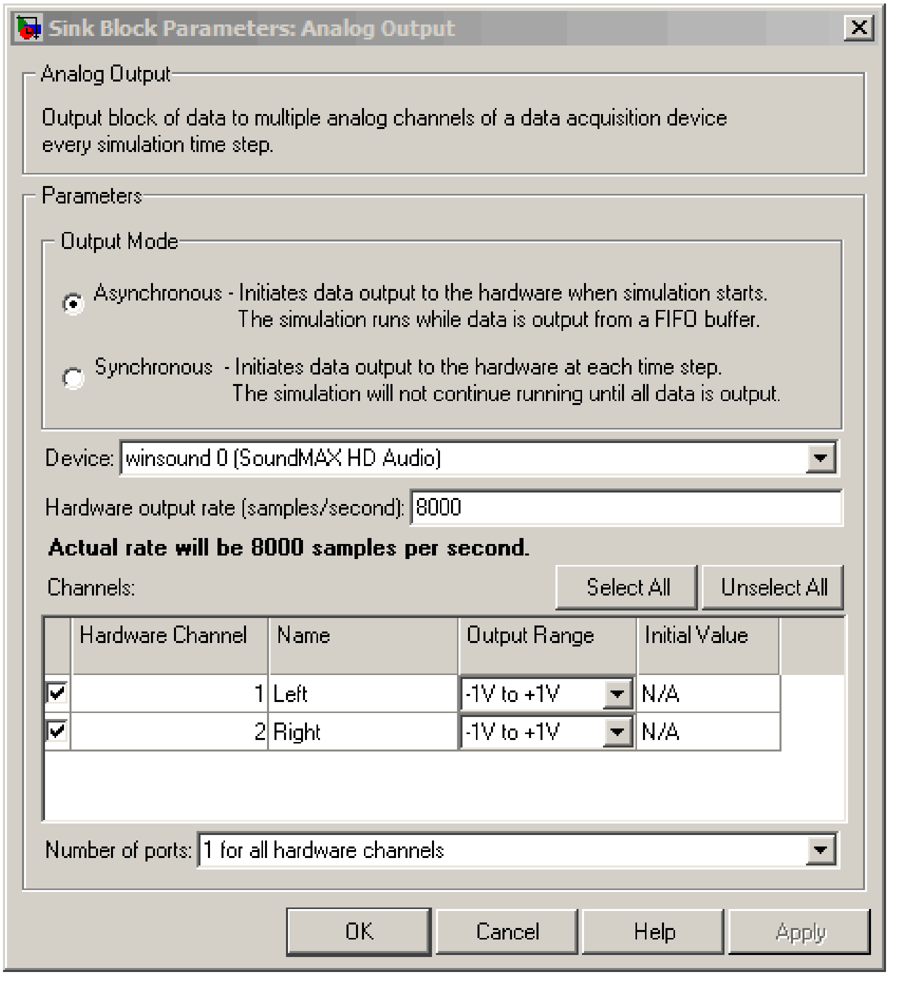
Note You need the C++ Compiler to use Simulink Accelerator mode.

The block supports the use of model referencing. This feature lets your model include other Simulink models as modular components.

For more information on these features, see the Simulink documentation.

Dialog Box

Use the Sink Block Parameters dialog box to select your acquisition mode and to set other configuration options.



Output Mode

Asynchronous

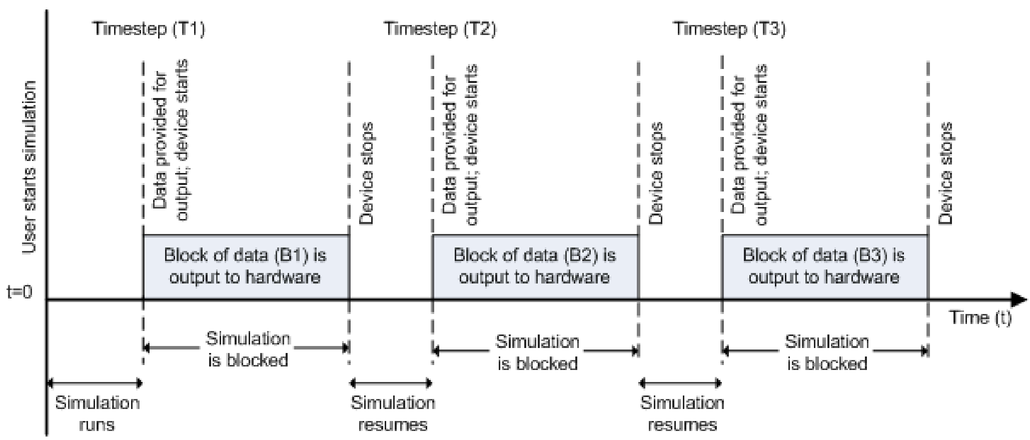
Initiates data output to the hardware when simulation starts. The simulation runs while data is output from a FIFO (First In, First Out) buffer. This mode buffers and outputs data from the block, letting you perform a frame-based or sample-based output.

Synchronous

Initiates data output to the hardware at each time step. The simulation will not continue running until the current block of data is output. In synchronous mode, the block synchronously outputs a vector or frame of samples provided at each time step.

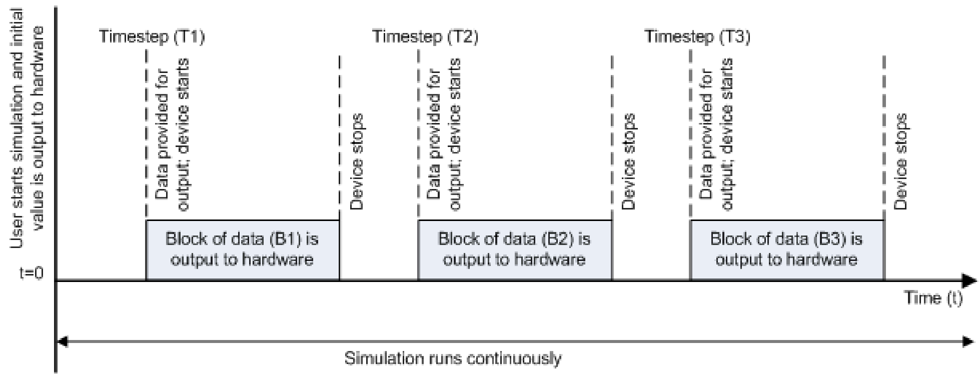
The following diagrams show the difference between synchronous and asynchronous analog output.

Synchronous Analog Output



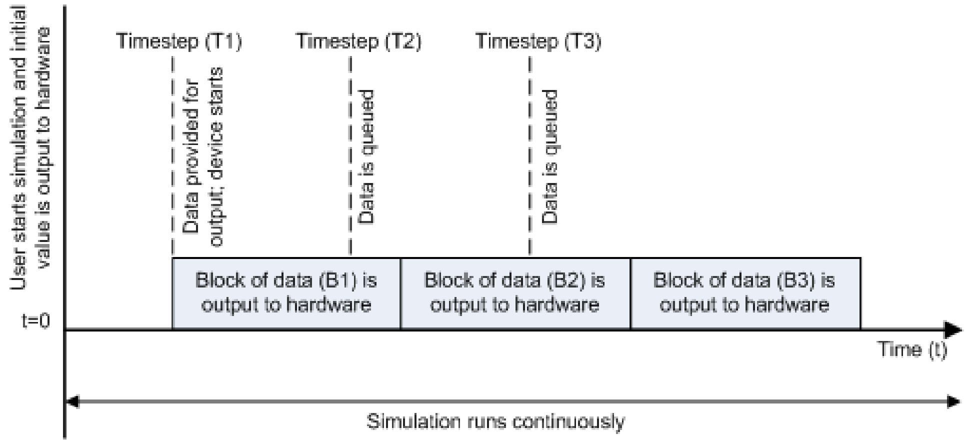
At the first time step (T1), data output is initiated and the corresponding block of data (B1) is output to the hardware. The simulation does not continue until B1 is output completely.

Asynchronous Analog Output – Scenario 1



Scenario 1 shows the case when data output speed outpaces simulation speed. At the first time step (T1), data output is initiated and the corresponding block of data (B1) is output to the hardware. The simulation runs continuously in this mode.

Asynchronous Analog Output – Scenario 2



Scenario 2 shows the case when simulation speed outpaces data acquisition speed. At the first time step (T1), data output is initiated and the corresponding block of data (B1) is output to the hardware. Data is queued at successive time steps and is output to the hardware once the previous block completes. The simulation runs continuously in this mode.

Note Several factors, including device hardware and model complexity, can affect the simulation speed, causing both scenarios 1 and 2 to occur within the same simulation.

Options

Device

The data acquisition device to which you want to output data. The items in the list vary, depending on which devices you have connected to your system. Devices in the list are specified by adaptor/vendor name and unique device ID, followed by the name of the device. The first available device is selected by default.

Hardware output rate

The rate at which samples are output to the device, in samples per second. This output rate for the hardware is defined when a device is selected. The output rate specified must be within the range supported by the selected device.

Channels

The channel configuration table lists your device's hardware channels and lets you configure them. Use the check boxes and selection buttons to specify which channels to send data to.

Hardware Channel — Displays the channel ID specified by the device, and is read only.

Name — specifies the channel name. By default the table displays any names provided by the hardware, but you can edit the names. For example, if the device is a sound card with two channels, you can name them `Left` and `Right`.

Output Range — Specifies the output ranges available for each channel supported by the hardware, and is defined by the selected device.

Initial Value — Specifies the initial value to be output at the start of the simulation, if you are using Asynchronous mode. The default value is `0`. In Synchronous mode, the **Initial Value** column does not appear in the table.

Note For AC-coupled devices like a sound card, this column is not used and is read only.

Number of ports

Select **1 for all hardware channels** (default) or **1 per hardware channel**.

Using **1 for all hardware channels** inputs data from a single port as a matrix, with a size of [S x Number of Channels selected], where S is number of samples provided as input.

Using **1 per hardware channel** inputs data from N ports, where N is equal to the number of selected channels. Each input port will be a column vector with a size of [S x 1], where S is the number of samples provided as an input. For naming, each output port will use the channel name if one was specified, or otherwise use ["HWChannel" + channel ID], for example, `HWChannel2`.

See Also

Analog Input, Analog Input (Single Sample), Analog Output (Single Sample), Digital Input, Digital Output

Purpose

Output single sample to multiple analog channels of data acquisition device

Library

Data Acquisition Toolbox

Note You cannot use certain devices with Data Acquisition Toolbox Simulink blocks. Refer to the Supported Hardware page to see if your device supports Simulink use.

Description

The Analog Output (Single Sample) block opens, initializes, configures, and controls an analog data acquisition device. The opening, initialization, and configuration of the device occur once at the start of the model's execution. The block outputs a single sample every sample time, synchronously to the hardware, during the model's run time.

Note You need a license for both Data Acquisition Toolbox and Simulink software to use this block.

The block has one or more input ports, depending on the option you choose in the Sink Block Parameters dialog box. It has no output ports. The following diagram shows the block configured with one port for both channels and with one port for each channel, in the case of a device that has two channels selected.

Analog Output
(Single Sample) - one port

Analog Output
(Single Sample) - two ports

Note You can use Analog Output (Single Sample) block only with devices that support single sample output. The block will error out when the model is run with a device that does not support single sample acquisition. To send data using devices that do not support acquisition of a single sample (like devices designed for sound and vibration), use the Analog Output block.

The Analog Output (Single Sample) block inherits the sample time from the driving block connected to the input port. The valid data type of the signal at the input port is double.

Other Supported Features

The Analog Output (Single Sample) block supports the use of Simulink Accelerator mode. This feature speeds up the execution of Simulink models.

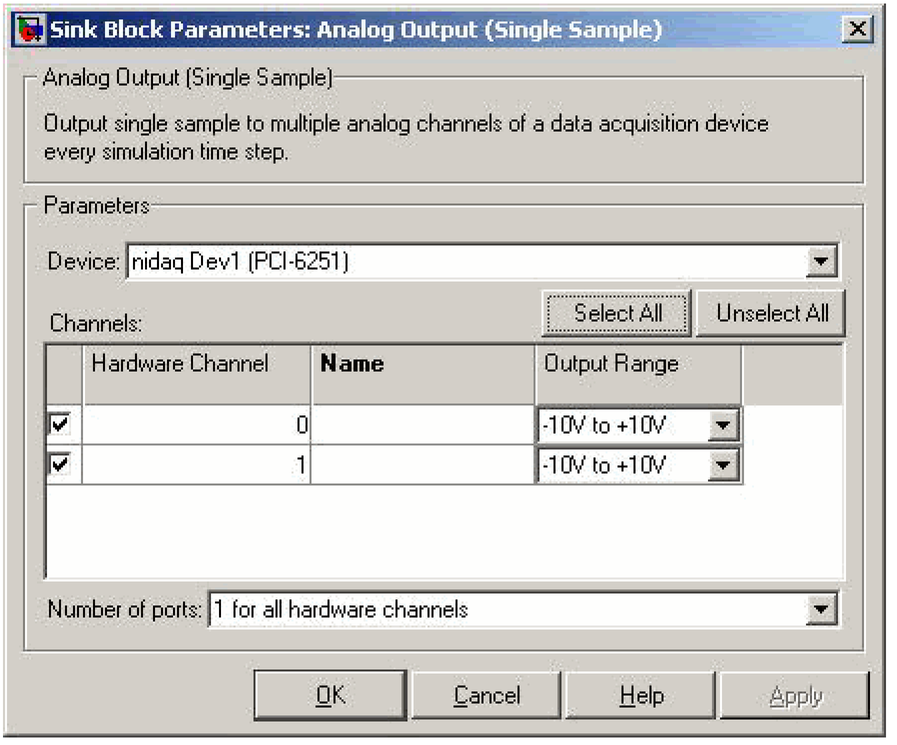
Note You need the C++ Compiler to use Simulink Accelerator mode.

The Analog Output (Single Sample) block supports the use of model referencing. This feature lets your model include other Simulink models as modular components.

For more information on these features, see the Simulink documentation.

Dialog Box

Use the Sink Block Parameters dialog box to select your device and to set other configuration options.



Device

The data acquisition device to which you want to output data. The items in the list vary, depending on which devices you have connected to your system. Devices in the list are specified by adaptor/vendor name and unique device ID, followed by the name of the device. The first available device is selected by default.

Channels

The channel configuration table lists your device's hardware channels and lets you configure them. Use the check boxes and selection buttons to specify which channels to acquire data from. These parameters are specified for each selected channel:

Hardware Channel — Displays the hardware channel ID specified by the device. The **Hardware Channel** column is read-only and the parameters are defined when the device is selected.

Name — Specifies the channel name. By default the table will display any names provided by the hardware, but you can edit the names. For example, if you are sending data and trigger signals to an output device, you can name them `Data` and `TriggerStatus`.

Output Range — Specifies the output ranges available for each channel supported by the hardware, and the available values are defined when a device is selected.

Number of ports

Select **1 for all hardware channels** (default) or **1 per hardware channel**.

Using **1 for all hardware channels**, receives data from a single port as a matrix, with a size of [Block size x Number of Channels selected].

Using **1 per hardware channel**, receives data from N ports, where N is equal to the number of selected channels. Each input port will be a scalar. For naming, each output port will use the channel name if one was specified, or otherwise use ["HWChannel" + channel ID], for example, `HWChannel2`.

See Also

Analog Input, Analog Input (Single Sample), Analog Output, Digital Input, Digital Output

Purpose

Acquire latest set of values from multiple digital lines of data acquisition device

Library

Data Acquisition Toolbox

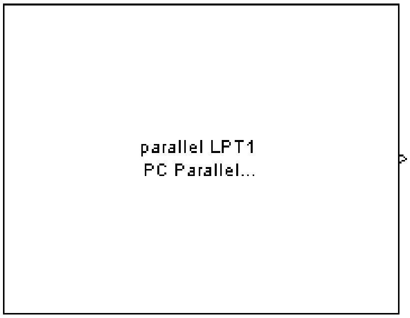
Note You cannot use certain devices with Data Acquisition Toolbox Simulink blocks. Refer to the Supported Hardware page to see if your device supports Simulink use.

Description

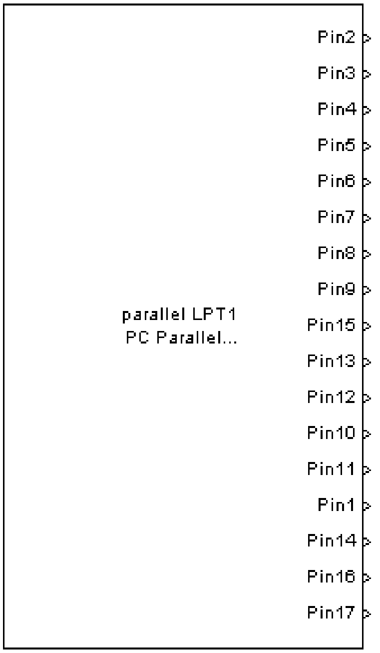
The Digital Input block synchronously outputs the latest scan of data available from the digital lines selected at each simulation time step. It acquires unbuffered digital data, and the data delivered is a binary vector.

Note You need a license for both Data Acquisition Toolbox and Simulink software to use this block.

The block has no input ports. It has one or more output ports, depending on the option you choose in the Source Block Parameters dialog box. The following diagram shows the block configured with one port for all lines and with one port for each line, in the case of a device that has 17 lines selected.



Digital Input - one port for all lines

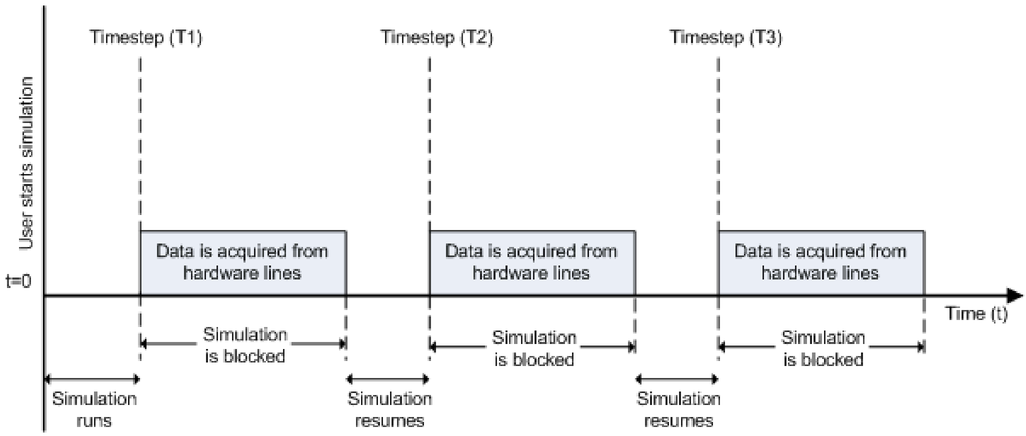


Digital Input - one port for each line

The block inherits the sample time of the model.

The output data is always a binary vector (binvec), i.e., a vector of logical values.

Digital input acquisition is done synchronously. The following diagram shows synchronous digital input.



At the first time step (T1), data is acquired from the selected hardware lines. The simulation does not continue until data is read from all lines.

Other Supported Features

The Digital Input block supports the use of Simulink Accelerator mode. This feature speeds up the execution of Simulink models.

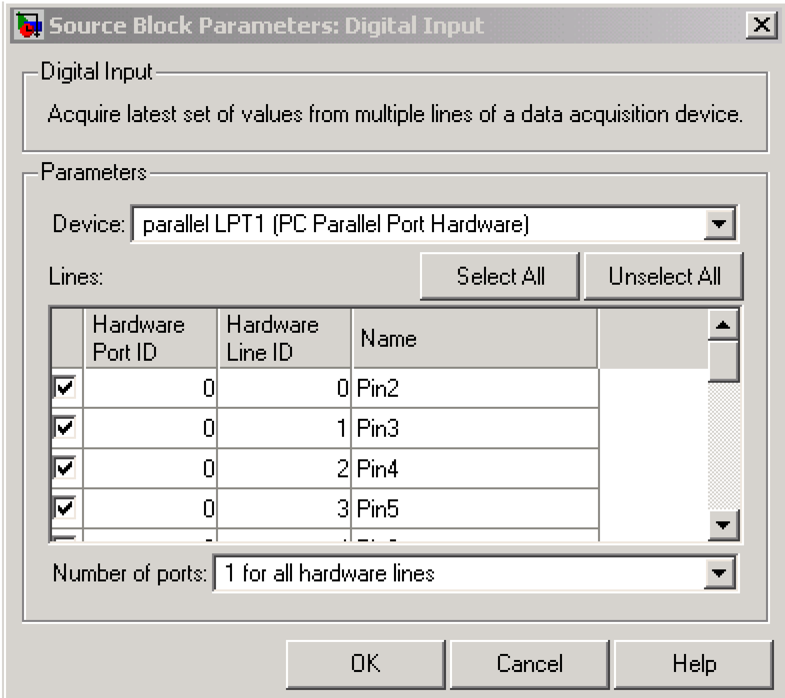
Note You need the C++ Compiler to use Simulink Accelerator mode.

The block supports the use of model referencing. This feature lets your model include other Simulink models as modular components.

For more information on these features, see the Simulink documentation.

Dialog Box

Use the Source Block Parameters dialog box to set configuration options.



Device

The data acquisition device from which you want to acquire data. The items in the list vary, depending on which devices you have connected to your system. Devices in the list are specified by adaptor/vendor name and unique device ID, followed by the name of the device. The first available device is selected by default.

Lines

The line configuration table lists your device's lines and lets you configure them. The table lists all the lines that can be configured for input. Use the check boxes and selection buttons to specify which lines to acquire data from.

Hardware Port ID
Specifies the ID for each hardware port. This is automatically detected and filled in by the selected device, and is read only.

Hardware Line ID
Specifies the ID of the hardware line. This is automatically detected and filled in by the selected device, and is read only.

Name
Specifies the hardware line name. This is automatically detected and filled in from the hardware, though you can edit the name.

Number of ports
Select **1 for all hardware lines** (default) or **1 per hardware line**.

Using **1 for all hardware lines** means that the block will have only one output port for all of the lines that are selected in the table. Data must be [S x number of lines], where S is the number of samples. Data will be a binary vector (binvec).

Using **1 per hardware line** means the block will have one output port per selected line. The name of each output port is the name specified in the table for each line. If no name is provided, the name is "Port" + HwPort ID + "Line" + Line ID. For example, if line 2 of hardware port 3 is selected, and you did not specify a name in the line table, `Port3Line2` appears in the block. Data will be [1 x 1].

See Also

Analog Input, Analog Input (Single Sample), Analog Output, Analog Output (Single Sample), Digital Output

Digital Output

Purpose

Output data to multiple digital lines of data acquisition device

Library

Data Acquisition Toolbox

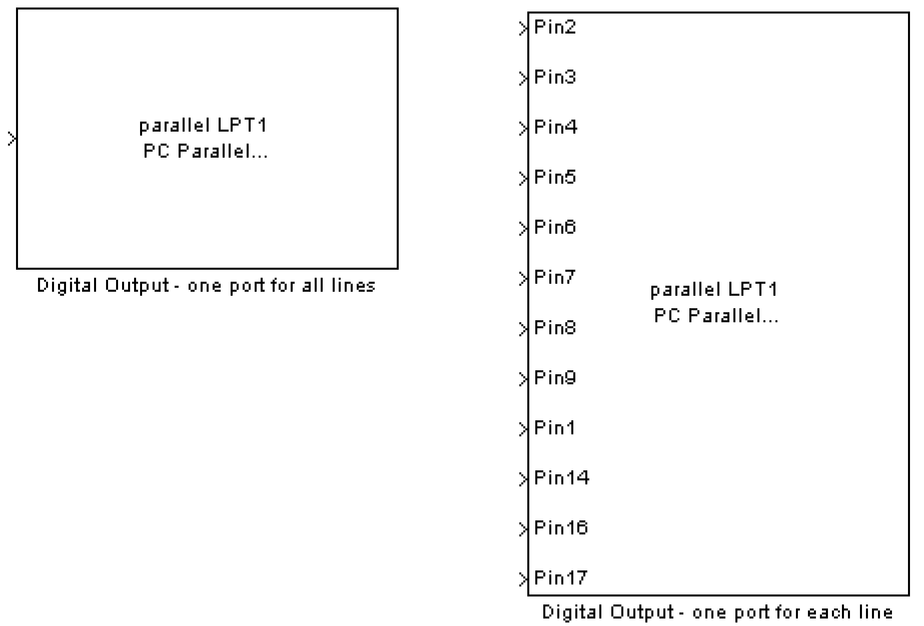
Note You cannot use certain devices with Data Acquisition Toolbox Simulink blocks. Refer to the Supported Hardware page to see if your device supports Simulink use.

Description

The Digital Output block synchronously outputs the latest set of data to the hardware at each simulation time step. It outputs unbuffered digital data. The output data is always a binary vector (binvec).

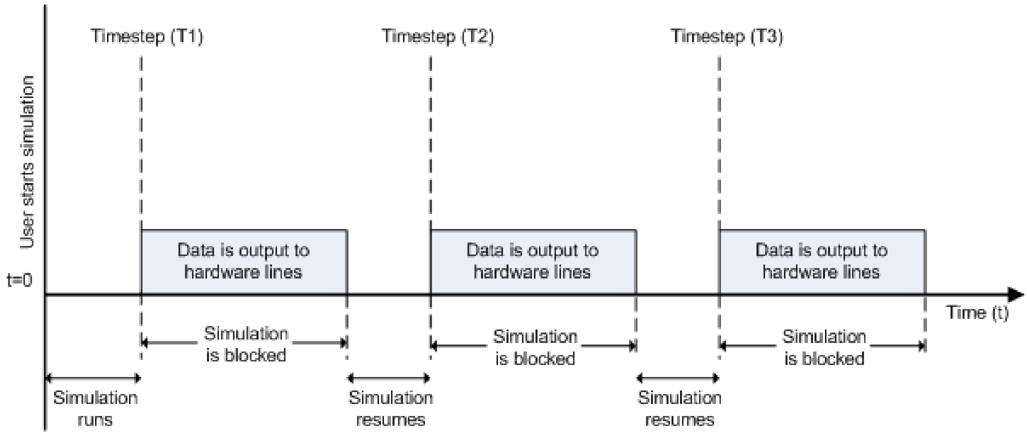
Note You need a license for both Data Acquisition Toolbox and Simulink software to use this block.

The block has no output ports. It can have one or more input ports, depending on the option you choose in the Sink Block Parameters dialog box. The following diagram shows the block configured with one port for all lines and with one port for each line, in the case of a device that has 12 lines selected.



The Digital Output block inherits the sample time from the driving block connected to the input port. The data type of the signal at the input port must be a logical data type.

Digital output is done synchronously. The following diagram shows synchronous digital output.



At the first time step (T1), data is output to the selected hardware lines. The simulation does not continue until data is output to all lines.

Other Supported Features

The Digital Output block supports the use of Simulink Accelerator mode. This feature speeds up the execution of Simulink models.

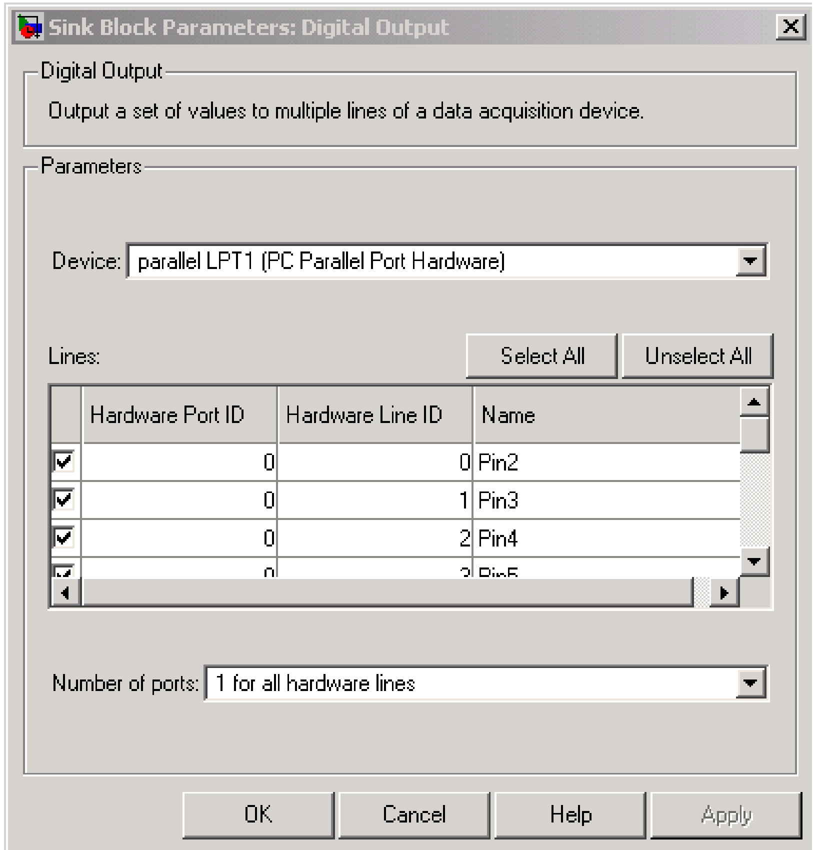
Note You need the C++ Compiler to use Simulink Accelerator mode.

The block supports the use of model referencing. This feature lets your model include other Simulink models as modular components.

For more information on these features, see the Simulink documentation.

Dialog Box

Use the Sink Block Parameters dialog box to set configuration options.



Device

The data acquisition device to which you want to output data. The items in the list vary, depending on which devices you have connected to your system. Devices in the list are specified by adaptor/vendor name and unique device ID, followed by the name of the device. The first available device is selected by default.

Lines

The line configuration table lists your device's lines and lets you configure them. Use the check boxes and selection buttons to specify which lines to send data to.

Hardware Port ID

Specifies the ID for each hardware port. This is automatically detected and filled in by the selected device, and is read only.

Hardware Line ID

Specifies the ID of the hardware line. This is automatically detected and filled in by the selected device, and is read only.

Name

Specifies the hardware line name. This is automatically detected and filled in by the selected device, though you can edit the name.

Number of ports

Select **1 for all hardware lines** (default) or **1 per hardware line**.

Using **1 for all hardware lines** means that the block will have only one input port for all lines selected in the table. Data needs to be [S x number of lines], where S is the number of samples. Data at the input port needs to be a binary vector (binvec).

Using **1 per hardware line** means the block will have one input port per selected line. The name of each input port is the name specified in the table for each line. If no name is provided, the name is "Port" + HwPort ID + Line + Line ID. For example, if line 2 of port 3 is selected, and you did not specify a name in the line table, `Port3Line2` appears in the block. Data needs to be [1 x 1].

See Also

Analog Input, Analog Input (Single Sample), Analog Output, Analog Output (Single Sample), Digital Input

Class Reference

daq.Session	Represent data acquisition session using National Instruments devices

Purpose

Represent data acquisition session using National Instruments devices

Description

The session object configures and controls one or more devices including devices plugged into a CompactDAQ chassis, using the session-based interface. This class is not instantiated directly.

Construction

`s = daq.createSession('vendor')` creates the data acquisition session `s` to work with `vendor` devices. Currently the only supported vendor is National Instruments.

Input Arguments

vendor

Is the ID of the device vendor you want to use. Currently the only supported vendor is `'ni'`.

Properties

ActiveEdge	Rising or falling edges of EdgeCount signals
ActivePulse	Active pulse measurement of PulseWidth counter channel
ADCTimingMode	Set channel timing mode
AutoSyncDSA	Automatically Synchronize DSA devices
BridgeMode	Specify analog input device bridge mode
Channels	Array of channel objects associated with session object
Connections	Array of connections in a session
CountDirection	Specify direction of counter channel
Coupling	Specify input coupling mode

Destination	Indicates trigger destination terminal
Device	Channel device information
Direction	Specify digital channel direction
DurationInSeconds	Specify duration of acquisition
DutyCycle	Duty cycle of counter output channel
EncoderType	Encoding type of counter channel
ExcitationCurrent	Voltage of external source of excitation
ExcitationSource	External source of excitation
ExcitationVoltage	Voltage of excitation source
ExternalTriggerTimeout	Indicate if external trigger timed out
Frequency	Frequency of generated pulses on counter output channel
ID	ID of channel in session
IdleState	Default state of counter output channel
InitialCount	Specify initial count point
InitialDelay	Delay until output channel generates pulses
IsContinuous	Specify if operation continues until manually stopped
IsDone	Indicate if operation is complete
IsLogging	Indicate if hardware is acquiring or generating data
IsNotifyWhenDataAvailableExceedsAuto	Control if is set automatically

IsNotifyWhenScansQueuedBelowAuto	Control if is set automatically
IsRunning	Indicate if operation is still in progress
IsSimulated	Indicate if device is simulated
IsWaitingForExternalTrigger	Indicates if synchronization is waiting for an external trigger
MaxSoundPressureLevel	Sound pressure level for microphone channels
MeasurementType	Type counter channel measurement
Name	Specify descriptive name for the channel
NominalBridgeResistance	Resistance of sensor
NotifyWhenDataAvailableExceeds	Control firing of `DataAvailable` event
NotifyWhenScansQueuedBelow	Control firing of `DataRequired` event
NumberOfScans	Number of scans for operation when starting
R0	Specify resistance value
Range	Specify channel measurement range
Rate	Rate of operation in scans per second
RateLimit	Limit of rate of operation based on hardware configuration
RTDConfiguration	Specify wiring configuration of RTD device
RTDType	Specify sensor sensitivity

ScansAcquired	Number of scans acquired during operation
ScansOutputByHardware	Indicate number of scans output by hardware
ScansQueued	Indicate number of scans queued for output
Sensitivity	Sensitivity of an analog channel
ShuntLocation	Indicate location of channel's shunt resistor
ShuntResistance	Resistance value of channel's shunt resistor
Source	Indicates trigger source terminal
Terminal	PFI terminal of counter subsystem
TerminalConfig	Specify terminal configuration
Terminals	Terminals available on device or CompactDAQ chassis
ThermocoupleType	Select thermocouple type
TriggerCondition	Specify condition that must be satisfied before trigger executes
TriggersPerRun	Indicate the number of times the trigger executes in an operation
TriggersRemaining	Indicates the number of trigger to execute in an operation
TriggerType	Type of trigger executed
Type	Display synchronization trigger type
Units	Specify unit of RTD measurement

Vendor	Vendor information associated with session object
ZResetCondition	Reset condition for Z-indexing
ZResetEnable	Enable reset for Z-indexing
ZResetValue	Reset value for Z-indexing

Methods

addAnalogInputChannel	Add analog input channel
addAnalogOutputChannel	Add analog output channel
addClockConnection	Add clock connection
addCounterInputChannel	Add counter input channel
addCounterOutputChannel	Add counter output channel
addDigitalChannel	Add digital channel
addlistener	Create event listener
addTriggerConnection	Add trigger connection
binaryVectorToDecimal	Convert binary vector value to decimal value
binaryVectorToHex	Convert binary vector value to hexadecimal
DataAvailable	Notify when acquired data is available to process
DataRequired Event	Notify when additional data is required for output on continuous generation
decimalToBinaryVector	Convert decimal value to binary vector
ErrorOccurred Event	Notify when device-related errors occur

hexToBinaryVector	Convert hexadecimal value to binary vector
inputSingleScan	Acquire single scan from all input channels
outputSingleScan	Generate single scan on all output channels
prepare	Prepare session for operation
queueOutputData	Queue data to be output
release	Release session resources
removeChannel	Remove channel from session object
removeConnection	Remove clock or trigger connection
resetCounters	Reset counter channel to initial count
startBackground	Start background operations
startForeground	Start foreground operations
stop	Stop background operation
wait	Block MATLAB until background operation completes

Events

DataAvailable	Notify when acquired data is available to process.
DataRequired	Notify when additional data is required for output on continuous generation.
ErrorOccurred	Notify when device-related errors occur.

Examples

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

See Also

daq.createSession | daq.getDevices | daq.getVendors

5

Functions — Alphabetical List

Purpose

Add hardware channels to analog input or output object

Syntax

```
chans = addchannel(obj,hwch)
chans = addchannel(obj,hwch,index)
chans = addchannel(obj,hwch,'names')
chans = addchannel(obj,hwch,index,'names')
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	An analog input or analog output object.
`hwch`	Specifies the numeric IDs of the hardware channels added to the device object. Any MATLAB vector syntax can be used.
`index`	The MATLAB indices to associate with the hardware channels. Any MATLAB vector syntax can be used provided the vector elements are monotonically increasing.
`'names'`	A descriptive channel name or cell array of descriptive channel names.
`chans`	A column vector of channels with the same length as `hwch`.

Description

`chans = addchannel(obj,hwch)` adds the hardware channels specified by `hwch` to the device object `obj`. The MATLAB indices associated with the added channels are assigned automatically. `chans` is a column vector of channels.

`chans = addchannel(obj,hwch,index)` adds the hardware channels specified by `hwch` to the device object `obj`. `index` specifies the MATLAB indices to associate with the added channels.

`chans = addchannel(obj,hwch,'names')` adds the hardware channels specified by `hwch` to the device object `obj`. The MATLAB indices associated with the added channels are assigned automatically. `names` is a descriptive channel name or cell array of descriptive channel names.

`chans = addchannel(obj,hwch,index,'names')` adds the hardware channels specified by `hwch` to the device object `obj`. `index` specifies the MATLAB indices to associate with the added channels. `names` is a descriptive channel name or cell array of descriptive channel names.

Tips

Rules for Adding Channels

- The numeric values you supply for `hwch` depend on the hardware you access. For National Instruments and Measurement Computing hardware, channels are "zero-based" (begin at zero). For sound cards, channels are "one-based" (begin at one).
- Hardware channel IDs are stored in the `HwChannel` property and the associated MATLAB indices are stored in the `Index` property.
- You can add individual hardware channels to multiple device objects.
- For sound cards, you cannot add a hardware channel multiple times to the same device object.
- You can configure sound cards in one of two ways: mono mode or stereo mode. For mono mode, `hwch` must be `1`. For stereo mode, the first `hwch` value specified must be `1`.

Note If you are using National Instruments AMUX-64T multiplexer boards, you must use the `addmuxchannel` function to add channels.

- When you use the sound card, and only one channel is added to an analog output object the card is put into mono mode. The same signal is output to both channels.

More About MATLAB Indices

Every hardware channel contained by a device object has an associated MATLAB index that is used to reference the channel. Index assignments are made either automatically by `addchannel` or explicitly with the `index` argument and follow these rules:

- If `index` is not specified and no hardware channels are contained by the device object, then the assigned indices automatically start at one and increase monotonically. If hardware channels have already been added to the device object, then the assigned indices automatically start at the next highest index value and increase monotonically.
- If `index` is specified but the indices are previously assigned, then the requested assignment takes precedence and the previous assignment is reindexed to the next available values. If the lengths of `hwch` and `index` are not equal, then an error is returned and no channels are added to the device object.
- The resulting indices begin at one and increase monotonically up to the size of the channel group.
- If you are using scanning hardware, then the indices define the scan order.
- Sound cards cannot be reindexed.

Rules for Adding Channels to National Instruments 1200 Series Boards

When using National Instruments 1200 Series hardware, you need to modify the above rules in these ways:

- Channel IDs are given in reverse order with `addchannel`. For example, to add eight single-ended channels to the analog input object `ai`:

  ```
  addchannel(ai,7:-1:0);
  ```

- The scan order is from the highest ID to the lowest ID (which must be 0).
- There cannot be any gaps in the channel group.

- When channels are configured in differential mode, the hardware IDs are 0, 2, 4, and 6.

More About Descriptive Channel Names

You can assign hardware channels descriptive names, which are stored in the `ChannelName` property. Choosing a unique descriptive name can be a useful way to identify and reference channels. For a single call to `addchannel`, you can

- Specify one channel name that applies to all channels that are to be added
- Specify a different name for each channel to be added

If the number of names specified in a single `addchannel` call is more than one but not equal to the number of channels to be added, then an error is returned. If a channel is to be referenced by its name, then that name must not contain symbols. If you are naming a large number of channels, then the `makenames` function might be useful. If a channel is not assigned a descriptive name, then it must be referenced by index.

A sound card configured in mono mode is automatically assigned the name `Mono`, while a sound card configured in stereo mode is automatically assigned the names `Left` for the first channel and `Right` for the second channel. You can change these default channel names when the device object is created, or any time after the channel is added.

Examples

National Instruments

Suppose you create the analog input object `AI1` for a National Instruments board, and add the first four hardware channels (channels 0-3) to it.

```
AI1 = analoginput('nidaq','Dev1');
addchannel(AI1,0:3);
```

The channels are automatically assigned the indices 1-4. If you want to add the first four hardware channels to `AI1` and assign descriptive names to the channels,

```
addchannel(AI1,0:3,{'chan1','chan2','chan3','chan4'});
```

Note that you can use the makenames function to create a cell array of channel names. If you add channels 4, 5, and 7 to the existing channel group,

```
addchannel(AI1,[4 5 7]);
```

the new channels are automatically assigned the indices 5-7. Suppose instead you add channels 4, 5, and 7 to the channel group and explicitly assign them indices 1-3.

```
addchannel(AI1,[4 5 7],1:3);
```

The new channels are assigned the indices 1-3, and the previously defined channels are reindexed as indices 4-7. However, if you assigned channels 4, 5, and 7 to indices 6-8, an error is returned because there is a gap in the indices (index 5 has no associated hardware channel).

Sound Card

Suppose you create the analog input object AI1 for a sound card. Most sound cards have only two channels that can be added to a device object. To configure the sound card to operate in mono mode, you must specify hwch as 1.

```
AI1 = analoginput('winsound');
addchannel(AI1,1);
```

The ChannelName property is automatically assigned the value Mono. You can now configure the sound card to operate in stereo mode by adding the second channel.

```
addchannel(AI1,2);
```

The ChannelName property is assigned the values Left and Right for the two hardware channels. Alternatively, you can configure the sound card to operate in stereo mode with one call to addchannel.

```
addchannel(AI1,1:2);
```

See Also

`delete` | `makenames` | `ChannelName` | `HwChannel` | `Index`

addline

Purpose

Add hardware lines to digital I/O object

Syntax

```
lines = addline(obj,hwline,'direction')
lines = addline(obj,hwline,port,'direction')
lines = addline(obj,hwline,'direction','names')
lines = addline(obj,hwline,port,'direction','names')
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	A digital I/O object.
`hwline`	The numeric IDs of the hardware lines added to the device object. Any MATLAB vector syntax can be used.
`'`*`direction`*`'`	The line directions can be `In` or `Out`, and can be specified as a single value or a cell array of values.
`port`	The numeric IDs of the digital I/O port.
`'names'`	A descriptive line name or cell array of descriptive line names.
`lines`	A column vector of lines with the same length as `hwline`.

Description

`lines = addline(obj,hwline,'`*`direction`*`')` adds the hardware lines specified by `hwline` to the digital I/O object `obj`. *`direction`* configures the lines for either input or output. `lines` is a row vector of lines.

`lines = addline(obj,hwline,port,'`*`direction`*`')` adds the hardware lines specified by `hwline` from the port specified by `port` to the digital I/O object `obj`.

`lines = addline(obj,hwline,'`*`direction`*`','names')` adds the hardware lines specified by `hwline` to the digital I/O object `obj`. `names` is a descriptive line name or cell array of descriptive line names.

`lines = addline(obj,hwline,port,'direction','names')` adds the hardware lines specified by `hwline` from the port specified by `port` to the digital I/O object `obj`. `direction` configures the lines for either input or output. `names` is a descriptive line name or cell array of descriptive line names.

You cannot configure lines independently on devices that use the NI-DAQmx adaptor. Refer to "Line and Port Characteristics" for more information about line configurable devices.

Tips

Rules for Adding Lines

- The numeric values you supply for `hwline` depend on the hardware you access. For National Instruments and Measurement Computing hardware, line IDs are "zero-based" (begin at zero).
- You can add a line only once to a given digital I/O object.
- Hardware line IDs are stored in the `HwLine` property and the associated MATLAB indices are stored in the `Index` property.
- For a single call to `addline`, you can add multiple lines from one port or the same line ID from multiple ports. You cannot add multiple lines from multiple ports.
- If a port ID is not explicitly referenced, lines are added first from port 0, then from port 1, and so on.
- You can specify the line directions as a single value or a cell array of values. If a single direction is specified, then all added lines have that direction. If supported by the hardware, you can configure individual lines by supplying a cell array of directions.

More About MATLAB Indices

Every hardware line contained by a device object has an associated MATLAB index that is used to reference the line. Index assignments are made automatically by `addline` and follow these rules:

- If no hardware lines are contained by the device object, then the assigned indices automatically start at one and increase monotonically. If hardware lines have already been added to the device object, then the assigned indices automatically start at the next highest index value and increase monotonically.

- The resulting indices begin at one and increase monotonically up to the size of the line group.
- The first indexed line represents the least significant bit (LSB) and the highest indexed line represents the most significant bit (MSB).

More About Descriptive Line Names

You can assign hardware lines descriptive names, which are stored in the `LineName` property. Choosing a unique descriptive name can be a useful way to identify and reference lines. For a single call to `addline`, you can

- Specify one line name that applies to all lines that are to be added
- Specify a different name for each line to be added

If the number of names specified in a single `addline` call is more than one but differs from the number of lines to be added, then an error is returned. If a line is to be referenced by its name, then that name must not contain symbols. If you are naming a large number of lines, then the `makenames` function might be useful. If a line is not assigned a descriptive name, then it must be referenced by index.

Examples

Create the digital I/O object `dio` and add the first four hardware lines (line IDs 0-3) from port 0.

```
dio = digitalio('nidaq','Dev1');
addline(dio,0:3,'in');
```

These lines are automatically assigned the indices 1-4. If you want to add the first four hardware lines to `dio` and assign descriptive names to the lines,

```
addline(dio,0:3,'in',{'line1','line2','line3','line4'});
```

Note that you can use the `makenames` function to create a cell array of line names. You can add the first four hardware lines (line IDs 0-3) from port 1 to the existing line group.

```
addline(dio,0:3,1,'out');
```

The new lines are automatically assigned the indices 5-8.

See Also

`delete` | `makenames` | `HwLine` | `Index` | `LineName`

addmuxchannel

Purpose

Add hardware channels to analog input objects when using National Instruments multiplexer board

Syntax

```
addmuxchannel(obj)
addmuxchannel(obj,chanids)
chans = addmuxchannel(...)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	An analog input object associated with a National Instruments Traditional NI-DAQ board.
`chanids`	The hardware channel IDs.
`chans`	The channels that are added to `obj`.

Description

`addmuxchannel(obj)` adds as many channels to `obj` as is physically possible based on the number of National Instruments AMUX-64T multiplexer (mux) boards specified by the `NumMuxBoards` property. For one mux board, 64 channels are added. For two mux boards, 128 channels are added. For four mux boards, 256 channels are added.

`addmuxchannel(obj,chanids)` adds the channels specified by `chanids` to `obj`. `chanids` refers to the hardware channel IDs of the data acquisition board.

The actual number of channels added to `obj` depends on the number of mux boards used. For example, suppose you are using a data acquisition board with 16 channels connected to one mux board. If `chanid` is 0, then `addmuxchannel` adds four channels. Refer to the *AMUX-64T User Manual* for more information about adding mux channels based on hardware channel IDs and the number of mux boards used.

`chans = addmuxchannel(...)` returns the channels added to `chans`.

Tips

This function is not available for National Instruments NI-DAQmx boards.

Before using `addmuxchannel`, you must set the `NumMuxBoards` property to the appropriate value. You can use as many as four mux boards with one analog input object. `addmuxchannel` deletes all channels contained by `obj` before new channels are added.

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. See the supported hardware page at `www.mathworks.com/products/daq/supportedio.html` for more information.

See Also

`muxchanidx`

Purpose

Create analog input object

Syntax

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

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Description

`AI = analoginput('adaptor')` creates the analog input object `AI` for a sound card having an `ID` of 0 (*`adaptor`* must be `winsound`). This is the only case where `ID` is not required.

`AI = analoginput('adaptor',ID)` creates the analog input object `AI` for the specified `adaptor` and for the hardware device with device identifier `ID`. `ID` can be specified as an integer or a string.

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. See the supported hardware page at `www.mathworks.com/products/daq/supportedio.html` for more information.

Tips

Creating Analog Input Objects

- When an analog input object is created, it does not contain any hardware channels. To execute the device object, hardware channels must be added with the `addchannel` function.
- You can create multiple analog input objects that are associated with a particular analog input subsystem. However, you can typically execute only one object at a time.

- The analog input object exists in the data acquisition engine and in the MATLAB workspace. If you create a copy of the device object, it references the original device object in the engine.
- If `ID` is a numeric value, then you can specify it as an integer or a string. If `ID` contains any nonnumeric characters, then you must specify it as a string. (See the National Instruments example below.)
- The `Name` property is automatically assigned a descriptive name that is produced by concatenating *`adaptor`*, `ID`, and `-AI`. You can change this name at any time.

Notes When you create an analog input object, it consumes system resources. To avoid this issue, make sure that you do not create objects in a loop. If you must create objects in a loop, make sure you delete them within the loop.

Hardware Device Identifier

When data acquisition devices are installed, they are assigned a unique number which identifies the device in software. The device identifier is typically assigned automatically and can usually be manually changed using a vendor-supplied device configuration utility. National Instruments refers to this identifier as the device name.

For sound cards, the device identifier is typically not exposed to you through the Microsoft® Windows® environment. However, Data Acquisition Toolbox software automatically associates each sound card with an integer `ID` value. There are two cases to consider:

- If you have one sound card installed, then `ID` is 0. You are not required to specify `ID` when creating an analog input object associated with this device.
- If you have multiple sound cards installed, the first one installed has an `ID` of 0, the second one installed has an `ID` of 1, and so on. You must specify `ID` when creating analog input objects associated with devices not having an `ID` of 0.

There are two ways you can determine the `ID` for a particular device:

- Type `daqhwinfo('adaptor')`.
- Execute the vendor-supplied device configuration utility.

Input Arguments

adaptor

The hardware driver adaptor name. The supported adaptors are `advantech`, , `mcc`, `nidaq`, and `winsound`.

ID

The hardware device identifier. `ID` is optional if the device object is associated with a sound card having an `ID` of 0.

Output Arguments

AI

The analog input object.

Properties

Basic Setup

`SampleRate`	Specify per-channel rate at which analog data is converted to digital data, or vice versa
`SamplesPerTrigger`	Specify number of samples to acquire for each channel group member for each trigger that occurs
`TriggerType`	Specify type of trigger to execute

Channel Properties

`ChannelName`	Specify descriptive channel name
`HwChannel`	Specify hardware channel ID
`HwLine`	Specify hardware line ID

`Index`	MATLAB index of hardware channel or line
`InputRange`	Specify range of analog input subsystem
`NativeOffset`	Indicate offset to use when converting between native data format and doubles
`NativeScaling`	Indicate scaling to use when converting between native data format and doubles
`Parent`	Indicate parent (device object) of channel or line
`SensorRange`	Specify range of data expected from sensor
`Type`	Indicate device object type, channel, or line
`Units`	Specify engineering units label
`UnitsRange`	Specify range of data as engineering units

Trigger Properties

`InitialTriggerTime`	Absolute time of first trigger
`ManualTriggerHwOn`	Specify hardware device starts at manual trigger
`TriggerChannel`	Specify channel serving as trigger source
`TriggerCondition`	Specify condition that must be satisfied before trigger executes

`TriggerConditionValue`	Specify voltage value(s) that must be satisfied before trigger executes
`TriggerDelay`	Specify delay value for data logging
`TriggerDelayUnits`	Specify units in which trigger delay data is measured
`TriggerFcn`	Specify callback function to execute when trigger occurs
`TriggerRepeat`	Specify number of additional times trigger executes
`TriggersExecuted`	Indicate number of triggers that execute
`TriggerType`	Specify type of trigger to execute

Logging Properties

`LogFileName`	Specify name of disk file information is logged to
`Logging`	Indicate whether data is being logged to memory or disk file
`LoggingMode`	Specify destination for acquired data
`LogToDiskMode`	Specify whether data, events, and hardware information are saved to one or more disk files

Status Properties

`Logging`	Indicate whether data is being logged to memory or disk file
`Running`	Indicate whether device object is running
`SamplesAcquired`	Indicate number of samples acquired per channel
`SamplesAvailable`	Indicate number of samples available per channel in engine

Hardware Configuration Properties

`ChannelSkew`	Specify time between consecutive scanned hardware channels
`ChannelSkewMode`	Specify how channel skew is determined
`ClockSource`	Specify clock that governs hardware conversion rate
`InputType`	Specify analog input hardware channel configuration
`SampleRate`	Specify per-channel rate at which analog data is converted to digital data, or vice versa

Callback Properties

`DataMissedFcn`	Specify callback function to execute when data is missed
`InputOverRangeFcn`	Specify callback function to execute when acquired data exceeds valid hardware range

RuntimeErrorFcn	Specify callback function to execute when run-time error occurs
SamplesAcquired	Indicate number of samples acquired per channel
SamplesAcquiredFcn	Specify callback function to execute when predefined number of samples is acquired for each channel group member
SamplesAcquiredFcnCount	Specify number of samples to acquire for each channel group member before samples acquired event is generated
StartFcn	Specify callback function to execute before device object runs
StopFcn	Specify callback function to execute after device object runs
TimerFcn	Specify callback function to execute when predefined time period passes
TimerPeriod	Specify time period between timer events
TriggerFcn	Specify callback function to execute when trigger occurs

General Purpose Properties

BufferingConfig	Specify per-channel allocated memory
BufferingMode	Specify how memory is allocated
Channel	Contain hardware channels added to device object

`EventLog`	Store information for specific events
`Name`	Specify descriptive name for the channel
`Tag`	Specify device object label
`Timeout`	Specify additional waiting time to extract or queue data
`Type`	Indicate device object type, channel, or line
`UserData`	Store data to associate with device object

Examples

To create an analog input object for a National Instruments device defined as `'Dev1'`:

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

To create an analog input object for a Measurement Computing device defined as `'1'`:

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

Alternatives

See Also

`addchannel` | `daqhwinfo`

Purpose

Create analog output object

Syntax

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

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`'adaptor'`	The hardware driver adaptor name. The supported adaptors are `advantech`, `mcc`, `nidaq`, and `winsound`.
`ID`	The hardware device identifier. `ID` is optional if the device object is associated with a sound card having an `ID` of 0.
`AO`	The analog output object.

Description

`AO = analogoutput('`*`adaptor`*`')` creates the analog output object `AO` for a sound card having an `ID` of 0 (*`adaptor`* must be `winsound`). This is the only case where `ID` is not required.

`AO = analogoutput('`*`adaptor`*`',ID)` creates the analog output object `AO` for the specified `adaptor` and for the hardware device with device identifier `ID`. `ID` can be specified as an integer or a string.

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. See the supported hardware page at `www.mathworks.com/products/daq/supportedio.html` for more information.

Tips

More About Creating Analog Output Objects

- When an analog output object is created, it does not contain any hardware channels. To execute the device object, hardware channels must be added with the `addchannel` function.
- You can create multiple analog output objects that are associated with a particular analog output subsystem. However, you can typically execute only one object at a time.
- The analog output object exists in the data acquisition engine and in the MATLAB workspace. If you create a copy of the device object, it references the original device object in the engine.
- If `ID` is a numeric value, then you can specify it as an integer or a string. If `ID` contains any nonnumeric characters, then you must specify it as a string.
- The `Name` property is automatically assigned a descriptive name that is produced by concatenating *`adaptor`*, `ID`, and `-AO`. You can change this name at any time.

Notes When you create an analog output object, it consumes system resources. To avoid this issue, make sure that you do not create objects in a loop. If you must create objects in a loop, make sure you delete them within the loop.

More About the Hardware Device Identifier

When data acquisition devices are installed, they are assigned a unique number which identifies the device in software. The device identifier is typically assigned automatically and can usually be manually changed using a vendor-supplied device configuration utility. National Instruments refers to this number as the device number.

For sound cards, the device identifier is typically not exposed to you through the Microsoft Windows environment. However, Data Acquisition Toolbox software automatically associates each sound card with an integer `ID` value. There are two cases to consider:

- If you have one sound card installed, then `ID` is 0. You are not required to specify `ID` when creating an analog output object associated with this device.
- If you have multiple sound cards installed, the first one installed has an `ID` of 0, the second one installed has an `ID` of 1, and so on. You must specify `ID` when creating analog output objects associated with devices not having an `ID` of 0.

There are two ways you can determine the `ID` for a particular device:

- Type `daqhwinfo('`*`adaptor`*`')`.
- Execute the vendor-supplied device configuration utility.

Properties

Basic Setup Properties

`SampleRate`	Specify per-channel rate at which analog data is converted to digital data, or vice versa
`TriggerType`	Specify type of trigger to execute

Channel Properties

`ChannelName`	Specify descriptive channel name
`DefaultChannelValue`	Specify value held by analog output subsystem
`HwChannel`	Specify hardware channel ID
`Index`	MATLAB index of hardware channel or line
`NativeOffset`	Indicate offset to use when converting between native data format and doubles
`NativeScaling`	Indicate scaling to use when converting between native data format and doubles

`OutputRange`	Specify range of analog output hardware subsystem
`Parent`	Indicate parent (device object) of channel or line
`Type`	Indicate device object type, channel, or line
`Units`	Specify engineering units label
`UnitsRange`	Specify range of data as engineering units

Trigger Properties

`InitialTriggerTime`	Absolute time of first trigger
`TriggerFcn`	Specify callback function to execute when trigger occurs
`TriggersExecuted`	Indicate number of triggers that execute
`TriggerType`	Specify type of trigger to execute

Status Properties

`Running`	Indicate whether device object is running
`SamplesAvailable`	Indicate number of samples available per channel in engine
`SamplesOutput`	Indicate number of samples output per channel from engine
`Sending`	Indicate whether data is being sent to hardware device

Hardware Configuration Properties

`ClockSource`	Specify clock that governs hardware conversion rate
`SampleRate`	Specify per-channel rate at which analog data is converted to digital data, or vice versa

Data Management Properties

`MaxSamplesQueued`	Indicate maximum number of samples that can be queued in engine
`RepeatOutput`	Specify number of additional times queued data is output
`Timeout`	Specify additional waiting time to extract or queue data

Callback Properties

`RuntimeErrorFcn`	Specify callback function to execute when run-time error occurs
`SamplesOutputFcn`	Specify callback function to execute when predefined number of samples is output for each channel group member
`SamplesOutputFcnCount`	Specify number of samples to output for each channel group member before samples output event is generated
`StartFcn`	Specify callback function to execute before device object runs

StopFcn	Specify callback function to execute after device object runs
TimerFcn	Specify callback function to execute when predefined time period passes
TimerPeriod	Specify time period between timer events
TriggerFcn	Specify callback function to execute when trigger occurs

General Purpose Properties

BufferingConfig	Specify per-channel allocated memory
BufferingMode	Specify how memory is allocated
Channel	Contain hardware channels added to device object
EventLog	Store information for specific events
Name	Specify descriptive name for the channel
OutOfDataMode	Specify how value held by analog output subsystem is determined
Tag	Specify device object label
Type	Indicate device object type, channel, or line
UserData	Store data to associate with device object

Examples

National Instruments

To create an analog output object for a National Instruments device defined as `'Dev1'`:

```
AO = analogoutput('nidaq','Dev1');
```

To create an analog output object for a Measurement Computing device defined as `'1'`:

```
AO = analogoutput('mcc','1');
```

See Also

`addchannel` | `daqhwinfo` | `Name`

Purpose

Convert digital input and output binary vector to decimal value

Syntax

```
out = binvec2dec(bin)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`bin`	A binary vector.
`out`	A double array.

Description

`out = binvec2dec(bin)` converts the binary vector `bin` to the equivalent decimal number and stores the result in `out`. All nonzero binary vector elements are interpreted as a 1.

Tips

A binary vector (binvec) is constructed with the least significant bit (LSB) in the first column and the most significant bit (MSB) in the last column. For example, the decimal number 23 is written as the binvec value [1 1 1 0 1].

Note The binary vector cannot exceed 52 values.

Examples

To convert the binvec value [1 1 1 0 1] to a decimal value:

```
binvec2dec([1 1 1 0 1])
ans =
    23
```

See Also

`dec2binvec`

clear

Purpose

Remove device objects from MATLAB workspace

Syntax

```
clear obj
clear obj.Channel(index)
clear obj.Line(index)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	A device object or array of device objects.
`obj.Channel(index)`	One or more channels contained by `obj`.
`obj.Line(index)`	One or more lines contained by `obj`.

Description

`clear obj` removes `obj` and all associated channels or lines from the MATLAB workspace, but not from the data acquisition engine.

`clear obj.Channel(index)` removes the specified channels contained by `obj` from the MATLAB workspace, but not from the data acquisition engine.

`clear obj.Line(index)` removes the specified lines contained by `obj` from the MATLAB workspace, but not from the data acquisition engine.

Tips

Clearing device objects, channels, and lines follows these rules:

- `clear` does not remove device objects, channels, or lines from the data acquisition engine. Use the `delete` function for this purpose.
- If multiple references to a device object exist in the workspace, clearing one reference will not invalidate the remaining references.
- You can restore cleared device objects to the MATLAB workspace with the `daqfind` function.

If you use the `help` command to display the file help for `clear`, then you must supply the pathname shown below.

```
help daq/private/clear
```

Examples

Create the analog input object `ai`, copy `ai` to a new variable `aicopy`, and then clear the original device object from the MATLAB workspace.

```
ai = analoginput('winsound');
ch = addchannel(ai,1:2);
aicopy = ai;
clear ai
```

Retrieve `ai` from the engine with `daqfind`, and show you that `ai` is identical to `aicopy`.

```
ainew = daqfind;
isequal(aicopy,ainew)
ans =
     1
```

See Also

`daqfind` | `delete`

daqcallback

Purpose

Callback function that displays event information for specified event

Syntax

```
daqcallback(obj,event)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	A device object.
`event`	A variable that captures the event information contained by the `EventLog` property.

Description

`daqcallback(obj,event)` is an example callback function that displays information to the MATLAB Command Window. For all events, the information includes the event type and the name of the device object that caused the event to occur. For events that record the absolute time in `EventLog`, the event time is also displayed. For run-time error events, the error message is also displayed.

Tips

You specify `daqcallback` as the callback function to be executed for any event by specifying it as the value for the associated callback property. For analog input objects, `daqcallback` is the default value for the `DataMissedFcn` and `RuntimeErrorFcn` properties. For analog output objects, `daqcallback` is the default value for the `RuntimeErrorFcn` property.

You can use the `showdaqevents` function to easily display event information captured by the `EventLog` property.

Examples

Create the analog input object `ai` and call `daqcallback` when a trigger event occurs.

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

```
set(ai,'TriggerRepeat',3)
set(ai,'TriggerFcn',@daqcallback)
start(ai)
```

See Also

showdaqevents | DataMissedFcn | EventLog | RuntimeErrorFcn

daqfind

Purpose

Return device objects, channels, or lines from data acquisition engine to MATLAB workspace

Syntax

```
out = daqfind
out = daqfind('PropertyName',PropertyValue,...)
out = daqfind(S)
out = daqfind(obj,'PropertyName',PropertyValue,...)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

'*PropertyName*'	A device object, channel, or line property name.
`PropertyValue`	A device object, channel, or line property value.
`obj`	A device object, array of device objects, channels, or lines.
`S`	A structure with field names that are property names and field values that are property values.
`out`	An array or cell array of device objects, channels, or lines.

Description

`out = daqfind` returns all device objects that exist in the data acquisition engine. The output `out` is an array.

`out = daqfind('`*PropertyName*`',PropertyValue,...)` returns all device objects, channels, or lines that exist in the data acquisition engine and have the specified property names and property values. The property name/property value pairs can be specified as a cell array.

`out = daqfind(S)` returns all device objects, channels, or lines that exist in the data acquisition and have the property names and property values specified by `S`. `S` is a structure with field names that are property names and field values that are property values.

`out = daqfind(obj,'PropertyName',PropertyValue,...)` returns all device objects, channels, or lines listed by `obj` that have the specified property names and property values.

Tips

More About Finding Device Objects, Channels, or Lines

`daqfind` is particularly useful in these circumstances:

- A device object is cleared from the MATLAB workspace, and it needs to be retrieved from the data acquisition engine.
- You need to locate device objects, channels, or lines that have particular property names and property values.

Rules for Specifying Property Names and Property Values

- You can use property name/property value string pairs, structures, and cell array pairs in the same call to `daqfind`. However, in a single call to `daqfind`, you can specify only device object properties or channel/line properties.
- You must use the same format as returned by `get`. For example, if `get` returns the `ChannelName` property value as `Left`, you must specify `Left` as the property value in `daqfind` (case matters). However, case does not matter when you specify enumerated property values. For example, `daqfind` will find a device object with a `Running` property value of `On` or `on`.

Examples

You can use `daqfind` to return a cleared device object.

```
ai = analoginput('winsound');
ch = addchannel(ai,1:2);
set(ch,{'ChannelName'},{'Joe';'Jack'})
clear ai
ainew = daqfind;
```

To return the channel associated with the descriptive name `Jack`:

```
ch2 = daqfind(ainew,'ChannelName','Jack');
```

To return the device object with a sampling rate of 8000 Hz and the descriptive name `winsound0-AI`, you can pass a structure to `daqfind`.

```
S.Name = 'winsound0-AI';
S.SampleRate = 8000;
daqobj = daqfind(S);
```

See Also

`clear` | `get` | `propinfo`

Purpose

Help for device objects, constructors, adaptors, functions, and properties

Syntax

```
daqhelp
out = daqhelp('name')
out = daqhelp(obj)
out = daqhelp(obj,'name')
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`'name'`	A device object, constructor, adaptor, function, or property name.
`obj`	A device object.
`out`	Contains the specified help text.

Description

`daqhelp` displays a complete listing of Data Acquisition Toolbox constructors and functions along with a brief description of each.

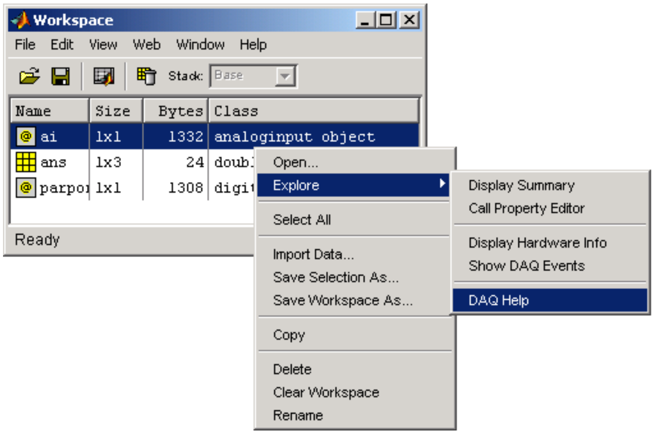
`out = daqhelp('name')` returns help for the device object, constructor, adaptor, function, or property specified by `name`. The help text is returned to `out`.

`out = daqhelp(obj)` returns a complete listing of functions and properties for the device object `obj` to `out`. Help for `obj`'s constructor is also displayed.

`out = daqhelp(obj,'name')` returns help for `name` for the specified device object `obj` to `out`. `name` can be a constructor, adaptor, property, or function name.

Tips

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



Access context (pop-up) menus by right-clicking a device object.

More About Displaying Help

- When displaying property help, the names in the "See Also" section that contain all uppercase letters are function names. The names that contain a mixture of upper- and lowercase letters are property names.
- When displaying function help, the "See Also" section contains only function names.

Rules for Specifying Names

For the `daqhelp('name')` syntax:

- If `name` is the name of a constructor, a complete listing of the device object's functions and properties is displayed along with a brief description of each function and property. The constructor help is also displayed.
- You can display object-specific function information by specifying `name` as object/function. For example, to display the help for an analog input object's `getdata` function, `name` is `analoginput/getdata`.

- You can display object-specific property information by specifying `name` as `obj.property`. For example, to display the help for an analog input object's `SampleRate` property, `name` is `analoginput.SampleRate`.

For the `daqhelp(obj,'name')` syntax:

- If `name` is the name of a device object constructor and the `.m` extension is included, the constructor help is displayed.
- If `name` is the name of a function or property, the function or property help is displayed.

Examples

The following commands are some of the ways you can use `daqhelp` to obtain help on device objects, constructors, adaptors, functions, and properties.

```
daqhelp('analogoutput');
out = daqhelp('analogoutput.m');
daqhelp set
daqhelp analoginput/peekdata
daqhelp analoginput.TriggerDelayUnits
```

The following commands are some of the ways you can use `daqhelp` to obtain information about functions and properties for an existing device object.

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

See Also

`propinfo`

daqhwinfo

Purpose

Data acquisition hardware information

Syntax

```
out = daqhwinfo
out = daqhwinfo('adaptor')
out = daqhwinfo(obj)
out = daqhwinfo(obj,'FieldName')
```

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

Arguments

`'adaptor'`	The hardware driver adaptor name. The supported adaptors are `advantech`, `mcc`, `nidaq`, `parallel`, and `winsound`.
`obj`	A device object or array of device objects.
`'FieldName'`	A single field name or a cell array of field names.
`out`	A structure containing the requested hardware information.

Description

`out = daqhwinfo` returns general hardware-related information as a structure to `out`. The returned information includes installed adaptors, the toolbox and the MATLAB software version, and the toolbox name.

`out = daqhwinfo('adaptor')` returns hardware-related information for the specified *`adaptor`*. The returned information includes the adaptor DLL name, the board names and IDs, and the device object constructor syntax.

Note If you are trying to discover National Instruments including CompactDAQ or Counter/Timer subsystem devices, use the `daq.getDevices` method.

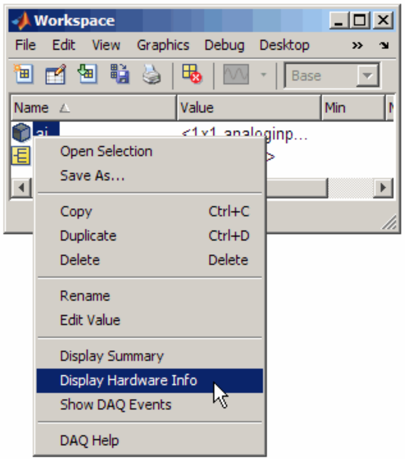
`out = daqhwinfo('`*`adaptor`*`','`*`FieldName`*`')` returns the hardware-related information specified by *`FieldName`* for *`adaptor`*. *`FieldName`* must be a single string. `out` is a cell array. You can return a list of valid field names with the `daqhwinfo('`*`adaptor`*`')` syntax.

`out = daqhwinfo(obj)` returns hardware-related information for the device object `obj`. If `obj` is an array of device objects, then `out` is a 1-by-n cell array of structures where n is the length of `obj`. The returned information depends on the device object type, and might include the maximum and minimum sampling rates, the channel gains, the hardware channel or line IDs, and the vendor driver version.

`out = daqhwinfo(obj,'`*`FieldName`*`')` returns the hardware-related information specified by *`FieldName`* for the device object `obj`. *`FieldName`* can be a single field name or a cell array of field names. `out` is an m-by-n cell array where m is the length of `obj` and n is the length of *`FieldName`*. You can return a list of valid field names with the `daqhwinfo(obj)` syntax.

Tips

As shown below, you can also return hardware information via the Workspace browser by right-clicking a device object, and selecting **Display Hardware Info** from the context menu.



Examples

Display all installed adaptors. Note that this list might be different for your platform.

```
out = daqhwinfo;
out.InstalledAdaptors

ans =
    'advantech'
    'mcc'
    'nidaq'
    'parallel'
    'winsound'
```

To display the device object constructor names for all installed `winsound` devices:

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

Create the analog input object `ai` for a sound card. To display the input ranges for `ai`:

```
ai = analoginput('winsound');
out = daqhwinfo(ai);
out.InputRanges
ans =
    -1     1
```

To display the minimum and maximum sampling rates for `ai`:

```
out = daqhwinfo(ai,{'MinSampleRate','MaxSampleRate'})
out =
    [8000]    [44100]
```

Notes 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.

The Parallel adaptor will be deprecated in a future version of the toolbox. If you create a Data Acquisition Toolbox™ object for `'parallel'` beginning in R2008b, you will receive a warning stating that this adaptor will be removed in a future release. See the supported hardware page at `www.mathworks.com/products/daq/supportedio.html` for more information.

---

daqmem

Purpose

Allocate or display analog input and output memory resources

Syntax

```
out = daqmem
out = daqmem(obj)
daqmem(obj,maxmem)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	A device object or array of device objects.
`maxmem`	The amount of memory to allocate.
`out`	A structure containing information about memory resources.

Description

`out = daqmem` returns the object `out`, which contains several properties describing the memory resources associated with your platform and Data Acquisition Toolbox. The fields are described below.

Field	Description
`MemoryLoad`	Specifies a number between 0 and 100 that gives a general idea of current memory utilization. 0 indicates no memory use and 100 indicates full memory use.
`TotalPhys`	Indicates the total number of bytes of physical memory.
`AvailPhys`	Indicates the number of bytes of physical memory available.

Field	Description
TotalPageFile	Indicates the total number of bytes that can be stored in the paging file. Note that this number does not represent the actual physical size of the paging file on disk.
AvailPageFile	Indicates the number of bytes available in the paging file.
TotalVirtual	Indicates the total number of bytes that can be described in the user mode portion of the virtual address space of the calling process.
AvailVirtual	Indicates the number of bytes of unreserved and uncommitted memory in the user mode portion of the virtual address space of the calling process.
UsedDaq	The total memory used by all device objects.

Note that all the above fields, except for `UsedDaq`, are identical to the fields returned by Windows' `MemoryStatus` function.

`out = daqmem(obj)` returns a 1-by-N structure `out` containing two fields: `UsedBytes` and `MaxBytes` for the device object `obj`. N is the number of device objects specified by `obj`. `UsedBytes` returns the number of bytes used by `obj`. `MaxBytes` returns the maximum number of bytes that can be used by `obj`.

`daqmem(obj,maxmem)` sets the maximum memory that can be allocated for `obj` to the value specified by `maxmem`.

Tips

More About Allocating and Displaying Memory Resources

- For analog output objects, `daqmem(obj,maxmem)` controls the value of the `MaxSamplesQueued` property.
- If you manually configure the `BufferingConfig` property, then this value supersedes the values specified by `daqmem(obj,maxmem)` and the `MaxSamplesQueued` property.

Examples

Create the analog input object aiwin for a sound card and the analog input object aini for a National Instruments board, and add two channels to each device object.

```
aiwin = analoginput('winsound');
addchannel(aiwin,1:2);
aini = analoginput('nidaq','Dev1');
addchannel(aini,0:1);
```

To display the total memory used by all existing device objects:

```
out = daqmem;
out.UsedDaq
ans =
       69120
```

To configure the maximum memory used by aiwin to 640 KB:

```
daqmem(aiwin,640000)
```

To configure the maximum memory used by each device object with one call to daqmem:

```
daqmem([aiwin aini],[640000 480000])
```

See Also

BufferingConfig | MaxSamplesQueued

Purpose

Read Data Acquisition Toolbox (`.daq`) file for analog input

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

See Also

`daqread`

daqregister

Purpose

Register or unregister hardware driver adaptor

Syntax

```
daqregister('adaptor')
daqregister('adaptor','unload')
out = daqregister(...)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

'*adaptor*'	The hardware driver adaptor name. The supported adaptors are `advantech`, `mcc`, `nidaq`, `parallel`, and `winsound`.
'**unload**'	Specifies that the hardware driver adaptor is to be unloaded.
`out`	Captures the message returned by `daqregister`.

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.

The Parallel adaptor will be deprecated in a future version of the toolbox. If you create a Data Acquisition Toolbox™ object for `'parallel'` beginning in R2008b, you will receive a warning stating that this adaptor will be removed in a future release. See the supported hardware page at `www.mathworks.com/products/daq/supportedio.html` for more information.

Description

`daqregister('`*`adaptor`*`')` registers the hardware driver adaptor specified by *`adaptor`*.

Notes You must have administrative privileges to register or unregister hardware driver adaptors.

If you are using a Windows Vista™ machine, you must log on with Administrative privileges and run MATLAB. You should then execute `daqregister` with elevated permissions. This will allow the User Account Control feature on your computer to run correctly.

For third-party adaptors, *`adaptor`* must include the full pathname.

`daqregister('`*`adaptor`*`','`**`unload`**`')` unregisters the hardware driver adaptor specified by *`adaptor`*. For third-party adaptors, *`adaptor`* must include the full pathname.

`out = daqregister(...)` captures the resulting message in `out`.

Tips

A hardware driver adaptor must be registered so the data acquisition engine can make use of its services. Unless an adaptor is unloaded, registration is required only once.

For adaptors that are included with the toolbox, registration occurs automatically when you first create a device object. However, you might need to register third-party adaptors manually. In either case, you must install the associated hardware driver before registration can occur.

Examples

The following command registers the sound card adaptor provided with the toolbox.

```
daqregister('winsound');
```

The following command registers the third-party adaptor `myadaptor.dll`. Note that you must supply the full pathname to `daqregister`.

```
daqregister('D:/MATLABR12/toolbox/daq/myadaptors/
myadaptor.dll');
```

daqreset

Purpose

Remove device objects, engine MEX-file, and adaptor DLLs from memory

Syntax

```
daqreset
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Description

`daqreset` removes all device objects existing in the engine, and unloads all data acquisition executables loaded by the engine (including the adaptor DLLs and the engine MEX-file).

You should use `daqreset` to return the MATLAB workspace to a known initial state of having no device objects and no data acquisition MEX-file or DLLs loaded in memory. When the MATLAB workspace returns to this state, the data acquisition hardware is reset.

Note `daqreset` only affects Data Acquisition Toolbox engine and its adaptors. It does not affect the hardware. To reset the hardware you must use the tools supplied by the hardware vendor. Refer to your hardware documentation for details.

See Also

`clear` | `delete`

Purpose

Convert digital input and output decimal value to binary vector

Syntax

```
out = dec2binvec(dec)
out = dec2binvec(dec,bits)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`dec`	A decimal value. `dec` must be nonnegative.
`bits`	Number of bits used to represent the decimal number.
`out`	A logical array containing the binary vector.

Description

`out = dec2binvec(dec)` converts the decimal value `dec` to an equivalent binary vector and stores the result as a logical array in `out`.

`out = dec2binvec(dec,bits)` converts the decimal value `dec` to an equivalent binary vector consisting of at least the number of bits specified by `bits`.

Tips

More About Binary Vectors

A binary vector (binvec) is constructed with the least significant bit (LSB) in the first column and the most significant bit (MSB) in the last column. For example, the decimal number 23 is written as the binvec value [1 1 1 0 1].

More About Specifying the Number of Bits

- If `bits` is greater than the minimum number of bits required to represent the decimal value, then the result is padded with zeros.
- If `bits` is less than the minimum number of bits required to represent the decimal value, then the minimum number of required bits is used.

- If `bits` is not specified, then the minimum number of bits required to represent the number is used.

Examples

To convert the decimal value 23 to a binvec value:

```
dec2binvec(23)
ans =
     1     1     1     0     1
```

To convert the decimal value 23 to a binvec value using six bits:

```
dec2binvec(23,6)
ans =
     1     1     1     0     1     0
```

To convert the decimal value 23 to a binvec value using four bits, then the result uses five bits. This is the minimum number of bits required to represent the number.

```
dec2binvec(23,4)
ans =
     1     1     1     0     1
```

See Also

`binvec2dec`

Purpose

Remove device objects, channels, or lines from data acquisition engine

Syntax

```
delete(obj)
delete(obj.Channel(index))
delete(obj.Line(index))
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	A device object or array of device objects.
`obj.Channel(index)`	One or more channels contained by `obj`.
`obj.Line(index)`	One or more lines contained by `obj`.

Description

`delete(obj)` removes the device object specified by `obj` from the engine. If `obj` contains channels or lines, they are removed as well. If `obj` is the last object accessing the driver, then the driver and associated adaptor are unloaded.

`delete(obj.Channel(index))` removes the channels specified by `index` and contained by `obj` from the engine. As a result, the remaining channels might be reindexed.

`delete(obj.Line(index))` removes the lines specified by `index` and contained by `obj` from the engine. As a result, the remaining lines might be reindexed.

Tips

Deleting device objects, channels, and lines follows these rules:

- `delete` removes device objects, channels, or lines from the data acquisition engine but not from the MATLAB workspace. To remove variables from the workspace, use the `clear` function.
- If multiple references to a device object exist in the workspace, then removing one device object from the engine invalidates the remaining

references. These remaining references should be cleared from the workspace with the `clear` function.

- If you delete a device object while it is running, then a warning is issued before it is deleted. You cannot delete a device object while it is logging or sending data.

You should use `delete` at the end of a data acquisition session. You can quickly delete all existing device objects with the command `delete(daqfind)`.

If you use the `help` command to display the file help for `delete`, then you must supply the pathname shown below.

```
help daq/daqdevice/delete
```

Examples

National Instruments

Create the analog input object `ai` for a National Instruments board, add hardware channels 0-7 to it, and make a copy of hardware channels 0 and 1.

```
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:7);
ch = ai.Channel(1:2);
```

To delete hardware channels 0 and 1:

```
delete(ch)
```

These channels are deleted from the data acquisition engine and are no longer associated with `ai`. The remaining channels are reindexed such that the indices begin at 1 and increase monotonically to 6. To delete `ai`:

```
delete(ai)
```

Sound Card

Create the analog input object `AI1` for a sound card, and configure it to operate in stereo mode.

```
AI1 = analoginput('winsound');
addchannel(AI1,1:2);
```

You can now configure the sound card for mono mode by deleting hardware channel 2.

```
delete(AI1.Channel(2))
```

If hardware channel 1 is deleted instead, an error is returned.

See Also

`clear` | `daqreset`

Purpose

Create digital I/O object

Syntax

```
DIO = digitalio('adaptor',ID)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

'adaptor'	The hardware driver adaptor name. The supported adaptors are `advantech`, `mcc`, `nidaq`, and `parallel`.
`ID`	The hardware device identifier.
`DIO`	The digital I/O object.

Description

`DIO = digitalio('`*`adaptor`*`',ID)` creates the digital I/O object `DIO` for the specified *`adaptor`* and for the hardware device with device identifier `ID`. `ID` can be specified as an integer or a string.

Notes 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.

The Parallel adaptor will be deprecated in a future version of the toolbox. If you create a Data Acquisition Toolbox™ object for `'parallel'` beginning in R2008b, you will receive a warning stating that this adaptor will be removed in a future release. See the supported hardware page at `www.mathworks.com/products/daq/supportedio.html` for more information.

Tips

More About Creating Digital I/O Objects

- When a digital I/O object is created, it does not contain any hardware lines. To execute the device object, hardware lines must be added with the `addline` function.
- You can create multiple digital I/O objects that are associated with a particular digital I/O subsystem. However, you can execute only one of these digital I/O objects at a time for the generation of timing events.
- The digital I/O object exists in the data acquisition engine and in the MATLAB workspace. If you create a copy of the device object, it references the original device object in the engine.
- The `Name` property is automatically assigned a descriptive name that is produced by concatenating *`adaptor`*, `ID`, and `-DIO`. You can change this name at any time.

Note When you create a digital input or output object, it consumes system resources. To avoid this issue, make sure that you do not create objects in a loop. If you must create objects in a loop, make sure you delete them within the loop.

The Parallel Port Adaptor

The toolbox provides basic DIO capabilities through the parallel port. The PC supports up to three parallel ports that are assigned the labels LPT1, LPT2, and LPT3. You can use only these ports. If you add additional ports to your system, or if the standard ports do not use the default memory resources, they will not be accessible by the toolbox. For more information about the parallel port, refer to "Parallel Port Characteristics".

More About the Hardware Device Identifier

When data acquisition devices are installed, they are assigned a unique number, which identifies the device in software. The device identifier is typically assigned automatically and can usually be manually changed using a vendor-supplied device configuration utility. National Instruments refers to this number as the device number.

There are two ways you can determine the `ID` for a particular device:

- Type `daqhwinfo('adaptor')`.
- Open the vendor-supplied device configuration utility.

Properties

Common Properties

`Line`	Contain hardware lines added to device object
`Name`	Specify descriptive name for the channel
`Running`	Indicate whether device object is running
`Tag`	Specify device object label
`TimerFcn`	Specify callback function to execute when predefined time period passes
`TimerPeriod`	Specify time period between timer events
`Type`	Indicate device object type, channel, or line
`UserData`	Store data to associate with device object

Line Properties

`Direction`	Specify whether line is for input or output
`HwLine`	Specify hardware line ID

`Index`	MATLAB index of hardware channel or line
`LineName`	Specify descriptive line name
`Parent`	Indicate parent (device object) of channel or line
`Port`	Specify port ID
`Type`	Indicate device object type, channel, or line

Examples

Create a digital I/O object for a National Instruments device defined as `'Dev1'`.

```
DIO = digitalio('nidaq','Dev1');
```

Create a digital I/O object for a Measurement Computing device defined as `'1'`.

```
DIO = digitalio('mcc','1');
```

Create a digital I/O object for parallel port LPT1.

```
DIO = digitalio('parallel','LPT1');
```

See Also

`addline` | `daqhwinfo` | `Name`

Purpose

Summary information for device objects, channels, or lines

Syntax

```
disp(obj)
disp(obj.Channel(index))
disp(obj.Line(index))
```

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

Arguments

`obj`	A device object.
`obj.Channel(index)`	One or more channels contained by `obj`.
`obj.Line(index)`	One or more lines contained by `obj`.

Description

`disp(obj)` displays summary information for the specified device object `obj`, and any channels or lines contained by `obj`. Typing `obj` at the Command Window produces the same summary information.

`disp(obj.Channel(index))` displays summary information for the specified channels contained by `obj`. Typing `obj.Channel(index)` at the Command Window produces the same summary information.

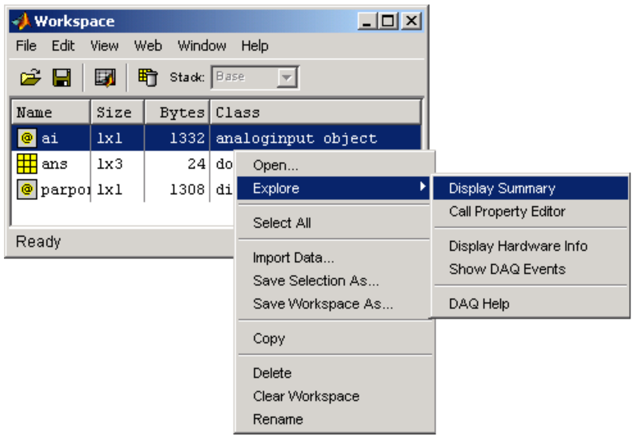
`disp(obj.Line(index))` displays summary information for the specified lines contained by `obj`. Typing `obj.Line(index)` at the Command Window produces the same summary information.

Tips

You can invoke `disp` by typing the device object at the MATLAB Command Window or by excluding the semicolon when

- Creating a device object
- Adding channel or lines
- Configuring property values using the dot notation

As shown below, you can also display summary information via the Workspace browser by right-clicking a device object, a channel object, or a line object and selecting **Explore > Display Summary** from the context menu.



Access context (pop-up) menus by right-clicking a device object.

Examples

All the commands shown below produce summary information for the device object `AI` or the channels contained by `AI`.

```
AI = analoginput('winsound')
chans = addchannel(AI,1:2)
AI.SampleRate = 44100
AI.Channel(1).ChannelName = 'CH1'
chans
```

flushdata

Purpose

Remove analog input data from data acquisition engine

Syntax

```
flushdata(obj)
flushdata(obj,'mode')
```

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

Arguments

`obj`	An analog input object or array of analog input objects.
`'mode'`	Specifies how much data is removed from the engine.

Description

`flushdata(obj)` removes all data from the data acquisition engine and resets the `SamplesAvailable` property to zero.

`flushdata(obj,'mode')` removes data from the data acquisition engine depending on the value of *`mode`*:

- If *`mode`* is `all`, then `flushdata` removes all data from the engine and the `SamplesAvailable` property is set to `0`. This is the same as `flushdata(obj)`.
- If *`mode`* is `triggers`, then `flushdata` removes the data acquired during one trigger. `triggers` is a valid choice only when the `TriggerRepeat` property is greater than 0 and the `SamplesPerTrigger` property is not `inf`. The data associated with the oldest trigger is removed first.

Examples

Create the analog input object `ai` for a National Instruments board and add hardware channels 0-7 to it.

```
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:7);
```

A 2-second acquisition is configured and the device object is executed.

```
set(ai,'SampleRate',2000)
duration = 2;
ActualRate = get(ai,'SampleRate');
set(ai,'SamplesPerTrigger',ActualRate*duration)
start(ai)
wait(ai,duration+1)
```

Four thousand samples will be acquired for each channel group member. To extract 1000 samples from the data acquisition engine for each channel:

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

You can use `flushdata` to remove the remaining 3000 samples from the data acquisition engine.

```
flushdata(ai)
ai.SamplesAvailable
ans =
     0
```

See Also

`getdata` | `SamplesAvailable` | `SamplesPerTrigger` | `TriggerRepeat`

Purpose

Device object properties

Syntax

```
out = get(obj)
out = get(obj.Channel(index))
out = get(obj.Line(index))
out = get(obj,'PropertyName')
out = get(obj.Channel(index),'PropertyName')
out = get(obj.Line(index),'PropertyName')
get(...)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	A device object or array of device objects.
`obj.Channel(index)`	One or more channels contained by `obj`.
`obj.Line(index)`	One or more lines contained by `obj`.
`'`*`PropertyName`*`'`	A property name or a cell array of property names.

Description

`out = get(obj)` returns the structure `out`, where each field name is the name of a property of `obj` and each field contains the value of that property.

`out = get(obj.Channel(index))` returns the structure `out`, where each field name is the name of a channel property of `obj` and each field contains the value of that property.

`out = get(obj.Line(index))` returns the structure `out`, where each field name is the name of a line property of `obj` and each field contains the value of that property.

`out = get(obj,'`*`PropertyName`*`')` returns the value of the property specified by *`PropertyName`* to `out`. If *`PropertyName`* is replaced by a

1-by-n or n-by-1 cell array of strings containing property names, then `get` returns a 1-by-n cell array of values to `out`. If `obj` is an array of data acquisition objects, then `out` will be an m-by-n cell array of property values where m is equal to the length of `obj` and n is equal to the number of properties specified.

`out = get(obj.Channel(index),'`*`PropertyName`*`')` returns the value of *`PropertyName`* to `out` for the specified channels contained by `obj`. If multiple channels and multiple property names are specified, then `out` is an m-by-n cell array where m is the number of channels and n is the number of properties.

`out = get(obj.Line(index),'`*`PropertyName`*`')` returns the value of *`PropertyName`* to `out` for the specified lines contained by `obj`. If multiple lines and multiple property names are specified, then `out` is an m-by-n cell array where m is the number of lines and n is the number of properties.

`get(...)` displays all property names and their current values for the specified device object, channel, or line. Base properties are displayed first followed by device-specific properties.

Tips

If you use the `help` command to display the file help for `get`, then you must supply the pathname shown below.

```
help daq/daqdevice/get
```

Examples

Create the analog input object `ai` for a sound card and configure it to operate in stereo mode.

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

The commands shown below are some of the ways you can use `get` to return property values.

```
chan = get(ai,'Channel');
out = get(ai,{'SampleRate','TriggerDelayUnits'});
out = get(ai);
```

```
get(chan(1),'Units')
get(chan,{'Index','HwChannel','ChannelName'})
```

See Also

`set` | `setverify`

Purpose

Extract analog input data, time, and event information from data acquisition engine

Syntax

```
data = getdata(obj)
data = getdata(obj,samples)
data = getdata(obj,samples,'type')
[data,time] = getdata(...)
[data,time,abstime] = getdata(...)
[data,time,abstime,events] = getdata(...)
[data,...] = getdata(obj, 'P1', V1, 'P2', V2,...)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

Arguments

`obj`	An analog input object.
`samples`	The number of samples to extract. If `samples` is not specified, the number of samples extracted is given by the `SamplesPerTrigger` property.
`'type'`	Specifies the format of the extracted data as `double` (the default) or as `native`.
`data`	An m-by-n array, where m is the number of samples extracted and n is the number of channels contained by `obj`.
`time`	An m-by-1 array of relative time values in seconds, where m is the number of samples extracted. `time = 0` is defined as the point at which data logging begins, i.e., when the `Logging` property of `obj` is set to `On`. Measurement of `time`, with respect to `0`, continues until the acquisition is stopped, i.e., when the `Logging` property of `obj` is set to `Off`.

`abstime`	The absolute time of the first trigger returned as a `clock` vector. This value is identical to the value stored by the `InitialTriggerTime` property.
`events`	A structure containing a list of events that occurred during the time period the samples were extracted.

Description

`data = getdata(obj)` extracts the number of samples specified by the `SamplesPerTrigger` property for each channel contained by `obj`. `data` is an m-by-n array, where m is the number of samples extracted and n is the number of channels.

`data = getdata(obj,samples)` extracts the number of samples specified by `samples` for each channel contained by `obj`.

`data = getdata(obj,samples,'type')` extracts the number of samples specified by `samples` in the format specified by *`type`* for each channel contained by `obj`.

`[data,time] = getdata(...)` returns data as sample-time pairs. `time` is an m-by-1 array of relative time values, where m is the number of samples returned in `data`. Each element of `time` indicates the relative time, in seconds, of the corresponding sample in `data`, measured with respect to the first sample logged by the engine.

`[data,time,abstime] = getdata(...)` extracts data as sample-time pairs and returns the absolute time of the trigger. The absolute time is returned as a `clock` vector and is identical to the value stored by the `InitialTriggerTime` property.

`[data,time,abstime,events] = getdata(...)` extracts data as sample-time pairs, returns the absolute time of the trigger, and returns a structure containing a list of events that occurred during the time period the samples were extracted. The events returned are a sub set of those stored by the `EventLog` property.

`[data,...] = getdata(obj, 'P1', V1, 'P2', V2,...)` specifies the number of samples to be returned, the format of the data matrix, and whether to return a `tscollection` object.

The following table shows a summary of properties.

Property	Description
`Samples`	Specify the number of samples to return.
`DataFormat`	Specify the data format as `double` (default) or `native`.
`OutputFormat`	Specify the output format as `matrix` (default) or `tscollection`.

Note When the `ClockSource` property for this function is set to one of the `External` options, the timing will be controlled externally and the values returned in the `time` variable will not accurately reflect the actual relative time of each sample. It is however an approximation based on the `SampleRate` you have configured.

Tips

More About getdata

- In most circumstances, `getdata` returns all requested data and does not miss any samples. In the unlikely event that the engine cannot keep pace with the hardware device, it is possible that data is missed. If data is missed, the `DataMissedFcn` property is called and the device object is stopped.
- `getdata` is a *blocking* function because it returns execution control to the MATLAB workspace only when the requested number of samples is extracted from the engine for each channel group member.
- You can issue ^C (**Ctrl+C**) while `getdata` is blocking. This will not stop the acquisition but will return control to the MATLAB software.
- The amount of data that you can extract from the engine is given by the `SamplesAvailable` property.
- It is a good practice to use a `wait` command before your `getdata` command if the `getdata` is going to get all data returned by the

analog input subsystem. For example, if your analog input object is `ai` and you have set `duration` to be the number of seconds for the acquisition, you could add the following line right before the `getdata`:

```
wait(ai,duration+1)
```

- Setting the `OutputFormat` property to `tscollection` causes `getdata` to return a `tscollection` object. In this case, only the `data` left-hand argument is used.
- For more information on using the Time Series functionality, see “Example: Time Series Objects and Methods” in the MATLAB documentation.

More About Extracting Data From the Engine

- After the requested data is extracted from the engine, the `SamplesAvailable` property value is automatically reduced by the number of samples returned.
- If the requested number of samples is greater than the samples to be acquired, then an error is returned.
- If the requested data is not returned in the expected amount of time, an error is returned. The expected time to return data is given by the time it takes the engine to fill one data block plus the time specified by the `Timeout` property.
- If multiple triggers are included in a single `getdata` call, a `NaN` is inserted into the returned data and time arrays and the absolute time returned is given by the first trigger.
- When you use multiple immediate triggers Data Acquisition Toolbox cannot determine the “dead” time between triggers. Because of this, the toolbox assumes the “dead” time = 1 sample. For example if the sample rate is 1000 samples per second the toolbox assumes the “dead” time between triggers is one millisecond. The `time` argument returned by `getdata` reflects this assumption.

Examples

Create the analog input object `ai` for a National Instruments board and add hardware channels 0 to 3 to it.

```
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:3);
```

Configure a 1-second acquisition with `SampleRate` set to 1000 samples per second and `SamplesPerTrigger` set to 1000 samples per trigger.

```
set(ai,'SampleRate',1000)
set(ai,'SamplesPerTrigger',1000)
start(ai)
```

The following `getdata` command blocks execution control until all sample-time pairs, the absolute time of the trigger, and any events that occurred during the `getdata` call are returned.

```
wait(ai,1)
[data,time,abstime,events] = getdata(ai);
```

`data` is returned as a 1000-by-4 array of doubles, `time` is returned as a 1000-by-1 vector of relative times, `abstime` is returned as a `clock` vector, and `events` is returned as a 3-by-1 structure array.

To extract the 1000 data samples from hardware channel 0 only, examine the first column of `data`.

```
chan0_data = data(:,1);
```

The three events returned are the start event, the trigger event, and the stop event. To return specific event information about the stop event, you must access the `Type` and `Data` fields.

```
EventType = events(3).Type;
EventData = events(3).Data;
```

See Also

`flushdata` | `getsample` | `peekdata` | `timeseries` | `tscollection` | `wait` | `DataMissedFcn` | `EventLog` | `SamplesAvailable` | `SamplesPerTrigger` | `Timeout`

getsample

Purpose

Immediately acquire one analog input sample

Syntax

```
sample = getsample(obj)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

Arguments

`obj`	An analog input object.
`sample`	A row vector containing one sample for each channel contained by `obj`.

Description

`sample = getsample(obj)` immediately returns a row vector containing one sample for each channel contained by `obj`.

Tips

Using `getsample` is a good way to test your analog input configuration. Additionally:

- `getsample` does not store samples in, or extract samples from, the data acquisition engine.
- You can execute `getsample` at any time after channels have been added to `obj`.
- `getsample` is not supported for sound cards and Dynamic Signal Acquisition and Generation (DSA) cards.

Note Refer to the “Hardware Limitations by Vendor” section before you access National Instruments devices with the NI-DAQmx adaptor simultaneously from multiple applications.

Examples

Create the analog input object ai and add eight channels to it.

```
ai = analoginput('nidaq','Dev1');
ch = addchannel(ai,0:7);
```

The following command returns one sample for each channel.

```
sample = getsample(ai);
```

See Also

getdata | peekdata

getvalue

Purpose Read values from digital input and output lines

Syntax

```
out = getvalue(obj)
out = getvalue(obj.Line(index))
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	A digital I/O object.
`obj.Line(index)`	One or more lines contained by `obj`.
`out`	A binary vector.

Description

`out = getvalue(obj)` returns the current value from all lines contained by `obj` as a binary vector to `out`.

`out = getvalue(obj.Line(index))` returns the current value from the lines specified by `obj.Line(index)`.

Tips

More About Reading Values from Lines

- By default, `out` is returned as a binary vector (binvec). A binvec value is constructed with the least significant bit (LSB) in the first column and the most significant bit (MSB) in the last column. For example, the decimal number 23 is written as the binvec value [1 1 1 0 1].
- You can convert a binvec value to a decimal value with the `binvec2dec` function.
- If `obj` contains lines from a port-configurable device, the data acquisition engine will automatically read from all the lines even if they are not contained by the device object.
- When `obj` contains lines configured for output, `getvalue` returns the most recently output value set by `putvalue`. If you have not called

putvalue since you created the digitalio object, then getvalue returns a 0. getvalue cannot ascertain the current output value on the hardware.

Note Refer to the "Hardware Limitations by Vendor" section before you access National Instruments devices with the NI-DAQmx adaptor simultaneously from multiple applications.

Examples

Create the digital I/O object dio and add eight input lines to it.

```
dio = digitalio('nidaq','Dev1');
lines = addline(dio,0:7,'in');
```

To return the current values from all lines contained by dio as a binvec value:

```
out = getvalue(dio);
```

See Also

binvec2dec

Purpose

Open Property Inspector

Syntax

```
inspect(obj)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	An object or an array of objects.

Description

`inspect(obj)` opens the Property Inspector and allows you to inspect and set properties for the object `obj`.

Tips

You can also open the Property Inspector via the Workspace browser by double-clicking an object in the Workspace list.

The Property Inspector does not automatically update its display. To refresh the Property Inspector, open it again.

Examples

Create the analog input object `ai` for a sound card and add two channels.

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

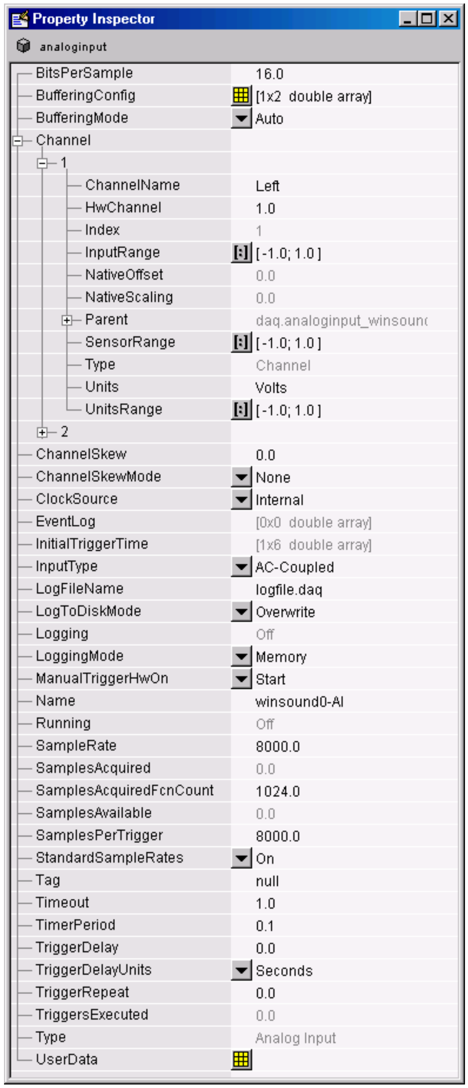
Open the Property Inspector for the object `ai`.

```
inspect(ai)
```

The Property Inspector is shown below.

You can expand the properties that are arrays of objects. In the following figure, the `Channel` property is expanded to enumerate the individual channel objects that make up this property.

You can also expand these individual channel objects to display their own properties, as shown for channel 1.



See Also

`daqfind` | `daqhelp` | `get` | `propinfo` | `set`

ischannel

Purpose

Check for channels

Syntax

```
out = ischannel(obj.Channel(index))
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj.Channel(index)`	One or more channels contained by `obj`.
`out`	A logical value.

Description

`out = ischannel(obj.Channel(index))` returns a logical 1 to `out` if `obj.Channel(index)` is a channel. Otherwise, a logical 0 is returned.

Tips

`ischannel` does not determine if channels are valid (associated with hardware). To check for valid channels, use the `isvalid` function.

Typically, you use `ischannel` directly only when you are creating your own files.

Examples

Suppose you create the function `myfunc` for use with Data Acquisition Toolbox software. If `myfunc` is passed one or more channels as an input argument, then the first thing you should do in the function is check if the argument is a channel.

```
function myfunc(chan)
% Determine if a channel was passed.
if ~ischannel(chan)
   error('The argument passed is not a channel.');
end
```

You can examine Data Acquisition Toolbox software files for examples that use `ischannel`.

See Also

isvalid

isdioline

Purpose

Check for lines

Syntax

```
out = isdioline(obj.Line(index))
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj.Line(index)`	One or more lines contained by `obj`.
`out`	A logical value.

Description

`out = isdioline(obj.Line(index))` returns a logical 1 to `out` if `obj.Line(index)` is a line. Otherwise, a logical 0 is returned.

Tips

`isdioline` does not determine if lines are valid (associated with hardware). To check for valid lines, use the `isvalid` function.

Typically, you use `isdioline` directly only when you are creating your own files.

Examples

Suppose you create the function `myfunc` for use with Data Acquisition Toolbox software. If `myfunc` is passed one or more lines as an input argument, then the first thing you should do in the function is check if the argument is a line.

```
function myfunc(line)
% Determine if a line was passed.
if ~isdioline(line)
   error('The argument passed is not a line.');
end
```

You can examine Data Acquisition Toolbox software files for examples that use `isdioline`.

See Also `isvalid`

islogging

Purpose

Determine whether analog input object is logging data

Syntax

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

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Description

`bool = islogging(obj)` returns `true` if the analog input object `obj` is logging data, otherwise `false`. An analog input object is logging if the value of its `Logging` property is set to `On`.

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

Examples

Create an analog input object and add a channel.

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

To put the analog input object in a logging state, start acquiring data. The example acquires 10 seconds of data to increase the amount of time that the object remains in the logging state.

```
set(ai,'SamplesPerTrigger', 10*get(ai,'SampleRate'))
start(ai)
```

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

```
bool = islogging(ai)
bool =
   1
```

Create a second analog input object.

```
ai2 = analoginput('winsound');
```

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

```
start(ai)
bool = islogging([ai ai2])
bool =
     1     0
```

See Also

`isrunning` | `issending` | `start` | `stop` | `Logging` | `LoggingMode`

isrunning

Purpose

Determine whether device object is running

Syntax

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

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Description

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

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

Examples

Create an analog input object and add a channel.

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

To put the analog input object in a running state, configure a manual trigger and then start the object.

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

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

```
bool = isrunning(ai)
bool =
   1
```

Create an analog output object.

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

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

```
bool = isrunning([ai ao])
bool =
     1     0
```

See Also

`islogging` | `issending` | `start` | `stop` | `Running`

Purpose

Determine whether analog output object is sending data

Syntax

```
bool = issending(obj)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Description

`bool = issending(obj)` returns `true` if the analog output object `obj` is sending data to the hardware device, otherwise `false`. An analog output object is sending if the value of its `Sending` property is set to `On`.

If `obj` is an array of analog output objects, `bool` is a logical array where each element in `bool` represents the corresponding element in `obj`. If an object in `obj` is sending, the `issending` function sets the corresponding element in `bool` to `true`, otherwise `false.` If any of the analog output objects in `obj` is invalid, `issending` returns an error.

Examples

Create an analog output object and add a channel.

```
ao = analogoutput('winsound');
addchannel(ao, 1);
```

To put the analog output object in a sending state, start acquiring data. The example sends 10 seconds of data to increase the amount of time that the object remains in the sending state.

```
putdata(ao, ones(10*get(ao,'SampleRate'),1));
start(ao)
```

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

```
bool = issending(ao)
bool =
   1
```

Create a second analog output object.

```
ao2 = analogoutput('winsound');
```

Start one of the analog output objects again, such as `ao`, and use `issending` to determine which of the two objects is sending.

```
putdata(ao, ones(10*get(ao,'SampleRate'),1));
start(ao)
bool = issending([ao ao2])
bool =
     1     0
```

See Also

`islogging` | `isrunning` | `start` | `stop` | `Sending`

isvalid

Purpose

Determine whether device objects, channels, or lines are valid

Syntax

```
out = isvalid(obj)
out = isvalid(obj.Channel(index))
out = isvalid(obj.Line(index))
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	A device object or array of device objects.
`obj.Channel(index)`	One or more channels contained by `obj`.
`obj.Line(index)`	One or more lines contained by `obj`.
`out`	A logical array.

Description

`out = isvalid(obj)` returns a logical 1 to `out` if `obj` is a valid device object. Otherwise, a logical 0 is returned.

`out = isvalid(obj.Channel(index))` returns a logical 1 to `out` if the channels specified by `obj.Channel(index)` are valid. Otherwise, a logical 0 is returned.

`out = isvalid(obj.Line(index))` returns a logical 1 to `out` if the lines specified by `obj.Line(index)` are valid. Otherwise, a logical 0 is returned.

Tips

Invalid device objects, channels, and lines are no longer associated with any hardware and should be cleared from the workspace with the `clear` function.

Typically, you use `isvalid` directly only when you are creating your own files.

Examples

Create the analog input object `ai` for a National Instruments board and add eight channels to it.

```
ai = analoginput('nidaq','Dev1');
ch = addchannel(ai,0:7);
```

To verify the device object is valid:

```
isvalid(ai)
ans =
     1
```

To verify the channels are valid:

```
isvalid(ch)'
ans =
     1     1     1     1     1     1     1     1
```

If you delete a channel, then `isvalid` returns a logical 0 in the appropriate location:

```
delete(ai.Channel(3))
isvalid(ch)'
ans =
     1     1     0     1     1     1     1     1
```

Typically, you use `isvalid` directly only when you are creating your own files. Suppose you create the function `myfunc` for use with Data Acquisition Toolbox software. If `myfunc` is passed the previously defined device object `ai` as an input argument,

```
myfunc(ai)
```

the first thing you should do in the function is check if `ai` is a valid device object.

```
function myfunc(obj)
% Determine if an invalid handle was passed.
if ~isvalid(obj)
```

```
   error('Invalid data acquisition object passed.');
end
```

You can examine Data Acquisition Toolbox software files for examples that use `isvalid`.

See Also

`clear` | `delete` | `ischannel` | `isdioline`

Purpose

Length of device object, channel group, or line group

Syntax

```
out = length(obj)
out = length(obj.Channel)
out = length(obj.Line)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

Arguments

`obj`	A device object or array of device objects.
`obj.Channel`	The channels contained by `obj`.
`obj.Line`	The lines contained by `obj`.
`out`	A double.

Description

`out = length(obj)` returns the length of the device object `obj` to `out`.

`out = length(obj.Channel)` returns the length of the channel group contained by `obj`.

`out = length(obj.Line)` returns the length of the line group contained by `obj`.

Examples

Create the analog input object `ai` for a National Instruments board and add eight channels to it.

```
ai = analoginput('nidaq','Dev1');
aich = addchannel(ai,0:7);
```

Create the analog output object `ao` for a National Instruments board, add one channel to it, and create the device object array `aiao`.

```
ao = analogoutput('nidaq','Dev1');
aoch = addchannel(ao,0);
```

```
aiao = [ai ao]

Index:    Subsystem:         Name:
   1         Analog Input       nidaqmxDev1-AI
   2         Analog Output      nidaqmxDev1-AO
```

To find the length of `aiao`:

```
length(aiao)
ans =
     2
```

To find the length of the analog input channel group:

```
length(aich)
ans =
     8
```

See Also

`size`

Purpose

Load device objects, channels, or lines into MATLAB workspace

Syntax

```
load file
load file obj1 obj2. . .
out = load('file','obj1','obj2',. . .)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`file`	The MAT-file name.
`obj1` `obj2...`	Device objects, an array of device objects, channels, or lines.
`out`	A structure containing the loaded device objects.

Description

`load file` returns all variables from the MAT-file `file` into the MATLAB workspace.

`load file obj1 obj2...` returns the specified device objects from the MAT-file `file` into the MATLAB workspace.

`out = load('file','obj1','obj2',...)` returns the specified device objects from the MAT-file `file` as a structure to `out` instead of directly loading them into the workspace. The field names in `out` match the names of the loaded device objects. If no device objects are specified, then all variables existing in the MAT-file are loaded.

Tips

Loading device objects follows these rules:

- Unique device objects are loaded into the MATLAB workspace as well as the engine.
- If a loaded device object already exists in the engine but not the MATLAB workspace, the loaded device object automatically reconnects to the engine device object.

- If a loaded device object already exists in the workspace or the engine but has different properties than the loaded object, then these rules are followed:
  - The read-only properties are automatically reset to their default values.
  - All other property values are given by the loaded object and a warning is issued stating that property values of the workspace object have been updated.
- If the workspace device object is running, then it is stopped before loading occurs.
- If identical device objects are loaded, then they point to the same device object in the engine. For example, if you saved the array

```
x = [ai1 ai1 ai2]
```

  only `ai1` and `ai2` are created in the engine, and `x(1)` will equal `x(2)`.
- Values for read-only properties are restored to their default values upon loading. For example, the `EventLog` property is restored to an empty vector. Use the `propinfo` function to determine if a property is read only.
- Values for the `BufferingConfig` property when the `BufferingMode` property is set to `Auto`, and the `MaxSamplesQueued` property might not be restored to the same value because both these property values are based on available memory.

Note `load` is not used to read in acquired data that has been saved to a log file. You should use the `daqread` function for this purpose.

If you use the `help` command to display the help for `load`, then you must supply the pathname shown below.

```
help daq/private/load
```

Examples

This example illustrates the behavior of `load` when the loaded device object has properties that differ from the workspace object.

```
ai = analoginput('winsound');
addchannel(ai,1:2);
save ai
ai.SampleRate = 10000;
load ai
Warning: Loaded object has updated property values.
```

See Also

`daqread` | `propinfo` | `save`

makenames

Purpose

List descriptive channel or line names

Syntax

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

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`'prefix'`	A string that constitutes the first part of the name.
`index`	Numbers appended to the end of `prefix` — any MATLAB vector syntax can be used to specify `index` as long as the numbers are positive.
`names`	An m-by-1 cell array of channel names where m is the length of `index`.

Description

`names = makenames('prefix',index)` generates a cell array of descriptive channel or line names by concatenating `prefix` and `index`.

Tips

You can pass `names` as an input argument to the `addchannel` or `addline` function.

If `names` contains more than one descriptive name, then the size of `names` must agree with the number of hardware channels specified in `addchannel`, or the number of hardware lines specified in `addline`.

If the channels or lines are to be referenced by name, then `prefix` must begin with a letter and contain only letters, numbers, and underscores. Otherwise the names can contain any character.

Examples

Create the analog input object `AI`. You can use `makenames` to define descriptive names for each channel that is to be added to `AI`.

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

```
names = makenames('chan',1:8);
```

`names` is an eight-element cell array of channel names `chan1`, `chan2`,..., `chan8`. You can now pass `names` as an input argument to the `addchannel` function.

```
addchannel(AI,0:7,names);
```

See Also

`addchannel` | `addline`

muxchanidx

Purpose

Multiplexed scanned analog input channel index

Syntax

```
scanidx = muxchanidx(obj,muxboard,muxidx)
scanidx = muxchanidx(obj,absmuxidx)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	An analog input object associated with a National Instruments Traditional NI-DAQ board.
`muxboard`	The multiplexer board.
`muxidx`	The index number of the multiplexed channel.
`absmuxidx`	The absolute index number of the multiplexed channel.
`scanidx`	The scanning index number of the multiplexed channel.

Description

`scanidx = muxchanidx(obj,muxboard,muxidx)` returns the scanning index number of the multiplexed channel specified by `muxidx`. The multiplexer (mux) board is specified by `muxboard`. For each mux board, `muxidx` can range from 0-31 for differential inputs and 0-63 for single-ended inputs. `muxboard` and `muxidx` are vectors of equal length.

`scanidx = muxchanidx(obj,absmuxidx)` returns the scanning index number of the multiplexed channel specified by `absmuxidx. absmuxidx` is the absolute index of the channel independent of the mux board.

For single-ended inputs, the first mux board has absolute index values that range between 0 and 63, the second mux board has absolute index values that range between 64 and 127, the third mux board has absolute index values that range between 128 and 191, the fourth mux board has absolute index values that range between 192 and 255. For example, the absolute index value of the second single-ended channel on the fourth mux board (`muxboard` is 4 and `muxidx` is 1) is 193.

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. See the supported hardware page at `www.mathworks.com/products/daq/supportedio.html` for more information.

Tips

`scanidx` identifies the column number of the data returned by `getdata` and `peekdata`.

Refer to the *AMUX-64T User Manual* for more information about adding mux channels based on hardware channel IDs and the number of mux boards used.

Examples

Create the analog input object `ai` for a National Instruments board that is connected to four AMUX-64T multiplexers, and add 256 channels to `ai` using `addmuxchannel`.

```
ai = analoginput('nidaq',1);
ai.InputType = 'SingleEnded';
ai.NumMuxBoards = 4;
addmuxchannel(ai);
```

The following two commands return a scanned index value of 14.

```
scanidx = muxchanidx(ai,4,1);
scanidx = muxchanidx(ai,193);
```

See Also

`addmuxchannel`

obj2mfile

Purpose

Convert device objects, channels, or lines to MATLAB code

Syntax

```
obj2mfile(obj,'file')
obj2mfile(obj,'file','syntax')
obj2mfile(obj,'file','all')
obj2mfile(obj,'file','syntax','all')
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	A device object, array of device objects, channels, or lines.
`'file'`	The file that the MATLAB code is written to. The full pathname can be specified. If an extension is not specified, the `.m` extension is used.
`'`*`syntax`*`'`	Syntax of the converted the MATLAB code. By default, the `set` syntax is used. If `dot` is specified, then the subscripted referencing syntax is used. If `named` is specified, then named referencing is used (if defined).
`'`**`all`**`'`	If **`all`** is specified, all properties are written to `file`. If **`all`** is not specified, only properties that are not set to their default values are written to `file`.

Description

`obj2mfile(obj,'file')` converts `obj` to the equivalent MATLAB code using the `set` syntax and saves the code to `file`. By default, only those properties that are not set to their default values are written to `file`.

`obj2mfile(obj,'file','`*`syntax`*`')` converts `obj` to the equivalent MATLAB code using *`syntax`* and saves the code to `file`. The values for *`syntax`* can be `set`, `dot`, or `named`. `set` uses the set syntax, `dot` uses subscripted assignment (dot notation), and `named` uses named referencing (if defined).

`obj2mfile(obj,'file','all')` converts `obj` to the equivalent MATLAB code using the `set` syntax and saves the code to `file`. **`all`** specifies that all properties are written to `file`.

`obj2mfile(obj,'file','syntax','all')` converts `obj` including all of `obj`'s properties to the equivalent MATLAB code using *`syntax`* and saves the code to `file`.

Tips

If the `UserData` property is not empty or if any of the callback properties are set to a cell array of values or a function handle, then the data stored in those properties is written to a MAT-file when the object is converted and saved. The MAT-file has the same name as the file containing the object code (see the example below).

You can recreate the saved device objects by typing the name of the file at the Command Window. You can also recreate channels or lines, by typing the name of the file with a device object as the only input.

Examples

Create the analog input object `ai` for a sound card, add two channels, and set values for several properties.

```
ai = analoginput('winsound');
addchannel(ai,1:2);
set(ai,'Tag','myai','TriggerRepeat',4)
set(ai,'StartFcn',{@mycallback,2,magic(10)})
```

The following command writes MATLAB code to the files `myai.m` and `myai.mat`.

```
obj2mfile(ai,'myai.m','dot')
```

`myai.m` contains code that recreates the analog input code shown above using the dot notation for all properties that have their default values changed. Because `StartFcn` is set to a cell array of values, this property appears in `myai.m` as

```
ai.StartFcn = startfcn1;
```

and is saved in `myai.mat` as

```
startfcn1 = {@mycallback,2,magic(10)};
```

To recreate `ai` and assign the device object to a new variable `ainew`:

```
ainew = myai;
```

The associated MAT-file, `myai.mat`, is automatically loaded.

Purpose Preview most recent acquired analog input data

Syntax

```
data = peekdata(obj,samples)
data = peekdata(obj,samples,'type')
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	An analog input object.
`samples`	The number of samples to preview for each channel contained by `obj`.
`'type'`	Specifies the format of the extracted data as `double` (the default) or as `native`.
`data`	An m-by-n matrix where m is the number of samples and n is the number of channels.

Description

`data = peekdata(obj,samples)` returns the latest number of samples specified by `samples` to `data`.

`data = peekdata(obj,samples,'type')` returns the number of samples specified by `samples` in the format specified by *`type`* for each channel contained by `obj`. If *`type`* is specified as `native`, the data is returned in the native data format of the device. If *`type`* is specified as `double` (the default), the data is returned as doubles.

Tips

More About Using peekdata

- Unlike `getdata`, `peekdata` is a *nonblocking* function that immediately returns control to the MATLAB workspace. Because `peekdata` does not block execution control, data might be missed or repeated.
- `peekdata` takes a "snapshot" of the most recent acquired data and does not remove samples from the data acquisition engine. Therefore,

the SamplesAvailable property value is not affected when peekdata is called.

Rules for Using peekdata

- You can call peekdata before a trigger executes. Therefore, peekdata is useful for previewing data before it is logged to the engine or to a disk file.
- In most cases, you will call peekdata while the device object is running. However, you can call peekdata once after the device object stops running.
- If samples is greater than the number of samples currently acquired, all available samples are returned with a warning message stating that the requested number of samples were not available.
- If you start an analog input object and LoggingMode is Memory or Disk&Memory, extract the acquired data from the engine, using getdata. You can also flush it out using flushdata. If you do not extract or flush data, you receive a DataMissed event when the amount of acquired data reaches the MaxBytes limit for the object as seen by daqmem. The acquisition then stops.

Examples

Create the analog input object ai for a National Instruments board, add eight input channels, and configure ai for a two-second acquisition.

```
ai = analoginput('nidaq','Dev1');
addchannel(ai,0:7);
set(ai,'SampleRate',2000)
set(ai,'SamplesPerTrigger',4000)
```

After issuing the start function, you can preview the data.

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

peekdata returns 100 samples of data for each of the eight channels added to the object. If 100 samples are not available, then whatever samples are available will be returned and a warning message is issued. The data is not removed from the data acquisition engine.

See Also

`daqmem` | `flushdata` | `getdata` | `getsample` | `SamplesAvailable`

propinfo

Purpose

Property characteristics for device objects, channels, or lines

Syntax

```
out = propinfo(obj)
out = propinfo(obj,'PropertyName')
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	A device object, channels, or lines.
`'PropertyName'`	A valid `obj` property name.
`out`	A structure whose field names are the property names for `obj` (if *`PropertyName`* is not specified).

Description

`out = propinfo(obj)` returns the structure `out` whose field names are the property names for `obj`. Each property name in `out` contains the fields shown below.

Field Name	Description
`Type`	The property data type. Possible values are `any`, `callback`, `double`, and `string`.
`Constraint`	The type of constraint on the property value. Possible values are `bounded`, `callback`, `enum`, and `none`.
`ConstraintValue`	The property value constraint. The constraint can be a range of valid values or a list of valid string values.
`DefaultValue`	The property default value.

Field Name	Description
`ReadOnly`	Indicates when the property is read-only. Possible values are `always`, `never`, and `whileRunning`.
`DeviceSpecific`	If the property is device-specific, a 1 is returned. If a 0 is returned, the property is supported for all device objects of a given type.

`out = propinfo(obj,'PropertyName')` returns the structure `out` for the property specified by *`PropertyName`*. If *`PropertyName`* is a cell array of strings, a cell array of structures is returned for each property.

Examples

Create the analog input object `ai` for a sound card and configure it to operate in stereo mode.

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

To capture all property information for all common `ai` properties:

```
out = propinfo(ai);
```

To display the default value for the `SampleRate` property:

```
out.SampleRate.DefaultValue
ans =
        8000
```

To display all the property information for the `InputRange` property:

```
propinfo(ai.Channel,'InputRange')
ans =
               Type: 'double'
         Constraint: 'Bounded'
    ConstraintValue: [-1 1]
       DefaultValue: [-1 1]
           ReadOnly: 'whileRunning'
     DeviceSpecific: 0
```

See Also

`daqhelp`

Purpose

Queue analog output data in engine for eventual output

Syntax

```
putdata(obj,data)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	An analog output object.
`data`	The data to be queued in the engine.

Description

`putdata(obj,data)` queues the data specified by `data` in the engine for eventual output to the analog output subsystem. `data` must consist of a column of data for each channel contained by `obj`. That is, `data` must be an m-by-n matrix, where m rows correspond to the number of samples and n columns correspond to the number of channels in `obj`.

`data` can consist of doubles or native data types but cannot contain NaNs. `data` must contain a column of data for each channel contained in `obj`. If `data` contains any data points that are not within the `UnitsRange` of the channel it pertains to, the data points will be clipped to the bounds of the `UnitsRange` property.

`data` can be a `tscollection` object or `timeseries` object. If `data` is a `tscollection` object, there must be one `timeseries` per channel in `obj`. If `data` is a `timeseries` object, there must be only one channel in `obj`. If the `tscollection` or `timeseries` object contains gaps, or is sampled at a different rate than the `SampleRate` of `obj`, the data will be resampled at the rate of `obj` using a zero order hold.

For more information on using the Time Series functionality, see "Example: Time Series Objects and Methods" in the MATLAB documentation.

Tips

More About Queuing Data

- Data must be queued in the engine before `obj` is executed.
- `putdata` is a *blocking* function because it returns execution control to the MATLAB workspace only when the requested number of samples is queued in the engine for each channel group member.
- If the value of the `RepeatOutput` property is greater than 0, then all queued data is automatically requeued until the `RepeatOutput` value is reached. `RepeatOutput` must be configured before `start` is issued.
- After `obj` executes, you can continue to queue data unless `RepeatOutput` is greater than 0.
- Due to buffering constraints on certain devices, additional data queued close to the termination of the previous data may not be output to the device. To insure that all data is output, queue additional data well before the device has output all data.
- You can queue data in the engine until the value specified by the `MaxSamplesQueued` property is reached, or the limitations of your hardware or computer are reached.
- You should not modify the `BitsPerSample`, `InputRange`, `SensorRange`, and `UnitsRange` properties after calling `putdata`. If these properties are modified, all data is deleted from the data acquisition engine. If you add a channel after calling `putdata`, all data will be deleted from the buffer.
- The `timeseries` object must contain a single column of data.

More About Outputting Data

- Data is output as soon as a trigger occurs.
- An error is returned if a `NaN` is included in the data stream.
- You can specify `data` as the native data type of the hardware.
- If the output data is not within the range specified by the `OutputRange` property, then the data is clipped.

- The `SamplesOutput` property keeps a running count of the total number of samples that have been output per channel.
- The `SamplesAvailable` property tells you how many samples are ready to be output from the engine per channel. After data is output, `SamplesAvailable` is automatically reduced by the number of samples sent to the hardware.

Examples

Create the analog output object `ao` for a National Instruments board, add two output channels to it, and generate 10 seconds of data to be output.

```
ao = analogoutput('nidaq','Dev1');
ch = addchannel(ao,0:1);
set(ao,'SampleRate',1000)
data = linspace(0,1,10000)';
```

Before you can output `data`, it must be queued in the engine using `putdata`.

```
putdata(ao,[data data])
start(ao)
```

See Also

`putsample` | `timeseries` | `tscollection` | `MaxSamplesQueued` | `OutputRange` | `RepeatOutput` | `SamplesAvailable` | `SamplesOutput` | `Timeout` | `UnitsRange`

putsample

Purpose

Immediately output one analog output sample

Syntax

```
putsample(obj,data)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

Arguments

`obj`	An analog output object.
`data`	The data to be queued in the engine.

Description

`putsample(obj,data)` immediately outputs the row vector `data`, which consists of one sample for each channel contained by `obj`.

Tips

Using `putsample` is a good way to test your analog output configuration. Additionally:

- `putsample` does not store samples in the data acquisition engine.
- `putsample` can be executed at any time after channels have been added to `obj`.
- `putsample` is not supported for sound cards and Dynamic Signal Acquisition and Generation (DSA) cards.

Note Refer to the “Hardware Limitations by Vendor” section before you access National Instruments devices with the NI-DAQmx adaptor simultaneously from multiple applications.

Examples

Create the analog output object `ao` for a National Instruments board and add two hardware channels to it.

```
ao = analogoutput('nidaq','Dev1');
ch = addchannel(ao,0:1);
```

To call `putsample` for `ao`:

```
putsample(ao,[1 1])
```

See Also

`putdata`

putvalue

Purpose

Write values to digital output lines

Syntax

```
putvalue(obj,data)
putvalue(obj.Line(index),data)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

Arguments

`obj`	A digital I/O object.
`obj.Line(index)`	One or more lines contained by `obj`.
`data`	A decimal value or binary vector.

Description

`putvalue(obj,data)` writes `data` to the hardware lines contained by the digital I/O object `obj`.

`putvalue(obj.Line(index),data)` writes `data` to the hardware lines specified by `obj.Line(index)`.

Tips

More About Writing Values to Lines

- You can specify `data` as either a decimal value or a binary vector. A binary vector (or *binvec*) is constructed with the least significant bit (LSB) in the first column and the most significant bit (MSB) in the last column. For example, the decimal number 23 is written as the binary vector [1 1 1 0 1].
- If `obj` contains lines from a port-configurable device, then all lines will be written to even if they are not contained by the device object.
- An error will be returned if data is written to an input line.
- An error is returned if you attempt to write a negative value.
- If a decimal value is written to a digital I/O object and the value is too large to be represented by the hardware, then an error is returned.

Note Refer to the "Hardware Limitations by Vendor" section before you access National Instruments devices with the NI-DAQmx adaptor simultaneously from multiple applications.

Examples

Create the digital I/O object `dio` and add four output lines to it.

```
dio = digitalio('nidaq','Dev1');
lines = addline(dio,0:3,'out');
```

Write the value 8 as a decimal value and as a binary vector.

```
putvalue(dio,8)
putvalue(dio,[0 0 0 1])
```

Purpose

Save device objects to MAT-file

Syntax

```
save file
save file obj1 obj2...
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`file`	The MAT-file name.
`obj1 obj2...`	One or more device objects or an array of device objects.

Description

`save file` saves all the MATLAB variables to the MAT-file `file`. If an extension is not specified for `file`, then a `.MAT` extension is used.

`save file obj1 obj2...` saves the specified device objects to `file`.

Tips

Saving device objects follows these rules:

- You can use `save` in the functional form as well as the command form shown above. When using the functional form, you must specify the filename and device objects as strings.
- Samples associated with a device object are not stored in the MAT-file. You can bring these samples into the MATLAB workspace with the `getdata` function, and then save them to the MAT-file using a separate variable name. You can also log samples to disk by configuring the `LoggingMode` property to `Disk` or `Disk&Memory`.
- Values for read-only properties are restored to their default values upon loading. For example, the `EventLog` property is restored to an empty vector. Use the `propinfo` function to determine if a property is read only.

- Values for the `BufferingConfig` property (if the `BufferingMode` property is set to `Auto`) and the `MaxSamplesQueued` property might not be restored because both these property values are based on available memory.

If you use the `help` command to display the help for `save`, then you must supply this pathname:

```
help daq/private/save
```

See Also

`getdata` | `load` | `propinfo`

Purpose

Configure or display device object properties

Syntax

```
set(obj)
props = set(obj)
set(obj,'PropertyName')
props = set(obj,'PropertyName')
set(obj,'PropertyName',PropertyValue,...)
set(obj,PN,PV)
set(obj,S)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	A device object, array of device objects, channels, or lines.
`'PropertyName'`	A property name.
`PropertyValue`	A property value.
`PN`	A cell array of property names.
`PV`	A cell array of property values.
`S`	A structure whose field names are device object, channel, or line properties.
`props`	A structure array whose field names are the property names for `obj`, or a cell array of possible values.

Description

`set(obj)` displays all configurable properties for `obj`. If a property has a finite list of possible string values, then these values are also displayed.

`props = set(obj)` returns all configurable properties to `props`. `props` is a structure array with fields given by the property names, and possible property values contained in cell arrays. if the property does not have a finite set of possible values, then the cell array is empty.

`set(obj,'PropertyName')` displays the valid values for the property specified by *PropertyName*. *PropertyName* must have a finite set of possible values.

`props = set(obj,'PropertyName')` returns the valid values for *PropertyName* to `props`. `props` is a cell array of possible values or an empty cell array if the property does not have a finite set of possible values.

`set(obj,'PropertyName',PropertyValue,...)` sets multiple property values with a single statement. Note that you can use structures, property name/property value string pairs, and property name/property value cell array pairs in the same call to `set`.

`set(obj,PN,PV)` sets the properties specified in the cell array of strings `PN` to the corresponding values in the cell array `PV`. `PN` must be a vector. `PV` can be m-by-n where m is equal to the specified number of device objects, channels, or lines and n is equal to the length of `PN`.

`set(obj,S)` where `S` is a structure whose field names are device object properties, sets the properties named in each field name with the values contained in the structure.

Tips

If you use the `help` command to display the help for `set`, then you must supply the pathname shown below.

```
help daq/daqdevice/set
```

Examples

Create the analog input object `ai` for a sound card and configure it to operate in stereo mode.

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

To display all of `ai`'s configurable properties and their valid values:

```
set(ai)
```

To set the value for the `SampleRate` property to 10000:

```
set(ai,'SampleRate',10000)
```

The following two commands set the value for the `SampleRate` and `InputType` properties using one call to `set`.

```
set(ai,'SampleRate',10000,'TriggerType','Manual')
set(ai,{'SampleRate','TriggerType'},{10000,'Manual'})
```

You can also set different channel property values for multiple channels.

```
ch = ai.Channel(1:2);
set(ch,{'UnitsRange','ChannelName'},{[-1 1] 'Name1'; [-2 2]
'Name2'})
```

See Also

`get` | `setverify`

Purpose

Configure and return specified property

Syntax

```
Actual = setverify(obj,'PropertyName',PropertyValue)
Actual = setverify(obj.Channel(index),'PropertyName',PropertyValue)
Actual = setverify(obj.Line(index),'PropertyName',PropertyValue)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	A device object or array of device objects.
`'`*`PropertyName`*`'`	A property name.
`PropertyValue`	A property value.
`obj.Channel(index)`	One or more channels contained by `obj`.
`obj.Line(index)`	One or more lines contained by `obj`.
`Actual`	The actual value for the specified property.

Description

`Actual = setverify(obj,'`*`PropertyName`*`',PropertyValue)` sets *`PropertyName`* to `PropertyValue` for `obj`, and returns the actual property value to `Actual`.

`Actual = setverify(obj.Channel(index),'`*`PropertyName`*`', PropertyValue)` sets *`PropertyName`* to `PropertyValue` for the channels specified by `index`, and returns the actual property value to `Actual`.

`Actual = setverify(obj.Line(index),'`*`PropertyName`*`', PropertyValue)` sets *`PropertyName`* to `PropertyValue` for the lines specified by `index`, and returns the actual property value to `Actual`.

Tips

`setverify` is equivalent to the commands

```
set(obj,'PropertyName',PropertyValue)
Actual = get(obj,'PropertyName')
```

Using `setverify` is not required for setting property values, but it does provide a convenient way to verify the actual property value set by the data acquisition engine.

`setverify` is particularly useful when setting the `SampleRate`, `InputRange`, and `OutputRange` properties because these properties can only be set to specific values accepted by the hardware. You can use the `propinfo` function to obtain information about the valid values for these properties.

If a property value is specified but does not match a valid value, then

- If the specified value is within the range of supported values,
  - For the `SampleRate` and `InputRange` properties, the value is automatically rounded up to the next highest supported value.
  - For all other properties, the value is automatically selected to be the nearest supported value.
- If the value is not within the range of supported values, an error is returned and the current property value remains unchanged.

Examples

Create the analog input object `ai` for a National Instruments AT-MIO-16DE-10 board, add eight hardware channels to it, and set the sample rate to 10,000 Hz using `setverify`.

```
ai = analoginput('nidaq','Dev1');
ch = addchannel(ai,0:7);
ActualRate = setverify(ai,'SampleRate',10000);
```

Suppose you use `setverify` to set the input range for all channels contained by `ai` to -8 to 8 volts.

```
ActualInputRange = setverify(ai.Channel,'InputRange',[-8 8]);
```

The `InputRange` value was actually rounded up to -10 to 10 volts.

```
ActualInputRange{1}
ans =
   -10    10
```

See Also

`get` | `propinfo` | `set` | `InputRange` | `OutputRange` | `SampleRate`

showdaqevents

Purpose

Analog input and output event log information

Syntax

```
showdaqevents(obj)
showdaqevents(obj,index)
showdaqevents(struct)
showdaqevents(struct,index)
out = showdaqevents(...)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	An analog input or analog output object.
`index`	The event index.
`struct`	An event structure.
`out`	A one column cell array of event information.

Description

`showdaqevents(obj)` displays a summary of the event log for `obj`.

`showdaqevents(obj,index)` displays a summary of the events specified by `index` for `obj`.

`showdaqevents(struct)` displays a summary of the events stored in the structure `struct`.

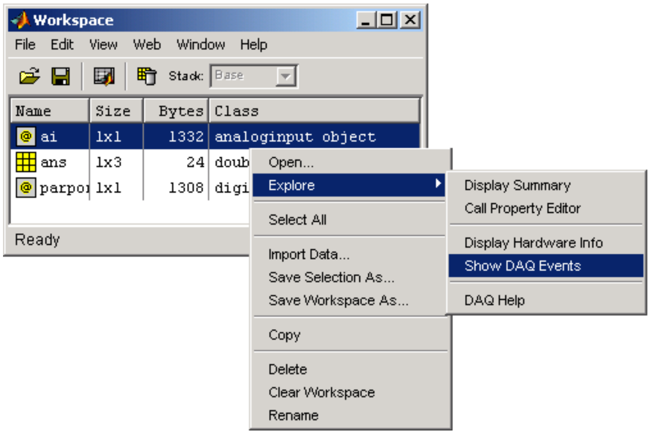
`showdaqevents(struct,index)` displays a summary of the events specified by `index` stored in the structure `struct`.

`out = showdaqevents(...)` outputs the event information to a one column cell array `out`. Each element of `out` is a string that contains the event information associated with that index value.

Tips

You can pass a structure of event information to `showdaqevents`. This structure can be obtained from the `getdata` function, the `daqread` function, or the `EventLog` property.

As shown below, you can also display event information via the Workspace browser by right-clicking a device object and selecting **Explore > Show DAQ Events** from the context menu.



Access context (pop-up) menus by right-clicking a device object.

Examples

Create the analog input object `ai` for a sound card, add two channels, and configure `ai` to execute three triggers.

```
ai = analoginput('winsound');
ch = addchannel(ai,1:2);
set(ai,'TriggerRepeat',2)
```

Start `ai` and display the trigger event information with `showdaqevents`.

```
start(ai)
showdaqevents(ai,2:4)

   2 Trigger#1       ( 17:07:06, 0 )          Channel: N/A
   3 Trigger#2       ( 17:07:07, 8000 )       Channel: N/A
   4 Trigger#3       ( 17:07:08, 16000 )      Channel: N/A
```

See Also

daqread | getdata | EventLog

Purpose

Size of device object, channel group, or line group

Syntax

```
d = size(obj)
[m1,m2,m3,...,mn] = size(obj)
m = size(obj,dim)
d = size(obj.Channel)
[m1,m2,m3,...,mn] = size(obj.Channel)
m = size(obj.Channel,dim)
d = size(obj.Line)
[m1,m2,m3,...,mn] = size(obj.Line)
m = size(obj.Line,dim)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

Arguments

`obj`	A device object or array of device objects.
`dim`	The dimension.
`obj.Channel`	The channels contained by `obj`.
`obj.Line`	The lines contained by `obj`.
`d`	A two-element row vector containing the number of rows and columns in `obj`.
`m1,m2,m3,...,mn`	Each dimension of `obj` is captured in a separate variable.
`m`	The length of the dimension specified by `dim`.

Description

`d = size(obj)` returns the two-element row vector `d = [m,n]` containing the number of rows and columns in `obj`.

`[m1,m2,m3,...,mn] = size(obj)` returns the length of the first `n` dimensions of `obj` to separate output variables. For example, `[m,n] =`

size(obj) returns the number of rows to m and the number of columns to n.

m = size(obj,dim) returns the length of the dimension specified by the scalar dim. For example, size(obj,1) returns the number of rows.

d = size(obj.Channel) returns the two-element row vector d = [m,n] containing the number of rows and columns in the channel group obj.Channel.

[m1,m2,m3,...,mn] = size(obj.Channel) returns the length of the first n dimensions of the channel group obj.Channel to separate output variables. For example, [m,n] = size(obj.Channel) returns the number of rows to m and the number of columns to n.

m = size(obj.Channel,dim) returns the length of the dimension specified by the scalar dim. For example, size(obj.Channel,1) returns the number of rows.

d = size(obj.Line) returns the two-element row vector d = [m,n] containing the number of rows and columns in the line group obj.Line.

[m1,m2,m3,...,mn] = size(obj.Line) returns the length of the first n dimensions of the line group obj.Line to separate output variables. For example, [m,n] = size(obj.Line) returns the number of rows to m and the number of columns to n.

m = size(obj.Line,dim) returns the length of the dimension specified by the scalar dim. For example, size(obj.Line,1) returns the number of rows.

Examples

Create the analog input object ai for a National Instruments board and add eight channels to it.

```
ai = analoginput('nidaq','Dev1');
ch = addchannel(ai,0:7);
```

To find the size of the device object:

```
size(ai)
ans =
```

```
     1     1
```

To find the size of the channel group:

```
size(ch)
ans =
     8     1
```

See Also

`length`

Purpose

Open data acquisition oscilloscope

Syntax

`softscope` `softscope(obj)` `softscope('fname.si')`

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	An analog input object.
`fname.si`	Name of the file containing Oscilloscope settings.

Description

`softscope` opens the Hardware Configuration graphical user interface (GUI), which allows you to configure the hardware device to be used with the Oscilloscope. The Oscilloscope opens when you click the **OK** button, and at least one hardware channel is selected.

`softscope(obj)` opens the Oscilloscope configured to display the data acquired from the analog input object, `obj`. `obj` must contain at least one hardware channel.

`softscope('fname.si')` pens the Oscilloscope using the settings saved in the softscope file specified by `fname`. `fname` is generated from the Oscilloscope's **File > Save** or **File > Save As** menu item.

Tips

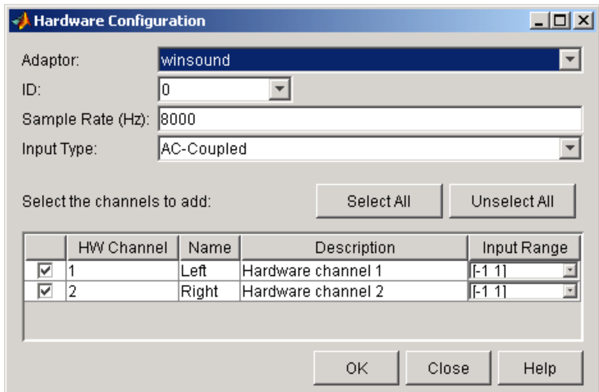
The Oscilloscope is a graphical user interface (GUI) that allows you to

- Stream acquired data into a display.
- Scale displayed data, and configure triggers and measurements.
- Configure analog input hardware settings.
- Export measurements and acquired data.

To support these tasks, the Oscilloscope includes several helper GUIs, which are described below.

Hardware Configuration

The Hardware Configuration GUI allows you to add channels from a particular hardware device to the Oscilloscope GUI. You can configure the device's sample rate and input type, as well as the input range for each added channel. The GUI shown below is configured to add both sound card channels using the default sample rate.



Oscilloscope

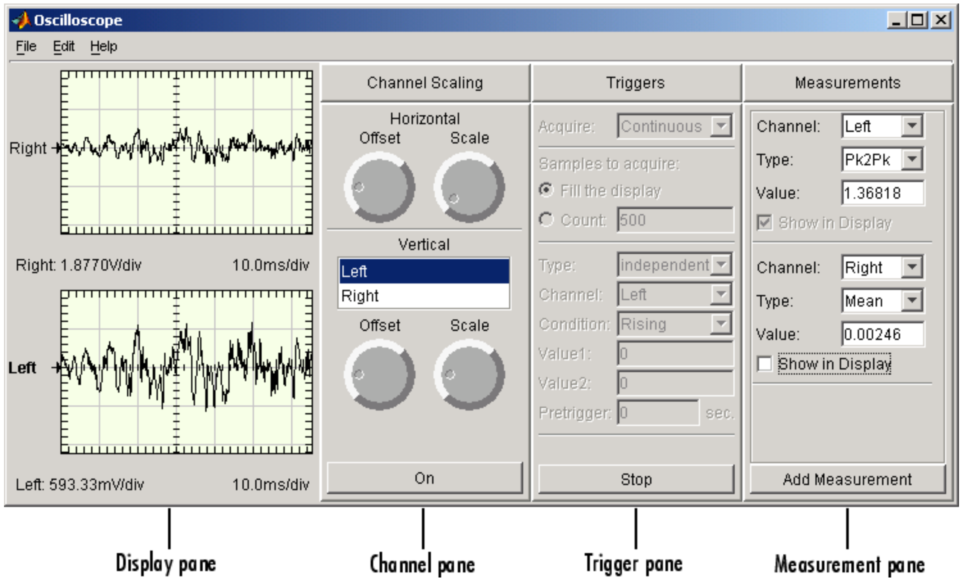
The Oscilloscope GUI consists of these panes:

- **Display** pane — The display pane contains the hardware channel data (a trace) and the measurements, if defined. The display area also contains labels for each channel's horizontal and vertical units, and indicators for
  - Each trace
  - The trigger level (if defined)
  - The location of the start of the trigger (used for pretriggers)
- **Channel** pane — The channel pane lists the hardware channels, math channels, and reference channels that are currently being viewed in a display. The Channel Panel also contains knobs for configuring

  - The display's horizontal offset and horizontal scale
  - The selected channel's vertical offset and vertical scale
- **Trigger** pane — The trigger pane allows you to define how data acquisition is initiated. There are three trigger types:
  - One-shot — Acquire the specified number of samples once.
  - Continuous — Continuously acquire the specified number of samples.
  - Sequence — Continuously acquire the specified number of samples, and use the dependent trigger type each time.

  For each trigger type, the Oscilloscope begins to acquire data after you press the **Trigger** button.
- **Measurement** pane — The measurement pane lists all measurements that are currently being taken. When defining a measurement, you must specify
  - The hardware, math, or reference channel
  - The measurement type
  - Whether the measurement result is drawn as a cursor in the display

The Oscilloscope GUI shown below is configured to display the sound card channels in separate displays.

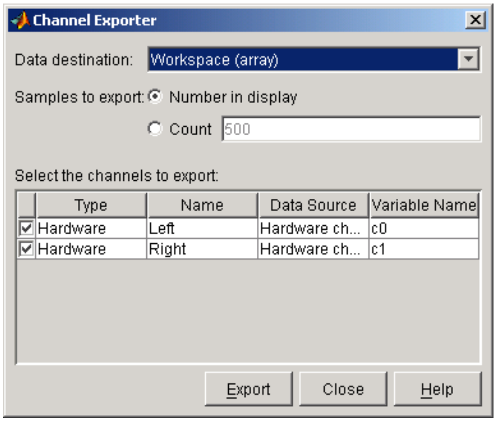


Channel Exporter

The Channel Exporter allows you to export the data associated with a hardware channel, a math channel, or a reference channel. You can export the channel data to one of four destinations:

- The MATLAB workspace as an array
- The MATLAB workspace as a structure
- A MATLAB figure window
- A MAT-file

All channels added to the oscilloscope are listed in the GUI.

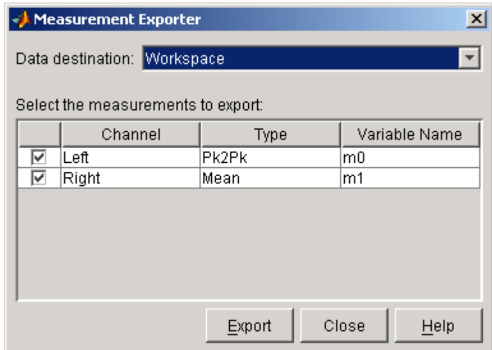


Measurement Exporter

The Measurement Exporter allows you to export the data associated with a measurement. You can export the measurement to one of three destinations:

- The MATLAB workspace
- A MATLAB figure window
- A MAT-file

The number of measurements exported depends on the `BufferSize` property value. By default, `BufferSize` is `1` indicating that the last measurement value calculated is available to export.

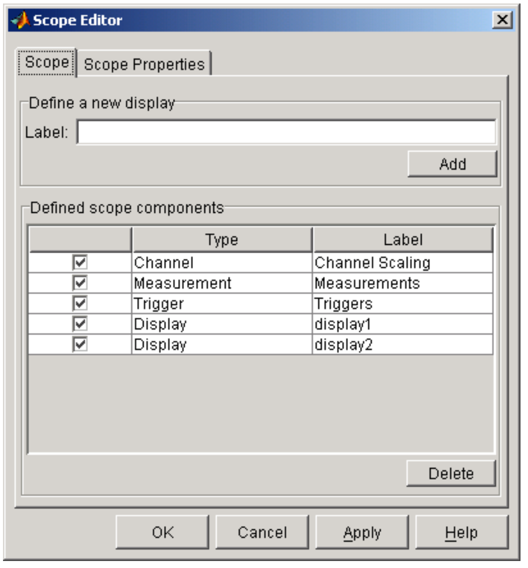


Scope Editor

The Scope Editor consists of two panes:

- **Scope** — Add and remove displays, the channel pane, the measurement pane, and the trigger pane. Note that you can define as many displays as you want, but there can only be only one channel pane, measurement pane, and trigger pane in the Oscilloscope at a time.

- **Scope Properties** — Configure properties for the displays, the channel pane, the measurement pane, and the trigger pane.

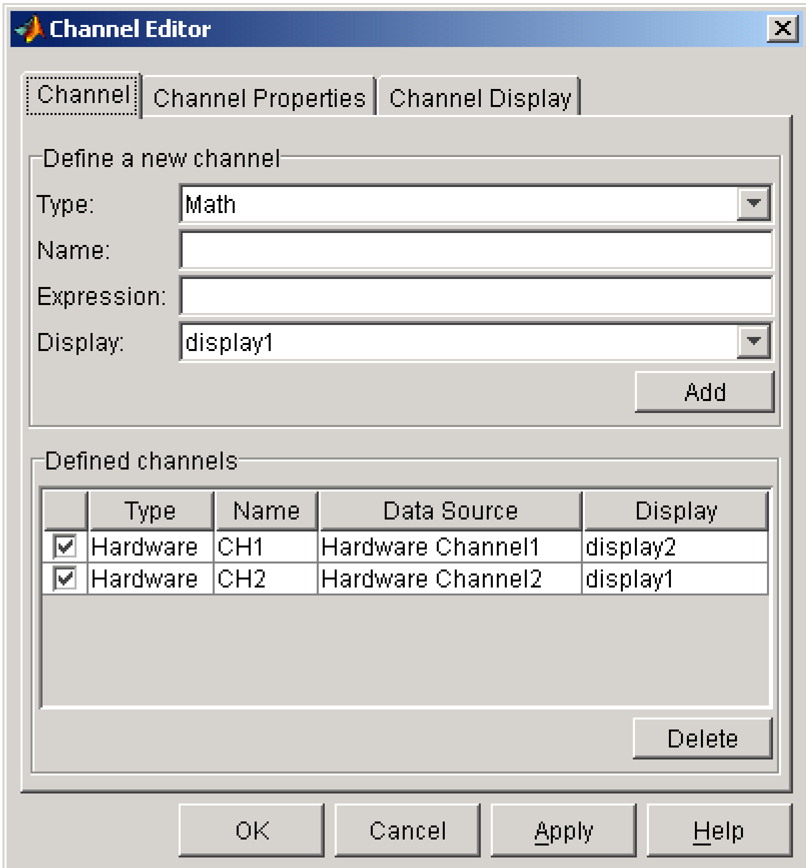


Channel Editor

The Channel Editor consists of three panes:

- **Channel** — Add or delete math channels and reference channels, and select which defined channels are available to the Oscilloscope.
- **Channel Properties** — Configure properties for defined hardware channels, math channels, and reference channels.

- **Channel Display** — Select the Oscilloscope display for each defined channel, or choose to not display a channel.

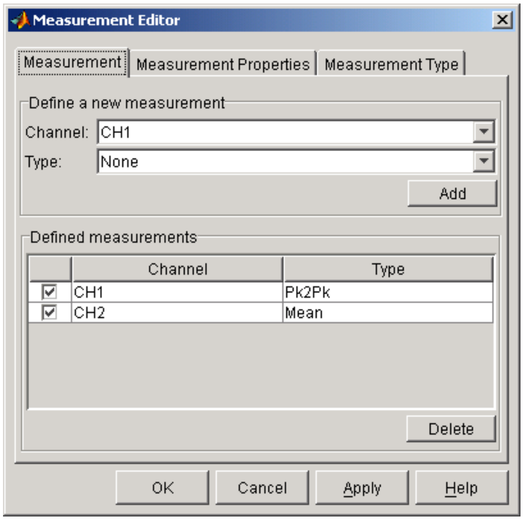


Measurement Editor

The Measurement Editor consists of three panes:

- **Measurement** — Add or delete measurements, and select which defined measurements are available to the Oscilloscope.

- **Measurement Properties** — Configure properties for the defined measurements.
- **Measurement Type** — Add or delete measurement types, and select which defined measurement types are available to the Oscilloscope.



Purpose

Start device object

Syntax

```
start(obj)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	A device object or an array of device objects.

Description

`start(obj)` initiates the execution of the device object `obj`.

Tips

When `start` is issued for an analog input or analog output object,

- The callback function specified for `StartFcn` is executed.
- The `Running` property is set to `On`.
- The start event is recorded in the `EventLog` property.
- Data existing in the engine is flushed.

Although an analog input or analog output object might be executing, data logging or sending is not necessarily initiated. Data logging or sending requires a trigger event to occur, and depends on the `TriggerType` property value.

For any device object, you can specify `start` as the value for a callback property.

```
ai.StopFcn = @start;
```

Note You typically execute a digital I/O object to periodically update and display its state. Refer to the `diopanel` example to see this behavior.

If you want to synchronize the input and output of data, or you require more control over when your hardware starts, you should use the `ManualTriggerHwOn` property.

See Also

`stop` | `trigger` | `EventLog` | `ManualTriggerHwOn` | `Running` | `Sending` | `TriggerType`

Purpose

Stop device object

Syntax

```
stop(obj)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	A device object or an array of device objects.

Description

`stop(obj)` terminates the execution of the device object `obj`.

Tips

An analog input object automatically stops when the requested samples are acquired or data is missed. An analog output object automatically stops when the queued data is output. These two device objects can also stop executing under one of these conditions:

- The `Timeout` property value is reached.
- A run-time error occurs.

For analog input objects, `stop` must be used when the `TriggerRepeat` property or `SamplesPerTrigger` property is set to `inf`. For analog output objects, `stop` must be used when the `RepeatOutput` property is set to `inf`. When `stop` is issued for either of these device objects,

- The `Running` property is set to `Off`.
- The `Logging` property or `Sending` property is set to `Off`.
- The callback function specified for `StopFcn` is executed.
- The stop event is recorded in the `EventLog` property.
- All pending callbacks for this object are discarded.

For any device object, you can specify `stop` as the value for a callback property.

```
ao.TimerFcn = @stop;
```

Note Issuing `stop` is the only way to stop an executing digital I/O object. You typically execute a digital I/O object to periodically update and display its state. Refer to the `diopanel` example.

See Also

`start` | `trigger` | `EventLog` | `Logging` | `RepeatOutput` | `Running` | `SamplesPerTrigger` | `Sending` | `Timeout` | `TriggerRepeat`

Purpose

Manually execute trigger for analog input or output object

Syntax

```
trigger(obj)
```

Note You cannot use the legacy interface on 64–bit MATLAB. See “Session-Based Interface” to acquire and generate data.

Arguments

`obj`	An analog input or analog output object or an array of these device objects.

Description

`trigger(obj)` manually executes a trigger.

Tips

After `trigger` is issued,

- The absolute time of the trigger event is recorded by the `InitialTriggerTime` property.
- The `Logging` property or `Sending` property is set to `On`.
- The callback function specified by `TriggerFcn` is executed.
- The trigger event is recorded in the `EventLog` property.

You can issue `trigger` only if `TriggerType` is set to `Manual`, `Running` is `On`, and `Logging` is `Off`.

See Also

`start` | `stop` | `InitialTriggerTime` | `Logging` | `Running` | `Sending` | `TriggerFcn` | `TriggerType`

Purpose

Wait until analog input or output device object stops running

Syntax

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

Note You cannot use the legacy interface on 64–bit MATLAB. See "Session-Based Interface" to acquire and generate data.

Arguments

`obj`	A device object or an array of device objects.
`waittime`	The maximum time to wait for `obj` to stop running.

Description

`wait(obj,waittime)` blocks the MATLAB Command Window, and waits for `obj` to stop running. You specify the maximum waiting time, in seconds, with `waittime`. `waittime` overrides the value specified for the `Timeout` property. If `obj` is an array of device objects, then `wait` might wait up to the specified time for each device object in the array.

`wait` is particularly useful if you want to guarantee that the specified data is acquired before another task is performed.

Tips

If `obj` is not running when `wait` is issued, or if an error occurs while `obj` is running, then `wait` immediately relinquishes control of the Command Window.

When `obj` stops running, its `Running` property is automatically set to `Off`. `obj` can stop running under one of these conditions:

- The requested number of samples is acquired (analog input) or sent out (analog output).
- The `stop` function is issued on that object.
- A run-time error occurs.
- The `Timeout` property value is reached (`waittime` supersedes this value).

All callbacks, including the `StopFcn`, are executed before `wait` returns.

Examples

Create the analog input object `ai` for a National Instruments board, add eight channels to it, and configure a 25-second acquisition.

```
ai = analoginput('nidaq','Dev1');
ch = addchannel(ai,0:7);
ai.SampleRate = 2000;
ai.TriggerRepeat = 4;
ai.SamplesPerTrigger = 10000;
```

You can use `wait` to block the MATLAB Command Window until all the requested data is acquired. Because the expected acquisition time is 25 seconds, the `waittime` argument is `26`. If the acquisition does not complete within this time, then a timeout occurs.

```
start(ai)
wait(ai,26)
```

See Also

`EventLog` | `Running` | `StopFcn` | `Timeout`

Purpose

Create data acquisition session for specific vendor hardware

Syntax

session = daq.createSession ('*vendor*')

Description

session = daq.createSession ('*vendor*') creates a session object that you can configure to perform operations using a CompactDAQ device.

Input Arguments

vendor

Enter the vendor name for the device you want to create a session object for. The session-based interface currently supports National Instruments devices only, represented with the abbreviation ni.

Output Arguments

session

The data acquisition session object that represents a session with hardware from a specific vendor.

Properties

Session acquisition and generation properties:

Channels	Array of channel objects associated with session object
Connections	Array of connections in a session
DurationInSeconds	Specify duration of acquisition
IsContinuous	Specify if operation continues until manually stopped
IsDone	Indicate if operation is complete
IsLogging	Indicate if hardware is acquiring or generating data
IsNotifyWhenDataAvailableExceedsAuto	Control if is set automatically
IsNotifyWhenScansQueuedBelowAuto	Control if is set automatically

`NotifyWhenDataAvailableExceeds`	Control firing of `DataAvailable` event
`NotifyWhenScansQueuedBelow`	Control firing of `DataRequired` event
`NumberOfScans`	Number of scans for operation when starting
`Range`	Specify channel measurement range
`Rate`	Rate of operation in scans per second
`RateLimit`	Limit of rate of operation based on hardware configuration
`ScansAcquired`	Number of scans acquired during operation
`ScansOutputByHardware`	Indicate number of scans output by hardware
`ScansQueued`	Indicate number of scans queued for output
`Vendor`	Vendor information associated with session object

Examples

Create a session object `s`:

```
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.
```

See Also

`daq.Session` | `daq.Session.addAnalogInputChannel` | `daq.Session.addAnalogOutputChannel` | `daq.getDevices` | `daq.getVendors`

How To

- "Session-Based Interface"

Purpose

Display available National Instruments devices

Syntax

```
daq.getDevices
device = daq.getDevices
```

Description

`daq.getDevices` lists National Instruments, including CompactDAQ devices available to your system. Use

`device = daq.getDevices` stores this list in the variable *`device`*.

Tips

Devices not supported by the toolbox are denoted with an `*`. For a complete list of supported CompactDAQ devices, see the Supported Hardware page in the Data Acquisition Toolbox area of the MathWorks Web site.

Output Arguments

device

The variable that you want to store a list of National Instruments devices available to your system.

Examples

Get a list of all devices available to your system and store it in the variable `d`:

```
 d = daq.getDevices

d =

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     cDAQ2Mod1 National Instruments NI 9201
10    ni     Dev1      National Instruments USB-6211
11    ni     Dev2      National Instruments USB-6218
12    ni     Dev3      National Instruments USB-6255
13    ni     Dev4      National Instruments USB-6363
14    ni     Dev5      National Instruments PCIe-6363
15    ni     PXI1Slot2 National Instruments PXI-4461
16    ni     PXI1Slot3 National Instruments PXI-4461
```

To get detailed information about a module on the chassis, type `d(`*`index`*`)`. For example, to get information about NI 9265, which has the index 5, type:

```
 d(5)

ans =

ni: National Instruments NI 9402 (Device ID: 'cDAQ1Mod5')
   Counter input subsystem supports:
      Rates from 0.1 to 80000000.0 scans/sec
      4 channels ('ctr0','ctr1','ctr2','ctr3')
      'EdgeCount','PulseWidth','Frequency','Position' measurement types

   Counter output subsystem supports:
      Rates from 0.1 to 80000000.0 scans/sec
      4 channels ('ctr0','ctr1','ctr2','ctr3')
      'PulseGeneration' measurement type

This module is in slot 5 of the 'cDAQ-9178' chassis with the name 'cDAQ1'.
```

You can also click on the name of the device in the list. You can now access detailed device information which includes:

- subsystem type

- rate
- number of available channels
- measurement type

See Also

`daq.Session` | `daq.getVendors` | `daq.createSession`

How To

- "Session-Based Interface"

daq.getVendors

Purpose Display available vendors

Syntax

```
daq.getVendors
vendor = daq.getVendors
```

Description `daq.getVendors` lists vendors available to your machine and MATLAB.

`vendor` `= daq.getVendors` stores this list in the variable *`vendor`*.

Note The current release of Data Acquisition Toolbox only supports National Instruments devices.

Output Arguments

vendor

The variable that stores a list of vendors available to your device.

Examples

Get a list of all vendors available to your machine and MATLAB and store it in the variable `v`:

```
 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 supp
Open the Support Package Installer to install additional vendors.
```

Data Acquisition Toolbox currently supports

- National Instruments, including CompactDAQ devices, denoted with the abbreviation `'ni'`.
- Digilent Analog Discovery™ devices denoted with `'digilent'`. To use this device use the Support Package Installer to download necessary drivers. For more information see “Digilent Analog Discovery Devices”.

See Also

`daq.Session` | `daq.getDevices` | `daq.createSession`

How To

- “Session-Based Interface”

daq.Session.addAnalogInputChannel

Purpose

Add analog input channel

Syntax

```
s.addAnalogInputChannel(deviceID, channelID,
measurementType)
ch = s.addAnalogInputChannel(deviceID, channelID,
   measurementType)
[ch, idx] = s.addAnalogInputChannel(deviceID, channelID,
   measurementType)
```

Description

`s.addAnalogInputChannel(deviceID, channelID, measurementType)` adds a channel on the device represented by *deviceID*, with the specified *channelID*, and channel measurement type, represented by *measurementType*, to the session *s*. Measurement types are vendor specific.

`ch = s.addAnalogInputChannel(deviceID, channelID, measurementType)` creates and displays the object *ch*.

`[ch, idx] = s.addAnalogInputChannel(deviceID, channelID, measurementType)` creates and displays the object *ch*, representing the channel that was added and the index, *idx*, which is an index into the array of the session object's `Channels` property.

Tips

- Use `daq.createSession` to create a session object before you use this method.
- To use counter channels, see `daq.Session.addCounterInputChannel`.

Input Arguments

deviceID

Specify the vendor-defined ID of the device. The specified channel is created for this device. The device ID is the ID of the device that you obtain by calling `daq.getDevices`.

channelID

Specify the ID of the channel added to the device. You can also add a range of channels.

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

Note Channel ID is the physical location of the channel on the device. The index for this channel displayed in the session indicates this channels position in the session. If you add a channel with channel ID `` `ai2' `` as the first channel in your session, the session index will be `1`.

---

measurementType

Specify a string that represents a vendor-defined measurement type. Measurement types include:

- `'Voltage'`
- `'Thermocouple'`
- `'Current'`
- `'Accelerometer'`
- `'RTD'`
- `'Bridge'`
- `'Microphone'`
- `'IEPE'`

Output Arguments

ch

Object representing the channel you added.

idx

An index into the array of the session object's `Channels` property.

Properties

ADCTimingMode	Set channel timing mode
BridgeMode	Specify analog input device bridge mode
Coupling	Specify input coupling mode
Device	Channel device information
ExcitationCurrent	Voltage of external source of excitation
ExcitationSource	External source of excitation
ExcitationVoltage	Voltage of excitation source
ExternalTriggerTimeout	Indicate if external trigger timed out
ID	ID of channel in session
MaxSoundPressureLevel	Sound pressure level for microphone channels
MeasurementType	Type counter channel measurement
Name	Specify descriptive name for the channel
NominalBridgeResistance	Resistance of sensor
R0	Specify resistance value
Range	Specify channel measurement range
RTDConfiguration	Specify wiring configuration of RTD device
RTDType	Specify sensor sensitivity
ScansAcquired	Number of scans acquired during operation

Sensitivity	Sensitivity of an analog channel
ShuntLocation	Indicate location of channel’s shunt resistor
ShuntResistance	Resistance value of channel’s shunt resistor
TerminalConfig	Specify terminal configuration
ThermocoupleType	Select thermocouple type
Units	Specify unit of RTD measurement

Examples

Add an analog input current channel:

```
s = daq.createSession ('ni')
s.addAnalogInputChannel('cDAQ1Mod3','ai0', 'Current');
```

Add an analog input thermocouple channel. Specify output arguments to represent the channel object and the index:

```
s = daq.createSession ('ni')
[ch, idx] = s.addAnalogInputChannel('cDAQ2Mod6', 'ai0', 'Thermocouple')
```

Add analog input voltage channels 0, 2, and 4:

```
s = daq.createSession ('ni')
s.addAnalogInputChannel('cDAQ1Mod1',[0 2 4], 'Voltage');
```

See Also

daq.Session.startBackground | daq.Session.startForeground | daq.Session.addAnalogOutputChannel | daq.Session.removeChannel | daq.Session

How To

- “Session-Based Interface”

daq.Session.addAnalogOutputChannel

Purpose

Add analog output channel

Syntax

```
s.addAnalogOutputChannel(deviceID, channelID,
   measurementType)
ch = s.addAnalogOutputChannel(deviceID, channelID,
   measurementType)
[ch, idx] = s.addAnalogOutputChannel(deviceID, channelID,
   measurementType)
```

Description

`s.addAnalogOutputChannel(deviceID, channelID, measurementType)` adds an analog output channel on the device represented by `deviceID`, with the specified `channelID`, and channel measurement type, defined by `measurementType`, on the session object, `s`. Measurement types are vendor specific.

`ch = s.addAnalogOutputChannel(deviceID, channelID, measurementType)` creates and displays the object `ch`, representing the channel that was added.

`[ch, idx] = s.addAnalogOutputChannel(deviceID, channelID, measurementType)` creates and displays the object `ch`, representing the channel that was added and the index, `idx`, which is an index into the array of the session object's `Channels` property.

Tips

- Use `daq.createSession` to create a session object before you use this method.
- To use counter channels, see `daq.Session.addCounterInputChannel`.

Input Arguments

deviceID

Specify the vendor-defined ID of the device on which you will create the analog output channel.

channelID

Specify the ID of the channel added to the device. You can also add a range of channels.

For NI devices, use either a terminal name, like `'ao2'`, or a numeric equivalent like `2`, for the channel ID.

measurementType

Specify a string that represents a vendor-defined measurement type. Measurement types include: .

- `'Voltage'`
- `'Current'`

Output Arguments

ch

Object representing the channel you added.

idx

An index into the array of the session object's `Channels` property.

Properties

`Device`	Channel device information
`ExcitationCurrent`	Voltage of external source of excitation
`ExcitationSource`	External source of excitation
`ExternalTriggerTimeout`	Indicate if external trigger timed out
`ID`	ID of channel in session
`MaxSoundPressureLevel`	Sound pressure level for microphone channels
`MeasurementType`	Type counter channel measurement
`Name`	Specify descriptive name for the channel

`Range`	Specify channel measurement range
`ScansOutputByHardware`	Indicate number of scans output by hardware
`ScansQueued`	Indicate number of scans queued for output
`Sensitivity`	Sensitivity of an analog channel
`TerminalConfig`	Specify terminal configuration

Examples

Add an analog output voltage channel:

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

Add four output current channels:

```
s = daq.createSession ('ni')
s.addAnalogOutputChannel('cDAQ1Mod8',0:3, 'Current');
```

See Also

`daq.session.startBackground` | `daq.session.startForeground` | `daq.Session.addAnalogInputChannel` | `daq.Session.removeChannel` | `daq.Session`

How To

- “Session-Based Interface”

Purpose

Remove channel from session object

Syntax

`s.removeChannel(idx)`

Description

`s.removeChannel(idx)` removes the channel specified by `idx` from the session object`s`.

Input Arguments

idx

Index of the channel in the session.

Examples

Start with a session `s`, with two analog input and two analog output voltage channels and display channel information.

```
s


s =


Data acquisition session using National Instruments hardware:
   No data queued.  Will run at 1000 scans/second.
   Operation starts immediately.
      Number of channels: 4
      index Type  Device    Channel InputType      Range       Name
      ----- ---- --------- ------- --------- ---------------- ----
      1     ai   cDAQ1Mod4 ai0     SingleEnd -10 to +10 Volts
      2     ai   cDAQ1Mod4 ai1     SingleEnd -10 to +10 Volts
      3     ao   cDAQ1Mod2 ao0     n/a       -10 to +10 Volts
      4     ao   cDAQ1Mod2 ao1     n/a       -10 to +10 Volts
```

Remove channel `'ai0'` currently with the index `1` from the session:

```
s.removeChannel(1)
```

To see how the indexes shift after you remove a channel, type:

```
s


s =
```

```
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 InputType      Range        Name
      ----- ---- --------- ------- --------- ---------------- ----
      1     ai   cDAQ1Mod4 ai1     SingleEnd -10 to +10 Volts
      2     ao   cDAQ1Mod2 ao0     n/a       -10 to +10 Volts
      3     ao   cDAQ1Mod2 ao1     n/a       -10 to +10 Volts
```

Remove the first output channel `'ao0'` currently at index 2:

```
s.removeChannel(2)
```

The session displays one input and one output channel:

```
s.Channels

ans =

Number of channels: 2
   index Type  Device    Channel InputType      Range        Name
   ----- ---- --------- ------- --------- ---------------- ----
   1     ai   cDAQ1Mod4 ai1     SingleEnd -10 to +10 Volts
   2     ao   cDAQ1Mod2 ao1     n/a       -10 to +10 Volts
```

To remove multiple channels together, type the channel indexes as an array. Create a session with multiple channels:

```
s = daq.createSession('ni');
s.addAnalogInputChannel('cDAQ1Mod7',1:3, 'Voltage');
s.addAnalogInputChannel('cDAQ1Mod7',0, 'Current');
s.addCounterInputChannel('Dev2',0:1,'EdgeCount');
s
```

```
s =

Data acquisition session using National Instruments hardware:
   Will run for 1 second (2 scans) at 2 scans/second.
   Number of channels: 6
      index Type  Device    Channel MeasurementType       Range        Name
      ----- ---- --------- ------- --------------- ------------------ ----
      1     ai   cDAQ1Mod7 ai1     Voltage (Diff)  -60 to +60 Volts
      2     ai   cDAQ1Mod7 ai2     Voltage (Diff)  -60 to +60 Volts
      3     ai   cDAQ1Mod7 ai3     Voltage (Diff)  -60 to +60 Volts
      4     ai   cDAQ1Mod7 ai0     Current         -0.025 to +0.025 A
      5     ci   Dev2      ctr0    EdgeCount       n/a
      6     ci   Dev2      ctr1    EdgeCount       n/a
```

Remove channel 2 to channel 5 with one `daq.Session.removeChannel` command:

```
s.removeChannel(2:5);
s

s =

Data acquisition session using National Instruments hardware:
   Will run for 1 second (2 scans) at 2 scans/second.
   Number of channels: 2
      index Type  Device    Channel MeasurementType      Range       Name
      ----- ---- --------- ------- -------------- --------------- ----
      1     ai   cDAQ1Mod7 ai1     Voltage (Diff)  -60 to +60 Volts
      2     ci   Dev2      ctr1    EdgeCount       n/a
```

See Also

`daq.Session.addAnalogInputChannel` | `daq.Session.addAnalogOutputChannel`

daq.Session.startBackground

Purpose

Start background operations

Syntax

`s.startBackground()`

Description

`s.startBackground()` starts the operation of the session object, `s`, without blocking MATLAB command line and other code. To block MATLAB execution, use `daq.Session.startForeground`.

When you use `startBackground()` with analog input channels, the operation uses the `DataAvailable` event to deliver the acquired data. This event is fired periodically while an acquisition is in progress. For more information, see “Events and Listeners — Concepts”.

When you add analog output channels to the session, you must call `queueOutputData()` before calling `startBackground()`.

During a continuous generation, the `DataRequired` event is fired periodically to request additional data to be queued to the session. See `DataRequired` for more information.

By default, the `IsContinuous` property is set to `false` and the operation stops automatically. If you have set it to `true`, use `daq.Session.stop` to stop background operations explicitly.

Use `daq.Session.wait` to block MATLAB execution until a background operation is complete.

Tips

- If your session has analog input channels, you must use a `DataAvailable` event to receive the acquired data in a background acquisition.
- If your session has analog output channels and is continuous, you can use a `DataRequired` event to queue additional data during background generations.
- Create an acquisition session and add a channel before you use this method. See `daq.createSession` for more information.
- Call `daq.Session.prepare` to reduce the latency associated with startup and to preallocate resources.

- Use an `ErrorOccurred` event to display errors during an operation.

Examples

Acquire data in the background by creating a session and adding a listener to access the acquired data using an anonymous function:

```
s = daq.createSession ('ni');
s.addAnalogInputChannel('cDAQ1Mod1', 'ai0', 'Voltage');
lh = s.addlistener('DataAvailable', @plotData);

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

Start the session and perform other MATLAB operations:

```
s.startBackground();
% perform other MATLAB operations.
```

Since this is not a continuous operation, the operation stops automatically.

Delete the listener:

```
delete (lh)
```

For a continuous background generation, add a listener event to queue additional data to be output:

```
s = daq.createSession('ni');
s.addAnalogOutputChannel('cDAQ1Mod2', 0, 'Voltage');
s.IsContinuous = true;
s.Rate = 10000;
data =  linspace(-1, 1, 5000)';
lh = s.addlistener('DataRequired', ...
        @(src,event) src.queueOutputData(data));
s.queueOutputData(data)
s.startBackground();
```

```
% perform other MATLAB operations during the generation.
```

The operation is continuous,

```
s.stop();
delete(lh);
```

See Also

daq.createSession | daq.Session.startForeground | daq.Session.addAnalogOutputChannel | daq.Session.addAnalogInputChannel | daq.Session.prepare | daq.Session.addlistener | daq.Session | DataAvailable | DataRequired | ErrorOccurred

Purpose

Start foreground operations

Syntax

```
s.startForeground
data = s.startForeground
[data,timeStamps,triggerTime] = s.startForeground
```

Description

`s.startForeground` starts operations of the session object, `s`, and blocks MATLAB command line and other code until the session operation is complete.

To perform other MATLAB operations while the session operation executes, use `daq.Session.startBackground`.

`data = s.startForeground` returns the data acquired in the output parameter, *data*.

`[data,timeStamps,triggerTime] = s.startForeground` returns the data acquired, timestamps relative to the time the operation is triggered, and a trigger time indicating the absolute time the operation was triggered.

If a session includes output channels, call `queueOutputData` before calling `startForeground`.

You cannot perform continuous operations using `startForeground`. To perform continuous operations use `daq.Session.startBackground`.

Output Arguments

data

An `mxn` matrix of doubles where `m` is the number of scans acquired, and `n` is the number of input channels in the session.

timeStamps

The timestamps relative to the time the operation is triggered in an `mx1` array where `m` is the number of scans.

triggerTime

A MATLAB serial date time stamp representing the absolute time when `timeStamps = 0`.

Examples

Acquire data by creating a session with an analog input channel:

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

Start the acquisition and save the acquired data into the variable `data`:

```
 data = s.startForeground;
```

Generate a signal by creating a session with an analog output channel:

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

Create and queue an output signal and start the generation:

```
outputSignal = linspace(-1, 1, 1000);
s.queueOutputData(outputSignal);
s.startForeground;
```

See Also

`daq.createSessiondaq.Session.startBackground` **|** `daq.Session.addAnalogInputChannel` **|** `daq.Session.addAnalogOutputChannel` **|** `daq.Session.addDigitalChannel` **|**

Related Examples

- "Acquire Data in the Foreground"
-

Concepts

- "Session-Based Interface and Data Acquisition Toolbox"

daq.Session.addlistener

Purpose

Create event listener

Syntax

```
lh = addlistener('eventName',@callback)
lh = addlistener('eventName', @(src, event) expr)
```

Description

lh = addlistener('*eventName*',@*callback*) creates a listener for the specified event, `eventName`, and fires the callback function, `callback`. *lh* is the variable in which the listener handle is stored. Create a callback function that executes when the listener detects the specified event. The callback can be any MATLAB function.

lh = addlistener('*eventName*', @(src, event) *expr*) creates a listener for the specified event, `eventName`, and fires an anonymous callback function. The anonymous function uses the specified input arguments and executes the operation specified in the expression `expr`. Anonymous functions provide a quick means of creating simple functions without storing them to a file. For more information, see Anonymous Functions.

Tips

- You must delete the listener once the operation is complete.

  ```
  delete (lh)
  ```

Input Arguments

'eventName'

Name of the event to listen for. Available events include:

- `'DataAvailable'`
- `'DataRequired'`
- `'ErrorOccurred'`

callback

Name of the function to execute when the specified event occurs.

src

The session object, where the event occurred.

event

Specified event object. For more information, see Session Events.

expr

Expression that represents the body of the function.

Output Arguments

lh

Handle to the event listener returned by `addlistener`. Delete the listener once the operation completes.

Examples

Add a listener to an acquisition session by first:

- Creating a session
- Adding an analog input channel

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

Add a listener for the `DataAvailable` event:

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

Create the `plotData` callback function and save it as `plotData.m`:

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

Acquire data in the background:

```
s.startBackground();
```

Wait for the operation to complete and delete the listener:

```
delete (lh)
```

Add a listener using an anonymous function to a signal generation session by first:

- Creating a session.
- Setting `IsContinuous` to `true`.
- Adding two analog output channels.

```
s = daq.createSession('ni');
s.IsContinuous = true;
s.addAnalogOutputChannel('cDAQ1Mod2', 0:1, 'Voltage');
```

Create output data for the two channels:

```
outputData0 = linspace(-1, 1, 1000)';
outputData1 = linspace(-2, 2, 1000)';
```

Queue the output data:

```
s.queueOutputData([outputData0 outputData1]);
```

Add an anonymous listener and generate the signal in the background:

```
lh = s.addlistener('DataRequired', @(src,event)...
     src.queueOutputData([outputData0 outputData1]));
```

Generate signals in the background:

```
s.startBackground();
```

Perform other MATLAB operations, and then stop the session.

```
s.stop ()
```

Delete the listener:

```
delete (lh)
```

See Also

daq.createSession | daq.Session.addAnalogInputChannel | daq.Session.addAnalogOutputChannel | daq.Session.startBackground | DataAvailable | DataRequired | ErrorOccurred

How To

- "Working with the Session-Based Interface"

Purpose

Prepare session for operation

Syntax

```
s.prepare
```

Description

`s.prepare` configures and allocates hardware resources for the session and reduces the latency of `daq.Session.startBackground` and `daq.Session.startForeground` methods. This method is optional and is automatically called as needed.

See Also

`daq.Session.addAnalogInputChannel` | `daq.Session.addAnalogInputChannel` | `daq.Session.addAnalogOutputChannel` | `daq.Session.release`

daq.Session.wait

Purpose

Block MATLAB until background operation completes

Syntax

```
s.wait ()
s.wait (timeout)
```

Description

`s.wait ()` blocks MATLAB until the background operation completes. Press **Ctrl+C** to abort the wait.

`s.wait (timeout)` blocks MATLAB until the operation completes or the specified time-out occurs.

Tips

- You cannot call `wait` if you have set the session's `IsContinuous` property to `true`.
- To terminate the operation, use `daq.Session.stop`

Input Arguments

timeout

Maximum time in seconds before the wait throws an error.

Examples

Create a session and add an analog output channel:

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

Queue some output data:

```
s.queueOutputData(zeros(10000,1));
```

Start the session and issue a wait. This blocks MATLAB for all data is output.

```
s.startBackground();
% perform other MATLAB operations.
s.wait()
```

Queue more data and wait for up to 15 seconds:

```
s.queueOutputData(zeros(10000,1));
```

```
s.startBackground();
% perform other MATLAB operations.
s.wait(15);
```

See Also

daq.Session.startBackground | daq.Session.stop

daq.Session.stop

Purpose Stop background operation

Syntax `s.stop()`

Description `s.stop()` stops the session and all associated hardware operations in progress. If your operation has acquired data and the `DataAvailable` event has not yet fired, the `stop` command will fire the event and deliver the data acquired up to that point.

See Also `daq.Session.startBackground` | `daq.Session.wait` | `daq.Session`

Purpose

Release session resources

Syntax

`s.release ()`

Description

`s.release ()` releases all reserved hardware resources.

When you associate hardware with a session using the Data Acquisition Toolbox, the session reserves exclusive access to the data acquisition hardware.

Hardware resources associated with a session are automatically released when you delete the session object, or you assign a different value to the variable containing your session object. Optionally, you can use `s.release` to release reserved hardware resources if you need to use it in another session or to use applications other than MATLAB to access the hardware.

Examples

Create a session and add an analog input voltage channel and acquire data in the foreground:

```
s1 = daq.createSession('ni');
s1.addAnalogInputChannel('cDAQ3Mod1', 'ai0', 'Voltage');
s1.startForeground;
```

Release the session hardware and create another session object with an analog input voltage channel on the same device as the previous session. Acquire in the foreground:

```
s1.release()
s2 = daq.createSession('ni');
s2.addAnalogInputChannel('cDAQ3Mod1', 'ai2', 'Voltage');
s2.startForeground;
```

See Also

`daq.Session.prepare` | `daq.Session.startForeground` | `daq.Session.startBackground` | `daq.Session`

daq.Session.inputSingleScan

Purpose

Acquire single scan from all input channels

Syntax

```
data = s.inputSingleScan()
[data, triggerTime] = s.inputSingleScan()
```

Description

data = *s*.inputSingleScan() immediately acquires a single scan from each input channel in the session and returns the data acquired in a 1xn array of doubles, data, where n is the number of input channels in the session.

[*data, triggerTime*] = *s*.inputSingleScan() immediately acquires a single scan from each input channel in the session and returns the data acquired in a 1xn array of doubles, data, where n is the number of input channels in the session. triggerTime is a MATLAB serial date time stamp representing the time the data is acquired.

Tips

To acquire more than a single input, use daq.Session.startForeground.

Output Arguments

data

A 1xn array of doubles, where n is the number of input channels in the session.

triggerTime

A MATLAB serial date time stamp representing the time the data is acquired.

Examples

Analog Input

Acquire a single input from an analog channel.

Create a session and add two analog input channels:

```
s = daq.createSession('ni');
s.addAnalogInputChannel('cDAQ1Mod1', 1:2, 'Voltage');
```

Input a single scan:

```
data = s.inputSingleScan

data =

   -0.1495    0.8643
```

Digital Input

Acquire a single input from a digital channel.

Create a session and add two digital channels with `InputOnly` measurement type:

```
s = daq.createSession('ni');
s.addDigitalChannel('dev1', 'Port0/Line0:1', 'InputOnly');
```

Input a single scan:

```
data = s.inputSingleScan
```

Counter Input

Acquire a single input from a counter channel.

Create a session and add a counter input channel with `EdgeCount` measurement type:

```
s = daq.createSession('ni');
s.addCounterInputChannel('Dev1',0,'EdgeCount');
```

Input a single edge count:

```
data = s.inputSingleScan
```

See Also

`daq.createSession` | `daq.Session.addAnalogInputChannel` | `daq.Session.addCounterInputChannel` | `daq.Session.addDigitalChannel` | `daq.Session.outputSingleScan` | `daq.Session.startForeground`

daq.Session.queueOutputData

Purpose

Queue data to be output

Syntax

`s.queueOutputData (data)`

Description

`s.queueOutputData (data)` queues data to be output. When using analog output channels, you must queue data before you call `daq.Session.startForeground` or `daq.Session.startBackground`.

Input Arguments

data

An `mxn` matrix of doubles where `m` is the number of scans to generate, and `n` is the number of output channels in the session.

Examples

Create a session, add an analog output channel, and queue some data to output:

```
s = daq.createSession('ni');
s.addAnalogOutputChannel('cDAQ1Mod2', 'ao0', 'Voltage');
s.queueOutputData (linspace(-1, 1, 1000)');
s.startForeground;
```

Queue output data for multiple channels:

```
s = daq.createSession('ni');
s.addAnalogOutputChannel('cDAQ1Mod2', 0:1, 'Voltage');
data0 = linspace(-1, 1, 1000)';
data1 = linspace(-2, 2, 1000)';
s.queueOutputData ([data0 data1]);
s.startBackground
```

See Also

`daq.createSession` | | `daq.Session.addAnalogOutputChannel` | `daq.Session.startBackground` | `daq.Session.startForeground`

Purpose

Generate single scan on all output channels

Syntax

`s.outputSingleScan(data)`

Description

`s.outputSingleScan(data)` outputs a single scan of data on one or more analog output channels.

Input Arguments

data

A `1xn` matrix of doubles where `n` is the number of output channels in the session.

Examples

Analog Output

Output a single scan on two analog output voltage channels

Create a session and add two analog output channels.

```
s = daq.createSession('ni');
s.addAnalogOutputChannel('cDAQ1Mod2', 0:1, 'Voltage');
```

Create an output value and output a single scan for each channel added.

```
s.outputSingleScan([1.5 4]);
```

Digital Output

Output one value each on 2 lines on a digital channel

Create a session and add two digital channels from port 0 that measures output only:

```
s = daq.createSession('ni');
s.addDigitalChannel('dev1', 'Port0/Line0:1', 'OutputOnly')
```

Output one value each on the two lines:

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

See Also

`daq.createSession` | `daq.Session.addAnalogOutputChannel` | | `daq.Session.addDigitalChannel` | `daq.Session.outputSingleScan` | `daq.Session.inputSingleScan`

Purpose

Notify when acquired data is available to process

Syntax

```
lh=session.addlistener('DataAvailable',callback);
lh=session.addlistener('DataAvailable', @(src, event), expr)
```

Description

`lh=session.addlistener('DataAvailable',callback);` creates a listener for the `DataAvailable` event. When data is available to process, the callback is executed. The callback can be any MATLAB function with the `(src, event)` signature.

`lh=session.addlistener('DataAvailable', @(src, event), expr)` creates a listener for the `DataAvailable` event and fires an anonymous callback function. The anonymous function requires the specified input arguments and executes the operation specified in the expression `expr`. Anonymous functions provide a quick means of creating simple functions without storing your function to a file. For more information see Anonymous Functions.

The callback has two required parameters: `src` and `event`. `src` is the session object for the listener and `event` is a `daq.DataAvailableInfo` object containing the data associated and timing information. Properties of `daq.DataAvailableInfo` are:

Data

An `mxn` matrix of doubles where `m` is the number of scans acquired, and `n` is the number of input channels in the session.

TimeStamps

The timestamps relative to `TriggerTime` in an `mx1` array where `m` is the number of scans acquired.

TriggerTime

A MATLAB serial date time stamp representing the absolute time the acquisition trigger occurs.

DataAvailable

Tips

- Frequency with which the `DataAvaialble` event is fired, is controlled by `NotifyWhenDataAvailableExceeds`

Examples

Create a session, add an analog input channel, and change the duration of the acquisition:

```
s = daq.createSession('ni');
s.addAnalogInputChannel('cDAQ1Mod1', 'ai0', 'Voltage');
s.DurationInSeconds = 5;
```

To add a listener for the `DataAvailable` event to plot the data, type:

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

Create a function that plots the data when the event occurs:

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

Start the acquisition and wait:

```
s.startBackground();
s.wait();
```

Delete the listener:

```
delete (lh)
```

See Also

`daq.Session.addlistener` | `daq.Session` | `daq.Session.startBackground` | `NotifyWhenDataAvailableExceeds` | `IsNotifyWhenDataAvailableExceedsAuto` |

Related Examples

- “Acquire Data in the Background”

Purpose

Notify when additional data is required for output on continuous generation

Syntax

```
lh = session.addlistener('DataRequired',callback);
lh = session.addlistener('DataRequired',
@(src,event) expr);
```

Description

`lh = session.addlistener('DataRequired',callback);` creates a listener for the `DataRequired` event. When more data is required, the callback is executed. The callback can be any MATLAB function with the `(src, event)` signature.

`lh = session.addlistener('DataRequired', @(src,event) expr);` creates a listener for the `DataRequired` event and fires an anonymous function. The anonymous function requires the specified input arguments and executes the operation specified in the expression `expr`. Anonymous functions provide a quick means of creating simple functions without storing your function to a file. For more information see Anonymous Functions.

The callback has two required parameters:. `src` and `event`. `src` is the session object for the listener and `event` is a `daq.DataRequiredInfo` object.

Tips

- The callback is typically used to queue more data to the device.
- Frequency is controlled by `NotifyWhenScansQueuedBelow`.

Examples

Add an anonymous listener to a signal generation session by first:

- Creating a session
- Adding two analog output channels

```
s = daq.createSession('ni');
s.IsContinuous = true
s.addAnalogOutputChannel('cDAQ1Mod2', 0:1, 'Voltage');
```

Create output data for the two channels :

```
outputData0 = (linspace(-1, 1, 1000))';
outputData1 = (linspace(-2, 2, 1000))';
```

Queue the output data and add an anonymous listener and generate the signal in the background:

```
s.queueOutputData([outputData0, outputData1]);
lh = s.addlistener('DataRequired', ...
    @(src,event) src.queueOutputData([outputData0, outputData1]));
```

Generate data and pause for up to 15 seconds:

```
s.startBackground();
pause (15)
```

Delete the listener:

```
delete (lh)
```

See Also

daq.Session.addlistener | IsContinuous | daq.Session | NotifyWhenScansQueuedBelow | IsNotifyWhenScansQueuedBelowAuto

Purpose

Notify when device-related errors occur

Syntax

`lh=session.addlistener('ErrorOccurred', callback);`
`lh=session.addlistener('ErrorOccurred' @(src,event), expr);`

Description

`lh=session.addlistener('ErrorOccurred', callback);` creates a listener for the `ErrorOccurred` event. When an error occurs, the call back is executed. The callback can be any MATLAB function with the `(src, event)` signature.

`lh=session.addlistener('ErrorOccurred' @(src,event), expr);` creates a listener for the `ErrorOccurred` event and fires an anonymous function. The anonymous function requires the specified input arguments and executes the operation specified in the expression `expr`. Anonymous functions provide a quick means of creating simple functions without storing your function to a file. For more information, see Anonymous Functions.

The callback has two required parameters: `src` and `event`. `src` is the session object for the listener and `event` is a `daq.ErrorOccurredInfo` object. The `daq.ErrorOccurredInfo` object contains the `Error` property, which is the `MException` associated with the error. You could use the `MException.getReport` method to return a formatted message string that uses the same format as errors thrown by internal MATLAB code.

Examples

Create a session, and add an analog input channel:

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

To get a formatted report of the error, type:

```
lh = s.addlistener('ErrorOccurred' @(src,event), disp(event.Error.getReport()));
```

Acquire data, wait and delete the listener:

```
s.startBackground();
s.wait()
```

```
delete(lh)
```

See Also

daq.Session.addlistener | daq.Session.startBackground | daq.Session | MException

Purpose

Add counter input channel

Syntax

```
s.addCounterInputChannel(deviceID, channelID,
   measurementType)
ch = s.addCounterInputChannel(deviceID, channelID,
   measurementType)
[ch, idx] = s.addCounterInputChannel(deviceID, channelID,
   measurementType)
```

Description

s.addCounterInputChannel(*deviceID*, *channelID*, *measurementType*) adds a counter channel on the device represented by *deviceID*, with the specified *channelID*, and channel measurement type, represented by *measurementType*, to the session *s*. Measurement types are vendor specific.

ch = *s*.addCounterInputChannel(*deviceID*, *channelID*, *measurementType*) returns the object *ch*.

[*ch, idx*] = *s*.addCounterInputChannel(*deviceID*, *channelID*, *measurementType*) returns the object *ch*, representing the channel that was added and the index, *idx*, which is an index into the array of the session object's `Channels` property.

Tips

Use `daq.createSession` to create a session object before you use this method.

Input Arguments

deviceID

Specify the vendor-defined ID of the device. The specified channel is created for this device. Obtain by calling `daq.getDevices`.

channelID

Specify the ID of the channel added to the device. You can also add a range of channels.

For NI devices, use either a terminal name, like `'ctr2'`, or a numeric equivalent like `2` for the channel ID.

measurementType

Specify a string that represents a vendor-defined measurement type. Measurement types include:

- `'EdgeCount'`
- `'PulseWidth'`
- `'Frequency'`
- `'Position'`

To see a list of all measurement types supported by a device, type `daq.getDevices` and then click on the device in the list.

Output Arguments

ch

Object representing the channel you added.

idx

An index into the array of the session object's `Channels` property.

Properties

`ActiveEdge`	Rising or falling edges of EdgeCount signals
`ActivePulse`	Active pulse measurement of PulseWidth counter channel
`CountDirection`	Specify direction of counter channel
`Device`	Channel device information
`EncoderType`	Encoding type of counter channel
`ID`	ID of channel in session
`InitialCount`	Specify initial count point
`MeasurementType`	Type counter channel measurement

`Name`	Specify descriptive name for the channel
`Terminal`	PFI terminal of counter subsystem
`ZResetCondition`	Reset condition for Z-indexing
`ZResetEnable`	Enable reset for Z-indexing
`ZResetValue`	Reset value for Z-indexing

Examples

Add a counter input EdgeCount channel:

```
s = daq.createSession ('ni')
s.addCounterInputChannel('cDAQ1Mod5','ctr0','EdgeCount')
```

Add a counter input Frequency channel. Specify output arguments to represent the channel object and the index:

```
s = daq.createSession ('ni')
[ch, idx] = s.addCounterInputChannel('cDAQ1Mod5',1,'Frequency')
```

Add counter input channels 0, 2, and 4:

```
s = daq.createSession ('ni')
s.addCounterInputChannel('cDAQ1Mod5',[0 2 4], 'EdgeCount');
```

See Also

`daq.Session.inputSingleScan` | `daq.Session.startForeground` | `daq.Session.startBackground` | `daq.Session.removeChannel` | `daq.Session`

How To

•

daq.Session.addCounterOutputChannel

Purpose

Add counter output channel

Syntax

```
s.addCounterOutputChannel(deviceID, channelID,
   measurementType)
ch = s.addCounterOutputChannel(deviceID, channelID,
   measurementType)
[ch, idx] = s.addCounterOutputChannel(deviceID, channelID,
   measurementType)
```

Description

`s.addCounterOutputChannel(deviceID, channelID, measurementType)` adds a counter output channel on the device represented by *deviceID*, with the specified *channelID*, and channel measurement type, defined by *measurementType*, on the session object, *s*. Measurement types are vendor specific.

`ch = s.addCounterOutputChannel(deviceID, channelID, measurementType)` returns the object *ch*, representing the channel that was added.

`[ch, idx] = s.addCounterOutputChannel(deviceID, channelID, measurementType)` returns the object *ch*, representing the channel that was added and the index, *idx*, which is an index into the array of the session object's `Channels` property.

Tips

Use `daq.createSession` to create a session object before you use this method.

Input Arguments

deviceID

Specify the vendor-defined ID of the device. The specified channel is created for this device. The device ID is the ID of the device that you obtain by calling `daq.getDevices`.

channelID

Specify the ID of the channel added to the device. You can also add a range of channels.

For NI devices, use either a terminal name, like `'ctr2'`, or a numeric equivalent like `2`, for the channel ID.

measurementType

Specify a string that represents a vendor-defined measurement type. A valid output measurement type is `'PulseGeneration'`.

Output Arguments

ch

Object representing the channel you added.

idx

An index into the array of the session object's `Channels` property.

Properties

`Device`	Channel device information
`DutyCycle`	Duty cycle of counter output channel
`Frequency`	Frequency of generated pulses on counter output channel
`ID`	ID of channel in session
`IdleState`	Default state of counter output channel
`InitialDelay`	Delay until output channel generates pulses
`MeasurementType`	Type counter channel measurement
`Name`	Specify descriptive name for the channel

Examples

Add an counter output PulseGeneration channel:

```
s = daq.createSession ('ni')
s.addCounterOutputChannel('cDAQ1Mod3','ctr0','PulseGeneration')
```

Add two counter output PulseGeneration channels:

```
s = daq.createSession ('ni')
s.addCounterOutputChannel('cDAQ1Mod3',0:1,'PulseGeneration')
```

See Also

daq.session.startBackground | daq.Session.addCounterInputChannel | daq.Session.removeChannel | daq.Session

How To

•

Purpose

Reset counter channel to initial count

Syntax

`s.resetCounters()`

Description

`s.resetCounters()` restarts the current value of counter channels configured in the session object, `s` to the specified `InitialCount` property on each channel.

Tips

- Reset counters only if you are performing on-demand operations using `daq.Session.inputSingleScan` or `daq.Session.outputSingleScan`
- Create an acquisition session and add a channel before you use this method. See `daq.createSession` for more information.

Examples

1. Create a session with a counter channel with an `'EdgeCount'` measurement type:

```
s = daq.createSession ('ni');
s.addCounterInputChannel('cDAQ1Mod5', 0, 'EdgeCount');
```

2. Acquire data:

```
s.inputSingleScan

ans =

   756
```

3. Reset the counter to the default value, `0`, and acquire again:

```
s.resetCounters

s.inputSingleScan

ans =

   303
```

See Also

`daq.createSession` | | | `daq.Session`

Tutorials

- "Counter Channels"

How To

-

Purpose

Add trigger connection

Syntax

```
s.addTriggerConnection(source,destination,type)
tc = s.addTriggerConnection(source,destination,type)
[tc,idx]= s.addTriggerConnection(source,destination,type)
```

Description

`s.addTriggerConnection(source,destination,type)` establishes a trigger connection from the specified source device and terminal to the specified destination device and terminal, of the specified connection type.

`tc = s.addTriggerConnection(source,destination,type)` establishes a trigger connection from the specified source and terminal to the specified destination device and terminal, of the specified connection type and displays it in the variable `tc`.

`[tc,idx]= s.addTriggerConnection(source,destination,type)` establishes a trigger connection from the specified source device and terminal to the specified destination device and terminal of the specified connection type and displays the connection in the variable `tc` and the connection index, `idx`.

Tips

- Before adding trigger connections, create a session using `daq.createSession`, and add channels to the session.

Input Arguments

source

Specify a source for the trigger connection. Valid values are:

`'external'`
: When your trigger is based on an external event.

`'`*`deviceID/terminal`*`'`
: When your trigger source is on a specific terminal on a device in your session. For example, `'Dev1/PFI1'`, for more information on device ID see `Device`. For more information on terminal see `Terminals`.

'*chassisId/terminal*'
When your trigger source is on a specific terminal on a chassis in your session, for example, `'cDAQ1/PFI1'`. For more information on terminal see `Terminals`.

You can have only one trigger source in a session.

destination

Specify a destination for the trigger connection. Valid values are:

`'external'`
When your trigger source is connected to an external device.

'*deviceID/terminal*'
When your trigger source is connected to another device in your session, for example, `'Dev1/PFI1'`. For more information on device ID see `Device`. For more information on terminal see `Terminals`.

'*chassisId/terminal*'
When your trigger source is connected to a chassis in your session, for example, `'cDAQ1/PFI1'`. For more information on terminal see `Terminals`.

You can also specify multiple destination devices as an array, for example, `{'Dev1/PFI1','Dev2/PFI1'}`.

type

Specify the trigger connection type. `StartTrigger` is the only connection type available for trigger connections at this time.

Output Arguments

tc

Object representing the trigger connection you added.

idx

An index into the array of the session object's `Connections` property.

Properties

`Destination`	Indicates trigger destination terminal
`ExternalTriggerTimeout`	Indicate if external trigger timed out
`IsWaitingForExternalTrigger`	Indicates if synchronization is waiting for an external trigger
`Source`	Indicates trigger source terminal
`Terminals`	Terminals available on device or CompactDAQ chassis
`TriggerCondition`	Specify condition that must be satisfied before trigger executes
`TriggersPerRun`	Indicate the number of times the trigger executes in an operation
`TriggersRemaining`	Indicates the number of trigger to execute in an operation
`TriggerType`	Type of trigger executed

Examples

Add External Start Trigger Connection

Create a session and add an analog input channel from `Dev1` to the session.

```
s=daq.createSession('ni')
s.addAnalogInputChannel('Dev1','ai0', 'Voltage');
```

Add a trigger connection from an external device to terminal PFI1 on Dev1 using the `'StartTrigger'` connection type:

```
s.addTriggerConnection('external','Dev1/PFI1','StartTrigger')
```

Export Trigger to External Device

To Add trigger connection going to an external destination, create a session and add an analog input channel from `Dev1` to the session.

```
s=daq.createSession('ni')
s.addAnalogInputChannel('Dev1','ai0','Voltage');
```

Add a trigger from terminal `PFI0` on `Dev1` to an external device using the `'StartTrigger'` connection type:

```
s.addTriggerConnection('Dev1/PFI1','external','StartTrigger')
```

Save Trigger Connection in Variables

To display a trigger connection in a variable, create a session and add an analog input channel from `Dev1` and `Dev2` to the session.

```
s=daq.createSession('ni')
s.addAnalogInputChannel('Dev1','ai0','Voltage');
s.addAnalogInputChannel('Dev2','ai1','Voltage');
```

Add a trigger connection from terminal `PFI1` on `Dev1` to terminal `PFI0` on `Dev2` using the `'StartTrigger'` connection type and store it in `tc`

```
tc = s.addTriggerConnection('Dev1/PFI1','Dev2/PFI0','StartTrigger');
```

See Also

`daq.createSessiondaq.Session.addClockConnection` | `daq.Session.removeConnection` |

Related Examples

-
- "Multiple-Device Synchronization"
- "Multiple-Chassis Synchronization"

Concepts

-
-

Purpose

Add clock connection

Syntax

```
s.addClockConnection(source,destination,type)
cc = s.addClockConnection(source,destination,type)
[cc,idx]= s.addClockConnection(source,destination,type)
```

Description

`s.addClockConnection(source,destination,type)` adds a clock connection from the specified source device and terminal to the specified destination device and terminal, of the specified connection type.

`cc = s.addClockConnection(source,destination,type)` adds a clock connection from the specified source device and terminal to the specified destination device and terminal, of the specified connection type and displays it in the variable `cc`.

`[cc,idx]= s.addClockConnection(source,destination,type)` adds a clock connection from the specified source device and terminal to the specified destination device and terminal, of the specified connection type and displays the connection in the variable `cc` and the connection index, `idx`.

Tips

- Before adding clock connections, create a session using `daq.createSession`, and add channels to the session.

Input Arguments

source

Specify a source for the clock connection. Valid values are:

`'external'`
: When your clock is based on an external event.

`'deviceID/terminal'`
: When your clock source is on a specific terminal on a device in your session, for example, `'Dev1/PFI1'`. For more information on device ID see `Device`. For more information on terminal see `Terminals`.

'chassisId/terminal'
: When your clock source is on a specific terminal on a chassis in your session, for example, `'cDAQ1/PFI1'`. For more information on terminal see `Terminals`.

You can have only one clock source in a session.

destination

Specify a destination for the clock connection. Valid values are:

`'external'`
: When your clock source is connected to an external device.

'deviceID/terminal'
: When your clock source is connected to another device in your session, for example, `'Dev1/PFI1'`. For more information on device ID see `Device`. For more information on terminal see `Terminals`.

'chassisId/terminal'
: When your clock source is connected to a chassis in your session, for example, `'cDAQ1/PFI1'`. For more information on terminal see `Terminals`.

You can also specify multiple destination devices as an array, for example, `{'Dev1/PFI1','Dev2/PFI1'}`.

type

Specify the clock connection type. `ScanClock` is the only connection type available for clock connections at this time.

Output Arguments

cc

Object representing the clock connection you added.

idx

An index into the array of the session object's Connections property.

Properties

`Destination`	Indicates trigger destination terminal
`Source`	Indicates trigger source terminal
`Terminals`	Terminals available on device or CompactDAQ chassis

Examples

Add External Scan Clock

Create a session and add an analog input channel from `Dev1` to the session.

```
s=daq.createSession('ni')
s.addAnalogInputChannel('Dev1','ai0', 'Voltage');
```

Add a clock connection from an external device to terminal `PFI1` on `Dev1` using the `'ScanClock'` connection type and save the connection settings to a variable:

```
cc = s.addClockConnection('external','Dev1/PFI1','ScanClock');
```

Export Scan Clock to External Device

To add clock connection going to an external destination, create a session and add an analog input channel from `Dev1` to the session.

```
s=daq.createSession('ni')
s.addAnalogInputChannel('Dev1','ai0', 'Voltage');
```

Add a clock from terminal `PFI0` on `Dev1` to an external device using the `'ScanClock'` connection type:

```
s.addClockConnection('Dev1/PFI1','external','ScanClock');
```

See Also

`daq.createSessiondaq.Session.addTriggerConnection` | `daq.Session.removeConnection` |

Related Examples

-
- "Acquire Clocked Digital Data with Imported Clock"
-
- "Acquire Clocked Digital Data with Shared Clock"
- "Acquire Digital Data Using Counter Channels"
- "Multiple-Device Synchronization"
- "Multiple-Chassis Synchronization"

Concepts

-
-

Purpose Remove clock or trigger connection

Syntax

```
s.removeConnection(idx);
```

Description `s.removeConnection(idx);` removes the specified clock or trigger with the index, `idx`, from the session. The connected device remains in the session, but no longer synchronizes with other connected devices in the session.

Input Arguments

idx

Index of the connection you want to remove.

Examples

Remove a Clock and Trigger Connection

Create clock and trigger connections in session `s`:

```
s = daq.createSession('ni');
s.addAnalogInputChannel('Dev1','ai0','Voltage')
s.addAnalogInputChannel('Dev2','ai0','Voltage')
s.addAnalogInputChannel('Dev3','ai0','Voltage')
s.addTriggerConnection('Dev1/PFI0',{'Dev2/PFI0','Dev3/PFI0'}','StartTrigger');
s.addClockConnection('Dev1/PFI1',{'Dev2/PFI1','Dev3/PFI1'},'ScanClock');
```

View existing synchronization connections in the session:

```
s.Connections
```

```
ans =


Start Trigger is provided by 'Dev1' at 'PFI0' and will be received by:
         'Dev2' at terminal 'PFI0'
         'Dev3' at terminal 'PFI0'
Scan Clock is provided by 'Dev1' at 'PFI1' and will be received by:
         'Dev2' at terminal 'PFI1'
         'Dev3' at terminal 'PFI1'
```

```
   index     Type      Source   Destination
   ----- ------------ --------- -----------
   1     StartTrigger Dev1/PFI0 Dev2/PFI0
   2     StartTrigger Dev1/PFI0 Dev3/PFI0
   3     ScanClock    Dev1/PFI1 Dev2/PFI1
   4     ScanClock    Dev1/PFI1 Dev3/PFI1
```

Remove the trigger connection with the index `2` from `Dev3/PFI0` to `Dev1/PFI0`:

```
s.removeConnection(2);
```

View updated connections:

```
s.Connections
```

```
ans =


Start Trigger is provided by 'Dev1' at 'PFI0' and will be received by 'De
Scan Clock is provided by 'Dev1' at 'PFI1' and will be received by:
         'Dev2' at terminal 'PFI1'
         'Dev3' at terminal 'PFI1'

   index     Type      Source   Destination
   ----- ------------ --------- -----------
   1     StartTrigger Dev1/PFI0 Dev2/PFI0
   2     ScanClock    Dev1/PFI1 Dev2/PFI1
   3     ScanClock    Dev1/PFI1 Dev3/PFI1
```

|

See Also

`daq.createSessiondaq.Session.addClockConnection` **|** `daq.Session.addTriggerConnection` **|**

Concepts

-
-

•

daq.Session.addDigital Channel

Purpose

Add digital channel

Syntax

```
s.addDigitalChannel(deviceID,channelID,measurementType)
ch =
s.addDigitalChannel(deviceID,channelID,measurementType)
[ch,idx] = s.addDigitalChannel(deviceID,channelID,
   measurementType)
```

Description

`s.addDigitalChannel(deviceID,channelID,measurementType)` adds a digital channel to the session, on the device represented by `deviceID`, with the specified port and single-line combination and the channel measurement type to the session, `s`.

`ch = s.addDigitalChannel(deviceID,channelID,measurementType)` creates and displays the digital channel `ch`.

`[ch,idx] = s.addDigitalChannel(deviceID,channelID, measurementType)` creates and displays the digital channel `ch` that was added and the index, `idx`, which is an index into the array of the session object's Channels property.

Note To input and output decimal values, use the conversion functions:

- `decimalToBinaryVector`
- `binaryVectorToDecimal`
- `hexToBinaryVector`
- `binaryVectorToHex`

Tips

- Create a session using `daq.createSession` before adding digital channels.

- Change the Direction property of a bidirectional channel before you read or write digital data.

Input Arguments

deviceID - Specify vendor-defined ID of device.

A channel is created for the specified device. To obtain the device ID, call `daq.getDevices`.

channelID - Specify port and line to use on this device

Add the channel with both the port and the line specified, for example, `'port0/line0'`. You can add a port and line or a range of lines on a port that you will use on this channel. To add multiple lines, use `'port0/line0:3'` or `'port1/line0, port0/line0, port2/line1'`.

measurementType - Specify string that represents channel's measurement type.

Digital channels can be:

- `InputOnly`
- `OutputOnly`
- `Bidirectional`

Output Arguments

ch - Object representing digital channels added in session.

Use this object to access properties on the channel.

Examine the channels added to the session using the

```
s = daq.createSession('ni')
[ch, idx]=s.addDigitalChannel('dev1', 'Port0/Line0:1', 'InputOnly
```

idx - An index into array of the session object's `Channels` property.

The index displays the position of the channel in the session . If you add a channel with channel ID `'port0/line0'` as the first channel in your session, the session index will be `1`. Use the index to access the `Channels` properties.

Examine the direction of two digital channels:

```
s = daq.createSession('ni')
[ch, idx]=s.addDigitalChannel('dev1', 'Port0/Line0:1', 'InputOnly')
s.Channels(idx).Direction
```

Properties

`Device`	Channel device information
`Direction`	Specify digital channel direction
`ID`	ID of channel in session
`Name`	Specify descriptive name for the channel

Examples

Add Digital Channels

Discover available digital devices on your system, create a session with digital channels.

Find all installed devices.

```
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
```

Get detailed subsystem information for NI USB-6255:

```
d(1)

ans =

ni: National Instruments USB-6255 (Device ID: 'Dev1')
   Analog input subsystem supports:
      7 ranges supported
      Rates from 0.1 to 1250000.0 scans/sec
      80 channels ('ai0' - 'ai79')
      'Voltage' measurement type

   Analog output subsystem supports:
      -5.0 to +5.0 Volts,-10 to +10 Volts ranges
      Rates from 0.1 to 2857142.9 scans/sec
      2 channels ('ao0','ao1')
      'Voltage' measurement type

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

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

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

Create a session with input, output, and bidirectional channels using `Dev1`:

```
s = daq.createSession('ni');
s.addDigitalChannel('dev1', 'Port0/Line0:1', 'InputOnly');
s.addDigitalChannel('dev1', 'Port0/Line2:3', 'OutputOnly');
```

```
s.addDigitalChannel('dev1', 'Port2/Line0:1', 'Bidirectional')

ans =

Data acquisition session using National Instruments hardware:
   Clocked operations using startForeground and startBackground are disab
   Only on-demand operations using inputSingleScan and outputSingleScan c
   Number of channels: 6
      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 OutputOnly              n/a
      4     dio  Dev1   port0/line3 OutputOnly              n/a
      5     dio  Dev1   port2/line0 Bidirectional (Unknown) n/a
      6     dio  Dev1   port2/line1 Bidirectional (Unknown) n/a
```

See Also

daq.Session.startForeground | daq.Session.startBackground | daq.Session.inputSingleScan | daq.Session.outputSingleScan | daq.createSession | decimalToBinaryVector | binaryVectorToDecimal | hexToBinaryVector | binaryVectorToHex

Related Examples

- “Acquire Non-Clocked Digital Data”
- “Generate Non-Clocked Digital Data”
- “Acquire Clocked Digital Data with Imported Clock”
- “Acquire Digital Data Using Counter Channels”
- “Acquire Clocked Digital Data with Imported Clock”

Concepts

-

Purpose

Convert decimal value to binary vector

Syntax

```
decimalToBinaryVector(decimalNumber)
decimalToBinaryVector(decimalNumber,numberOfBits)
decimalToBinaryVector(decimalNumber,numberOfBits,bitOrder)
decimalToBinaryVector(decimalNumber,[],bitOrder)
```

Description

`decimalToBinaryVector(decimalNumber)` converts a positive decimal number to a binary vector, represented using the minimum number of bits.

`decimalToBinaryVector(decimalNumber,numberOfBits)` converts a decimal number to a binary vector with the specified number of bits.

`decimalToBinaryVector(decimalNumber,numberOfBits,bitOrder)` converts a decimal number to a binary vector with the specified number of bits in the specified bit ordering.

`decimalToBinaryVector(decimalNumber,[],bitOrder)` converts a decimal number to a binary vector with default number of bits in the specified bit ordering.

Input Arguments

decimalNumber - Number to convert to binary vector

Numeric

The number to convert to a binary vector specified as a positive integer scalar.

Data Types

`single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64`

numberOfBits - Number of bits required to correctly represent the decimal number

Minimum number of bits needed to represent the value unless you specify a value (default) | Numeric

The number of bits required to correctly represent the decimal. This is an optional argument. If you do not specify the number of bits, the number is represented using the minimum number of bits needed.

bitOrder - Bit order for binary vector representation
MSBFirst (default) | LSBFirst

Bit order for the binary vector representation specified as:

- `MSBFirst` if you want the first element of the output to contain the most significant bit of the decimal number.
- `LSBFirst` if you want the first element of the output to contain the least significant bit of the decimal number.

Examples

Convert a Decimal to a Binary Vector

```
decimalToBinaryVector(6)

ans =

     1     1     0
```

Convert an Array of Decimals to a Binary Vector Array

```
decimalToBinaryVector(0:4)

ans =

     0     0     0
     0     0     1
     0     1     0
     0     1     1
     1     0     0
```

Convert a Decimal into a Binary Vector of Specific Bits

```
decimalToBinaryVector(6, 8, 'MSBFirst')

ans =
```

```
     0     0     0     0     0     1     1     0
```

Convert a Decimal into a Binary Vector with LSB First

```
decimalToBinaryVector(6, [], 'LSBFirst')

ans =

     0     1     1
```

Convert an Array of Decimals into a Binary Vector Array with LSB First

```
decimalToBinaryVector(0:4, 4, 'LSBFirst')

ans =

     0     0     0     0
     1     0     0     0
     0     1     0     0
     1     1     0     0
     0     0     1     0
```

See Also

`hexToBinaryVector``binaryVectorToDecimal``binaryVectorToHex`

Related Examples

- "Generate Signals Using Decimal Data Across Multiple Lines"

binaryVectorToDecimal

Purpose

Convert binary vector value to decimal value

Syntax

```
binaryVectorToDecimal(binaryVector)
binaryVectorToDecimal(binaryVector,bitOrder)
```

Description

`binaryVectorToDecimal(binaryVector)` converts a binary vector to a decimal.

`binaryVectorToDecimal(binaryVector,bitOrder)` converts a binary vector with the specified bit orientation to a decimal .

Input Arguments

binaryVector - Binary vector to convert to decimal
Binary Vectors

Binary vector to convert to a decimal specified as a single binary vector or a row or column-based array of binary vectors.

bitOrder - Bit order for binary vector representation
MSBFirst (default) | LSBFirst

Bit order for the binary vector representation specified as:

- `MSBFirst` if you want the first element of the output to contain the most significant bit of the decimal number.
- `LSBFirst` if you want the first element of the output to contain the least significant bit of the decimal number.

Examples

Convert Binary Vector to a Decimal Value

```
binaryVectorToDecimal([1 1 0])
```

```
ans =

     6
```

Convert a Binary Vector Array to a Decimal Value

```
binaryVectorToDecimal([1 0 0 0; 0 1 0 0])
```

```
ans =

     8
     4
```

Convert a Binary Vector with LSB First

```
binaryVectorToDecimal([1 0 0 0; 0 1 0 0],'LSBFirst')

ans =

     1
     2
```

Convert a Binary Vector Array with LSB First

```
binaryVectorToDecimal([1 1 0],'LSBFirst')

ans =

     6
```

See Also

`hexToBinaryVectordecimalToBinaryVectorbinaryVectorToHex`

Related Examples

- "Generate Signals Using Decimal Data Across Multiple Lines"

hexToBinaryVector

Purpose

Convert hexadecimal value to binary vector

Syntax

```
hexToBinaryVector(hexNumber)
hexToBinaryVector(hexNumber,numberOfBits)
hexToBinaryVector(hexNumber,numberOfBits,bitOrder)
```

Description

`hexToBinaryVector(hexNumber)` converts hexadecimal numbers to a binary vector.

`hexToBinaryVector(hexNumber,numberOfBits)` converts hexadecimal numbers to a binary vector with the specified number of bits.

`hexToBinaryVector(hexNumber,numberOfBits,bitOrder)` converts hexadecimal numbers to a binary vector with the specified number of bits in the specified bit ordering.

Input Arguments

hexNumber - Hexadecimal to convert to binary vector
Hexadecimal

Hexadecimal number to convert to a binary vector specified as a character or an array.

numberOfBits - Number of bits required to correctly represent the decimal number
Numeric

This is an optional argument. If you do not specify the number of bits, the number is represented using the minimum number of bits needed.

bitOrder - Bit order for binary vector representation
MSBFirst (default) | LSBFirst

Bit order for the binary vector representation specified as:

- `MSBFirst` if you want the first element of the output to contain the most significant bit of the decimal number.

- `LSBFirst` if you want the first element of the output to contain the least significant bit of the decimal number.

Examples

Convert a hexadecimal to a binary vector

```
hexToBinaryVector('A1')


ans =

     1     0     1     0     0     0     0     1
```

Convert a hexadecimal with a leading 0 to a binary Vector

```
hexToBinaryVector('0xA')

ans =

     1     0     1     0
```

Convert an array hexadecimal numbers to a binary vector

```
hexToBinaryVector(['A1'; 'B1'])


ans =

     1     0     1     0     0     0     0     1
     1     0     1     1     0     0     0     1
```

Convert a hexadecimal number into a binary vector of specific bits

```
hexToBinaryVector('A1',12, 'MSBFirst')

ans =

     0     0     0     0     1     0     1     0     0     0     0
```

Convert a cell array of hexadecimal numbers into a binary vector of specific bits

```
hexToBinaryVector({'A1';'B1'},8)

ans =

     1     0     1     0     0     0     0     1
     1     0     1     1     0     0     0     1
```

Convert a hexadecimal into a binary vector with LSB first

```
hexToBinaryVector('A1', [], 'LSBFirst')

ans =

     1     0     0     0     0     1     0     1
```

See Also

`decimalToBinaryVectorbinaryVectorToDecimalbinaryVectorToHex`

Related Examples

- “Acquire Digital Data in Hexadecimal Values”

Purpose

Convert binary vector value to hexadecimal

Syntax

```
binaryVectorToHex(binaryVector)
binaryVectorToHex(binaryVector,bitOrder)
```

Description

`binaryVectorToHex(binaryVector)` converts the input binary vector to a hexadecimal.

`binaryVectorToHex(binaryVector,bitOrder)` converts the input binary vector using the specified bit orientation.

Input Arguments

binaryVector - Binary vector to convert to hexadecimal

Binary Vector

The binary vector to convert to hexadecimal specified as a row vector with 0s and 1s. It can also be a column-based array of binary vectors

bitOrder - Bit order for binary vector representation

MSBFirst (default) | LSBFirst

Bit order for the binary vector representation specified as:

- `MSBFirst` if you want the first element of the output to contain the most significant bit of the decimal number.
- `LSBFirst` if you want the first element of the output to contain the least significant bit of the decimal number.

Examples

Convert a Binary Vector to a Hexadecimal

```
binaryVectorToHex([0 0 1 1 1 1 0 1])
```

```
ans =

     3D
```

Convert an Array of Binary Vectors to a Hexadecimal

```
binaryVectorToHex([1 1 0 0 0 1 0 0 ; 0 0 0 0 1 0 1 0])
```

```
ans =

    'C4'
    '0A'
```

The output is appended with 0s to make all hex values same length strings.

Convert a Binary Vector with LSB First

```
binaryVectorToHex([0 0 1 1 1 1 0 1], 'LSBFirst')

ans =

     BC
```

Convert a Binary Vector Array with LSB First

```
binaryVectorToHex([1 1 0 0 0 1 0 0 ; 0 0 0 0 1 0 1 0], 'LSBFirst')

ans =

    '23'
    '50'
```

The output is appended with 0s to make all hex values same length strings.

Note The binary vector array is converted to a cell array of hexadecimal numbers. If you input a single binary vector, it is converted to a hexadecimal string.

See Also

`hexToBinaryVectorbinaryVectorToDecimaldecimalToBinaryVector`

Related Examples

- "Acquire Digital Data in Hexadecimal Values"

Purpose

Start Support Package Installer and install support for third-party hardware or software

Syntax

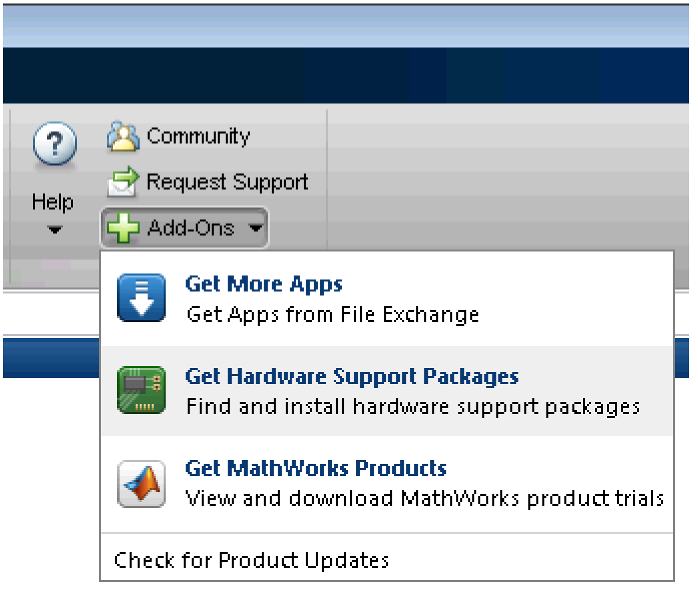
`supportPackageInstaller`

Description

The `supportPackageInstaller` function opens *Support Package Installer*. Support Package Installer can install *support packages*, which add support for specific third-party hardware or software to specific MathWorks® products. To see a list of available support packages, run Support Package Installer and advance to the second screen.

You can also start Support Package Installer in one of the following ways:

- On the MATLAB toolstrip, click **Add-Ons > Get Hardware Support Packages**.



- Double-click a support package installation file (`*.mlpkginstall`).

daq.reset

Purpose Reset MATLAB to initial state

Syntax `daq.reset`

Description `daq.reset` deletes all data acquisition objects from your MATLAB workspace and returns it to a known initial state of having no device objects and no data acquisition MEX-file or DLLs loaded in memory.

See Also `daq.createSession`

Concepts

- "Working with the Session-Based Interface"

Index