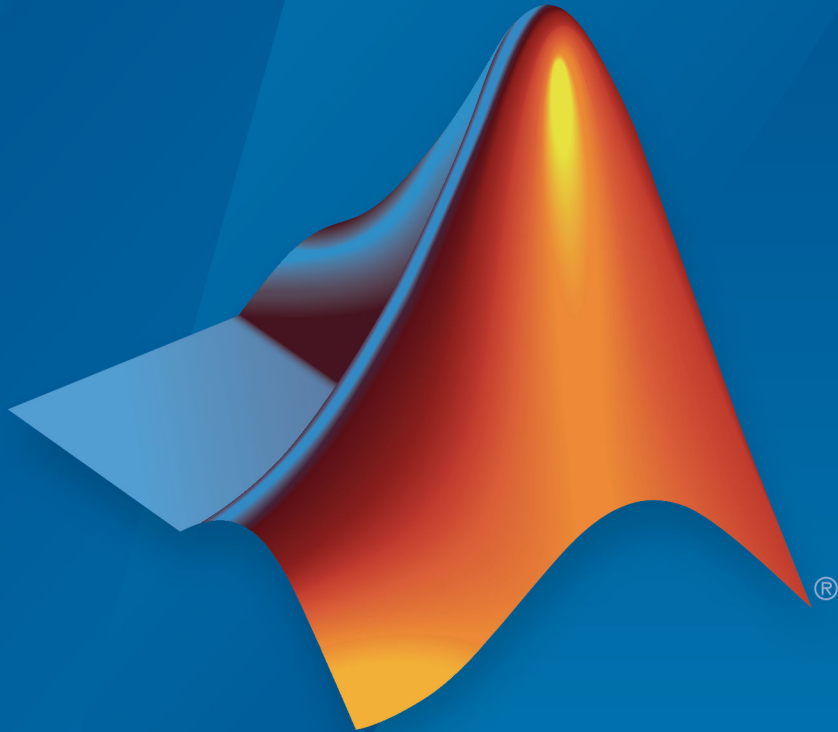


# Data Acquisition Toolbox™

Reference



# MATLAB® & SIMULINK®

R2019b



# How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



Phone: 508-647-7000



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

## *Data Acquisition Toolbox™ Reference*

© COPYRIGHT 2005–2019 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

### **Trademarks**

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

### **Patents**

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

## 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)
March 2014	Online only	Revised for Version 3.5 (Release 2014a)
October 2014	Online only	Revised for Version 3.6 (Release 2014b)
March 2015	Online only	Revised for Version 3.7 (Release 2015a)
September 2015	Online only	Revised for Version 3.8 (Release 2015b)
March 2016	Online only	Revised for Version 3.9 (Release 2016a)
September 2016	Online only	Revised for Version 3.10 (Release 2016b)
March 2017	Online only	Revised for Version 3.11 (Release 2017a)
September 2017	Online only	Revised for Version 3.12 (Release 2017b)
March 2018	Online only	Revised for Version 3.13 (Release 2018a)
September 2018	Online only	Revised for Version 3.14 (Release 2018b)
March 2019	Online only	Revised for Version 4.0 (Release 2019a)
September 2019	Online only	Revised for Version 4.0.1 (Release 2019b)



**1** | Functions – Alphabetical List

**2** | Properties – Alphabetical List

Data Acquisition Toolbox Properties ..... 2-2

**3** | Block Reference



# Functions — Alphabetical List

---

## addAnalogInputChannel

Add analog input channel

### Syntax

```
addAnalogInputChannel(s,deviceID,channelID,measurementType)
ch = addAnalogInputChannel(s,deviceID,channelID,measurementType)
[ch,idx] = addAnalogInputChannel(s,deviceID,channelID,
measurementType)
```

### Description

`addAnalogInputChannel(s,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.

- Use `daq.createSession` to create a session object before you use this method.
- To use counter channels, see `addCounterInputChannel`.

`ch = addAnalogInputChannel(s,deviceID,channelID,measurementType)` creates and returns the channel object `ch`.

`[ch,idx] = addAnalogInputChannel(s,deviceID,channelID,measurementType)` creates and 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 `Channels` property.

### Examples



## Add an Analog Input Current Channel

```
s = daq.createSession('ni')
addAnalogInputChannel(s, 'cDAQ1Mod3', 'ai0', 'Current');
```

## Add an Analog Input Channel and Return Its Index

```
s = daq.createSession('ni')
[ch, idx] = addAnalogInputChannel(s, 'cDAQ2Mod6', 'ai0', 'Thermocouple')
```

## Add a Range of Analog Input Channels

```
s = daq.createSession('ni')
ch = addAnalogInputChannel(s, 'cDAQ1Mod1', [0 2 4], 'Voltage');
```

# Input Arguments

### **s** — Data acquisition session

session object handle

Data acquisition session specified as a session object handle, created using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

For a list of relevant session object properties, see the following “Tips” on page 1-5.

### **deviceID** — Device ID

character vector or string

Device ID specified as a character vector or string, as defined by the device vendor. Obtain the device ID by calling `daq.getDevices`.

Data Types: `char` | `string`

### **channelID** — Channel ID

numeric value, character vector, or string

Channel ID specified as a numeric value, character vector, or string; or the physical location of the channel on the device. Supported values are specific to the vendor and

device. You can add multiple channels by specifying the channel ID as a numeric vector, or an array of character vectors or strings. The *index* for this channel in the session display indicates the position of this channel in the session. This channel ID is not the same as channel index in the session: if you add a channel with ID 2 as the first channel in a session, the session channel index is 1.

## **measurementType — Channel measurement type**

character vector or string

Channel measurement type specified as a character vector or string. `measurementType` represents a vendor-defined measurement type. Valid measurement types include:

- 'Voltage'
- 'Thermocouple'
- 'Current'
- 'Accelerometer'
- 'RTD'
- 'Bridge'
- 'Microphone'
- 'IEPE'

Not all devices support all types of measurement.

Data Types: `char` | `string`

## **Output Arguments**

### **ch — Analog input channel object**

1-by-n array

Analog input channel that you add, returned as an object containing a 1-by-n array of vendor-specific channel information. Use this channel object to access device and channel properties.

### **idx — Channel index**

numeric

Channel index returned as a numeric value. With this index, you can access the array of the session object `Channels` property.

## Tips

The relevant properties of the data acquisition session are:

ADCTimingMode	Set channel timing mode
BridgeMode	Specify analog input device bridge mode
Coupling	Specify input coupling mode
Device	Channel device information
ExcitationCurrent	Current of external source of excitation
ExcitationSource	External source of excitation
ExcitationVoltage	Voltage of excitation source
ExternalTriggerTimeout	Specify maximum wait time for external trigger
ID	ID of channel in session
MaxSoundPressureLevel	Sound pressure level for microphone channels
MeasurementType	Channel measurement type
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

## See Also

### Functions

`addAnalogOutputChannel` | `daq.createSession` | `inputSingleScan` |  
`removeChannel` | `startBackground` | `startForeground`

**Introduced in R2010b**

# addAnalogOutputChannel

Add analog output channel to session

## Syntax

```
addAnalogOutputChannel(s,deviceName,channelID,measurementType)
ch = addAnalogOutputChannel(s,deviceName,channelID,measurementType)
[ch,idx] = addAnalogOutputChannel(s,deviceName,channelID,
measurementType)
```

## Description

`addAnalogOutputChannel(s,deviceName,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.

- Use `daq.createSession` to create a session object before you use this method.
- To use counter channels, see `addCounterInputChannel`.

`ch = addAnalogOutputChannel(s,deviceName,channelID,measurementType)` creates and returns the channel object `ch`, representing the channel that was added.

`[ch,idx] = addAnalogOutputChannel(s,deviceName,channelID,measurementType)` creates and returns the object `ch`, representing the channel that was added, and the object `idx`, representing the index into the array of the session object `Channels` property.

## Examples

## Add an Analog Output Voltage Channel

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

## Add Analog Output Channel and Return Its Index

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

## Add a Range of Analog Output Channels

```
s = daq.createSession('ni')
ch = addAnalogOutputChannel(s, 'cDAQ1Mod8', 0:3, 'Current');
```

# Input Arguments

### **s** — Data acquisition session

session object handle

Data acquisition session specified as a session object handle, created using `daq.createSession`. Create one session per vendor, and use that vendor session to perform all data acquisition and generation operations.

For a list of relevant session object properties, see “Tips” on page 1-9.

### **deviceName** — Device ID

character vector or string

Device ID specified as a character vector or string, as defined by the device vendor. Obtain the device ID by calling `daq.getDevices`.

Data Types: `char` | `string`

### **channelID** — Channel ID

numeric value, character vector, or string

Channel ID specified as a numeric value, character vector, or string; or the physical location of the channel on the device. Supported values are specific to the vendor and

device. You can add multiple channels by specifying the channel ID as a numeric vector, or an array of character vectors or strings. The *index* for this channel indicates its position in the session display. The channel ID is not the same as the channel index in the session: if you add a channel with ID 2 as the first channel in a session, the session channel index is 1.

### **measurementType — Channel measurement type**

character vector or string

Channel measurement type specified as a character vector or string. `measurementType` represents a vendor-defined measurement type. Supported measurement types include:

- 'Voltage'
- 'Current'

Data Types: `char` | `string`

## **Output Arguments**

### **ch — Analog output channel object**

1-by-n array

Analog output channel, returned as an object containing a 1-by-n array of vendor-specific channel information. Use this channel object to access device and channel properties.

### **idx — Channel index**

numeric

Channel index, returned as a numeric value. With this index, you can access the array of the session object `Channels` property.

## **Tips**

The relevant properties of the data acquisition session are:

Device	Channel device information
ExcitationCurrent	Current of external source of excitation
ExcitationSource	External source of excitation
ExternalTriggerTimeout	Specify maximum wait time for external trigger
ID	ID of channel in session
MaxSoundPressureLevel	Sound pressure level for microphone channels
MeasurementType	Channel measurement type
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

## See Also

### Functions

`addAnalogInputChannel` | `daq.createSession` | `outputSingleScan` | `removeChannel` | `startBackground` | `startForeground`

### Introduced in R2010b



# addAudioInputChannel

Add audio input channel to session

## Syntax

```
ch = addAudioInputChannel(s,deviceName,channelID)
[ch,idx] = addAudioInputChannel(s,deviceName,channelID)
```

## Description

`ch = addAudioInputChannel(s,deviceName,channelID)` creates and displays the object `ch` representing a channel added to the session `s` using the device represented by `deviceName`, with the specified `channelID`. The channel object is stored in the variable `ch`.

---

### Tips

- Use `daq.createSession` to create a session object before you use this method.
  - To use analog channels, see `addAnalogInputChannel`.
- 

`[ch,idx] = addAudioInputChannel(s,deviceName,channelID)` additionally assigns to `idx` the index into the array of the session object's `Channels` property.

## Examples

### Add an Audio Input Channel

```
s = daq.createSession('directsound');
addAudioInputChannel(s,'Audio1',1);
```

## Add Multiple Audio Input Channels

Add two audio input channels and specify output arguments to represent the channel object and the index.

```
s = daq.createSession('directsound');  
[ch,idx] = addAudioInputChannel(s,'Audio1',1:2);
```

## Input Arguments

### **s** — Data acquisition session

session object

Data acquisition session specified as a session object created using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

### **deviceName** — Device ID

character vector or string

Device ID specified as a character vector or string, as defined by the device vendor. Obtain the device ID by calling `daq.getDevices`. The channel specified for this device is created for the session object.

Data Types: `char` | `string`

### **channelID** — Channel ID

numeric value

Channel ID, or the physical location of the channel on the device, added to the session, specified as numeric value. Supported values are specific to the vendor and device. You can also add a range of channels. The index for this channel displayed in the session indicates this channels position in the session. If you add a channel with channel ID 1 as the first channel in a session, the session index is 1.

## Output Arguments

### **ch** — Audio input channel

channel object

Audio input channel that you add, returned as a channel object containing vendor specific channel information. Use this channel object to access device and channel properties. The channel object has the following properties.

BitsPerSample	Display bits per sample
Device	Channel device information
ID	ID of channel in session
MeasurementType	Channel measurement type
Name	Specify descriptive name for the channel
Range	Specify channel measurement range
StandardSampleRates	Display standard rates of sampling
UseStandardSampleRates	Configure session to use standard sample rates

### **idx — Channel index**

numeric

Channel index returned as a numeric value. Through the index you can access the array of the session object's Channels property.

## **See Also**

### **Functions**

`addAudioOutputChannel` | `daq.createSession` | `removeChannel` | `startBackground` | `startForeground`

### **Topics**

“Hardware Discovery and Session Setup”

### **Introduced in R2014a**

## addAudioOutputChannel

Add audio output channel to session

### Syntax

```
ch = addAudioOutputChannel(s,deviceName,channelID)
[ch,idx] = addAudioOutputChannel(s,deviceName,channelID)
```

### Description

`ch = addAudioOutputChannel(s,deviceName,channelID)` creates and displays the object `ch` representing a channel added to the session `s` using the device represented by `deviceName`, with the specified `channelID`. The channel is stored in the variable `ch`.

---

#### Tips

- Use `daq.createSession` to create a session object before you use this method.
  - To use analog channels, see `addAnalogInputChannel`.
- 

`[ch,idx] = addAudioOutputChannel(s,deviceName,channelID)` additionally assigns `idx` with the index into the array of the session object's `Channels` property.

### Examples

#### Add an Audio Output Channel

Create a session and add an audio output channel to it.

```
s = daq.createSession ('directsound');
ch = addAudioOutputChannel(s, 'Audio1', 1);
```

## Add Multiple Audio Output Channels

Add several audio output channels to a session, and assign the index array.

Add two audio output channels to a session and assign output arguments to represent the channel objects and their indices.

```
s = daq.createSession ('directsound');  
[ch,idx] = addAudioOutputChannel(s,'Audio3',1:2);
```

## Input Arguments

### **s** — Data acquisition session

session object

Data acquisition session specified as a session object created using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

### **deviceName** — Device ID

character vector or string

Device ID as defined by the device vendor, specified as a character vector or string. Obtain the device ID by calling `daq.getDevices`. The channel specified for this device is created for the session object.

Data Types: `char` | `string`

### **channelID** — Channel ID

numeric value

Channel ID, or the physical location of the channel on the device, added to the session, specified as a numeric value. Supported values are specific to the vendor and device. You can also add a range of channels. The index for this channel displayed in the session indicates this channel's position in the session. If you add a channel with channel ID 1 as the first channel in a session, the session index is 1.

## Output Arguments

### **ch** — Audio output channel

channel object

Audio output channel that you add, returned as a channel object containing vendor specific channel information. Use this channel object to access device and channel properties. The channel object has the following properties.

BitsPerSample	Display bits per sample
Device	Channel device information
ID	ID of channel in session
MeasurementType	Channel measurement type
Name	Specify descriptive name for the channel
Range	Specify channel measurement range
StandardSampleRates	Display standard rates of sampling
UseStandardSampleRates	Configure session to use standard sample rates

### **idx** — Channel index

numeric

Channel index returned as a numeric value. Through the index you can access the array of the session object's Channels property.

## See Also

### Functions

`addAudioInputChannel` | `daq.createSession` | `removeChannel` | `startBackground` | `startForeground`

### Topics

“Hardware Discovery and Session Setup”

**Introduced in R2014a**

# addClockConnection

Add clock connection

## Syntax

```
addClockConnection(s,source,destination,type)
cc = addClockConnection(s,source,destination,type)
[cc,idx] = addClockConnection(s,source,destination,type)
```

## Description

`addClockConnection(s,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.

---

**Tip** Before adding clock connections, create a session using `daq.createSession`, and add channels to the session.

---

`cc = addClockConnection(s,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] = addClockConnection(s,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`.

## Examples

### Add External Scan Clock

Create a session and add an analog input channel from Dev1 to the session.

```
s = daq.createSession('ni')
addAnalogInputChannel(s, '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 = addClockConnection(s, 'external', 'Dev1/PFI1', 'ScanClock');
```

## Export Scan Clock to External Device

To add a 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')
addAnalogInputChannel(s, 'Dev1', 'ai0', 'Voltage');
```

Add a clock from terminal PFI0 on Dev1 to an external device using the 'ScanClock' connection type.

```
addClockConnection(s, 'Dev1/PFI1', 'external', 'ScanClock');
```

## Input Arguments

### **s — Data acquisition session**

session object

Data acquisition session, specified as a session object. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

### **source — Source of clock connection**

character vector or string

Source for the clock connection, specified as a character vector or string. 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.

Data Types: `char` | `string`

### **destination — Destination of clock connection**

character vector or string

Destination for the clock connection, specified as a character vector or string. 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'}`.

Data Types: `char` | `string` | `cell`

### **type — Clock connection type**

character vector or string

The clock connection type, specified as a character vector or string. `'ScanClock'` is the only connection type available for clock connections at this time.

Data Types: `char` | `string`

## Output Arguments

### **cc** — Clock connection

1-by-n object array

The added clock connection, returned as a ScanClockConnection object containing clock connection information.

### **idx** — Channel index

numeric

Channel index returned as a numeric value. Through the index you can access the array of the session object Channels property.

## See Also

`addTriggerConnection` | `daq.createSession` | `removeConnection`

## Topics

“Clock Connections”

“Synchronization”

“Import Scan Clock from External Source”

“Acquire Digital Data Using an External Clock”

“Export Scan Clock to External System”

“Acquire Digital Data Using a Shared Clock”

“Acquire Digital Data Using a Counter Output Channel as External Clock”

“Multiple-Device Synchronization Using USB or PXI Devices”

“Multiple-Chassis Synchronization with CompactDAQ Devices”

### **Introduced in R2012a**

# addCounterInputChannel

Add counter input channel

## Syntax

```
addCounterInputChannel(s,deviceID,channelID)
ch = addCounterInputChannel(s,deviceID,channelID,measurementType)
[ch,idx] = addCounterInputChannel(s,deviceID,channelID,
measurementType)
```

## Description

`addCounterInputChannel(s,deviceID,channelID)` 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 = addCounterInputChannel(s,deviceID,channelID,measurementType)` returns the object `ch`.

`[ch,idx] = addCounterInputChannel(s,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.

## Examples

### Add a Counter Input Edgecount Channel

```
s = daq.createSession('ni')
ch = addCounterInputChannel(s,'cDAQ1Mod5','ctr0','EdgeCount');
ch.Terminal % View device signal name for pin mapping.
```

## Add a Counter Input Frequency Channel

Specify output arguments to represent the channel object and the index.

```
s = daq.createSession('ni')
[ch,idx] = addCounterInputChannel(s,'cDAQ1Mod5',1,'Frequency');
ch.Terminal % View device signal name for pin mapping.
```

## Add Multiple Counter Input Channels

```
s = daq.createSession('ni')
ch = addCounterInputChannel(s,'cDAQ1Mod5',[0 1 2],'EdgeCount');
```

# Input Arguments

### **s** — Data acquisition session

session object

Data acquisition session, specified as a session object. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

### **deviceID** — Device ID

character vector or string

Device ID as defined by the device vendor, specified as a character vector or string. Obtain the device ID by calling `daq.getDevices`. The channel specified for this device is created for the session object.

Data Types: `char` | `string`

### **channelID** — Channel ID

numeric value, character vector, or string

Channel ID specified as a numeric value, character vector, or string, corresponding to the specific counter channel on the device added to the session. Channel ID 0 corresponds to the device counter 'ctr0', Channel ID 1 to 'ctr1', and so on. For the related device signal names and physical pins, see the pinout for your particular device.

You can add a range of channels by specifying the channel ID with a numeric array, or an array of character vectors or strings.

The index for a channel displayed in the session indicates the channel's position in the session. The first channel you add in a session has session index 1, and so on.

Data Types: `char` | `string` | `cell`

### **measurementType — Channel measurement type**

character vector or string

Channel measurement type, specified as a character vector or string. `measurementType` represents a vendor-defined measurement type, and can include:

- `'EdgeCount'`
- `'PulseWidth'`
- `'Frequency'`
- `'Position'`

Data Types: `char` | `string`

## **Output Arguments**

### **ch — Counter input channel object**

1-by-n array

Counter input channel that you add, returned as an object containing a 1-by-n array of vendor specific channel specific information. Use this channel object to access device and channel properties. For more information on the properties, see “Properties” on page 1-24.

### **idx — Channel index**

numeric

Channel index returned as a numeric value. Through the index you can access the array of the session object `Channels` property.

## Properties

The properties of the channel object are:

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	Channel measurement type
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

## See Also

### Functions

`addCounterOutputChannel` | `inputSingleScan` | `removeChannel` | `resetCounters`  
| `startBackground` | `startForeground`

### Properties

Terminal

### Topics

“Acquire Counter Input Data”

**Introduced in R2011a**

# addCounterOutputChannel

Add counter output channel

## Syntax

```
addCounterOutputChannel(s,deviceID,channelID)
ch = addCounterOutputChannel(s,deviceID,channelID,measurementType)
[ch,idx] = addCounterOutputChannel(s,deviceID,channelID,
measurementType)
```

## Description

`addCounterOutputChannel(s,deviceID,channelID)` 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.

---

**Tip** Use `daq.createSession` to create a session object before you use this method.

---

`ch = addCounterOutputChannel(s,deviceID,channelID,measurementType)` returns the object `ch`.

`[ch,idx] = addCounterOutputChannel(s,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.

## Examples

## Add a Counter Output PulseGeneration Channel

```
s = daq.createSession('ni');
ch = addCounterOutputChannel(s, 'cDAQ1Mod3', 'ctr0', 'PulseGeneration');
ch.Terminal % View device signal name for pin mapping.
```

## Add Two Counter Output PulseGeneration Channels

```
s = daq.createSession('ni')
ch = addCounterOutputChannel(s, 'cDAQ1Mod3', 0:1, 'PulseGeneration')
```

# Input Arguments

### **s** — Data acquisition session

session object

Data acquisition session, specified as a session object. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

### **deviceID** — Device ID

character vector

Device ID as defined by the device vendor specified as a character vector. Obtain the device ID by calling `daq.getDevices`. The channel specified for this device is created for the session object.

### **channelID** — Channel ID

numeric value, character vector, or string

Channel ID, specified as a numeric value, character vector, or string, corresponding to the specific counter channel on the device added to the session. Channel ID 0 corresponds to the device counter 'ctr0', Channel ID 1 to 'ctr1', and so on. For the related device signal names and physical pins, see the pinout for your particular device.

You can add a range of channels by specifying the channel ID with a numeric array, or an array of character vectors or strings.



The index for a channel displayed in the session indicates the channel's position in the session. The first channel you add in a session has session index 1, and so on.

Data Types: `char` | `string` | `cell`

### **measurementType — Channel measurement type**

character vector or string

Channel measurement type, specified as a character vector or string. `measurementType` represents a vendor-defined measurement type. A valid output measurement type is 'PulseGeneration'.

## **Output Arguments**

### **ch — Counter output channel object**

1-by-n array

Counter output channel that you add, returned as an object containing a 1-by-n array of vendor specific channel information. Use this channel object to access device and channel properties.

Device	Channel device information
DutyCycle	Duty cycle of output channel
Frequency	Frequency of generated output
ID	ID of channel in session
IdleState	Default state of counter output channel
InitialDelay	Delay until output channel generates pulses
MeasurementType	Channel measurement type
Name	Specify descriptive name for the channel

### **idx — Channel index**

numeric

Channel index returned as a numeric value. Through the index you can access the array of the session object's Channels property.

## See Also

### Functions

addCounterInputChannel | removeChannel | startBackground | startForeground

### Properties

Terminal

### Topics

“Generate Pulses on a Counter Output Channel”

**Introduced in R2011a**

# addDigitalChannel

Add digital channel

## Syntax

```
addDigitalChannel(s,deviceID,channelID,measurementType)
ch = addDigitalChannel(s,deviceID,channelID,measurementType)
[ch,idx] = addDigitalChannel(s,deviceID,channelID,measurementType)
```

## Description

`addDigitalChannel(s,deviceID,channelID,measurementType)` adds one or more digital channels to the session `s`, on the device represented by `deviceID`, with the specified port and single-line combination and channel measurement type.

---

### Tips

- Before adding digital channels, create a session using `daq.createSession`.
- Change the `Direction` property value of bidirectional channels before you read or write digital data.
- To input and output decimal or hexadecimal values, use these conversion functions:
  - `decimalToBinaryVector`
  - `binaryVectorToDecimal`
  - `hexToBinaryVector`
  - `binaryVectorToHex`

---

`ch = addDigitalChannel(s,deviceID,channelID,measurementType)` creates and displays the digital channels assigned to `ch`.

`[ch,idx] = addDigitalChannel(s,deviceID,channelID,measurementType)` additionally creates and displays `idx`, which is an index into the array of the session object `Channels` property.

## Examples

### Add Digital Channels

Discover available digital devices on your system, then 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 types

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');
addDigitalChannel(s,'dev1','Port0/Line0:1','InputOnly');
ch = addDigitalChannel(s,'dev1','Port0/Line2:3','OutputOnly');
[ch,idx] = addDigitalChannel(s,'dev1','Port2/Line0:1','Bidirectional')

```

ans =

```

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

## Input Arguments

### s — Data acquisition session

session object

Data acquisition session specified as a session object created using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

## **deviceID — Device ID**

character vector

Device ID as defined by the device vendor specified as a character vector. Obtain the device ID by calling `daq.getDevices`. The channel specified for this device is created for the session object.

Data Types: `char`

## **channelID — Channel ID**

character vector or string

Channel ID, or the physical location of the channel on the device, specified as a character vector or string. Supported values are specific to the vendor and device. You can add a range of channels using colon syntax, or an array of character vectors or strings. The index for this channel in the session display indicates this channel's position in the session. If you add a channel with channel ID 'Dev1' as the first channel in a session, its session index is 1.

Data Types: `cell` | `char` | `string`

## **measurementType — Channel measurement type**

character vector or string

Channel measurement type specified as a character vector or string. `measurementType` represents a vendor-defined measurement type. Supported measurements are:

- 'InputOnly'
- 'OutputOnly'
- 'Bidirectional'

Data Types: `char` | `string`

# Output Arguments

## **ch — Digital channels**

array of channel objects

Digital channels, returned as an array of channel objects. `ch` is a 1-by-n array, in which each element is a channel object with vendor-specific device and channel properties. See also the properties in “Digital Input and Output”.

**idx — Channel index**

numeric

Channel index returned as a numeric value. Use this index to access the channels in the array of the session Channels property.

## See Also

**Functions**

binaryVectorToDecimal | binaryVectorToHex | daq.createSession |  
decimalToBinaryVector | hexToBinaryVector | inputSingleScan |  
outputSingleScan | removeChannel | startBackground | startForeground

**Topics**

“Digital Subsystem Channels”

“Acquire Non-Clocked Digital Data”

“Generate Non-Clocked Digital Data”

“Acquire Digital Data Using an External Clock”

“Acquire Digital Data Using a Shared Clock”

“Acquire Digital Data Using a Counter Output Channel as External Clock”

**Introduced in R2012b**

## addFunctionGeneratorChannel

Add function generator channel

### Syntax

```
addFunctionGeneratorChannel(s, deviceID, channelID, waveformType)  
[ch, idx] = addFunctionGeneratorChannel(s, deviceID, channelID,  
waveformType)
```

### Description

Before you work with function generator channels, see “Supported Hardware” and “Create a Session”.

`addFunctionGeneratorChannel(s, deviceID, channelID, waveformType)` adds a channel on the device represented by `deviceID`, with the specified `channelID` and `waveformType` to the session `s`.

`[ch, idx] = addFunctionGeneratorChannel(s, deviceID, channelID, waveformType)` 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 `Channels` property.

### Examples

#### Add a Function Generator Channel

Add a channel on a Digilent device with a sine waveform type.

Create a session for Digilent devices.

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

Add a channel with a sine waveform type.



```
addFunctionGeneratorChannel(s, 'AD1',1, 'Sine')  
  
ans =  
  
Data acquisition sine waveform generator '1' on device 'AD1':  
  
    Phase: 0  
    Range: -5.0 to +5.0 Volts  
TerminalConfig: SingleEnded  
    Gain: 1  
    Offset: 0  
    SampleRate: 4096  
WaveformType: Sine  
    Name: ''  
    ID: '1'  
    Device: [1x1 daq.di.DeviceInfo]  
MeasurementType: 'Voltage'
```

### Save the Channel Information and the Channel Index of a Function Generator Channel

Create a session for Digilent devices.

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

Add a channel with a sine waveform type.

```
[ch,idx] = addFunctionGeneratorChannel(s, 'AD1',1, 'Sine')
```

```
ch =
```

```
Data acquisition sine waveform generator '1' on device 'AD1':  
  
    Phase: 0  
    Range: -5.0 to +5.0 Volts  
TerminalConfig: SingleEnded  
    Gain: 1  
    Offset: 0  
    SampleRate: 4096  
WaveformType: Sine  
    Name: ''  
    ID: '1'  
    Device: [1x1 daq.di.DeviceInfo]
```

MeasurementType: 'Voltage'

Properties, Methods, Events

idx =

1

## Input Arguments

### **s — Data acquisition session**

session object

Data acquisition session, specified as a session object. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

### **deviceID — Device ID**

character vector or string

Device ID as defined by the device vendor, specified as a character vector or string. Obtain the device ID by calling `daq.getDevices`. The channel specified for this device is created for the session object.

### **channelID — Channel ID**

numeric value, character array, or string

Channel ID or the physical location of the channel on the device, added to the session, specified as a numeric value, character vector, or string. You can add a range of channels with an array. The index for this channel displayed in the session indicates this channel's position in the session. If you add a channel with channel ID 1 as the first channel in a session, the session index is 1 because of position, not ID.

### **waveformType — Function generator waveform type**

character vector or string

Function generator waveform type specified as a character vector or string. Valid waveform types include:

- 'Sine'
- 'Square'
- 'Triangle'
- 'RampUp'
- 'RampDown'
- 'DC'
- 'Arbitrary'

Data Types: `char` | `string`

## Output Arguments

### **ch** — Analog input channel object

1-by-n array

Analog input channel that you add, returned as an object containing a 1xn array of vendor specific channel specific information. Use this channel object to access device and channel properties.

### **idx** — Channel index

numeric value

Channel index returned as a numeric value. Through the index you can access the array of the session object's Channels property.

## See Also

### Functions

`addAnalogInputChannel` | `daq.createSession` | `startForeground`

### Topics

“Generate a Standard Waveform Using Waveform Function Generation Channels”

“Digilent Analog Discovery Devices”

“Digilent Waveform Function Generation Channels”

“Waveform Types”

**Introduced in R2014b**

# addlistener

**Package:** daq

Create event listener

## Syntax

```
lh = addlistener(s,eventName,@callback)
lh = addlistener(s,eventName,@(src,event) expr)
```

## Description

`lh = addlistener(s,eventName,@callback)` creates a listener for the specified event, `eventName`, to execute the callback function, `callback` at the time of the event. `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.

---

**Tip** Delete the listener once the operation is complete.

```
delete(lh)
```

---

`lh = addlistener(s,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 in a file. For more information, see Anonymous Functions (MATLAB).

## Examples

## Add a Listener to an Acquisition Session

Creating a session and add an analog input channel.

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

Add a listener for the DataAvailable event.

```
lh = addlistener(s, '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.

```
startBackground(s);
```

Wait for the operation to complete, and delete the listener.

```
wait(s)  
delete(lh)
```

## Add a Listener to a Signal Generation Session Using an Anonymous Function

Create a session and set the IsContinuous property to true.

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

Add two analog output channels and create output data for the two channels.

```
addAnalogOutputChannel(s, 'cDAQ1Mod2', 0:1, 'Voltage');  
outputData0 = linspace(-1,1,1000)';  
outputData1 = linspace(-2,2,1000)';
```

Queue the output data.

```
queueOutputData(s, [outputData0 outputData1]);
```

Add a listener to call an anonymous function.

```
lh = addlistener(s, 'DataRequired', @(src,event)...  
    src.queueOutputData([outputData0 outputData1]));
```

Generate signals in the background.

```
startBackground(s);
```

Perform other MATLAB operations, and then stop the session. If the interim tasks do not allow enough time for the signal generation, use a `pause` before stopping.

```
pause(5)  
stop(s)
```

Delete the listener.

```
delete(lh)
```

## Input Arguments

### **s** — Data acquisition session

session object

Data acquisition session, specified as a session object. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

### **eventName** — Event name

'DataAvailable' | 'DataRequired' | 'ErrorOccurred'

Name of the event to listen for, specified as a character vector or string. Supported events include:

- 'DataAvailable'
- 'DataRequired'
- 'ErrorOccurred'

Data Types: char | string

### **callback** — Callback function

function handle

The callback function to execute, specified as a function handle. The function executes when the specified event occurs.

### **src — Session input argument**

variable name

Session input argument to the anonymous function, specified as a variable name. `addlistener` sends the data acquisition session object handle into the anonymous function as this variable.

### **event — Event input argument**

variable name

Event input argument to the anonymous function, specified as a variable name. `addlistener` sends the triggering event object handle into the anonymous function as this variable.

### **expr — Body of anonymous function**

executable text

Body of anonymous function, specified as a line of executable text. The expression can include the input argument variables names `src` and `event`.

## Output Arguments

### **lh — Listener event**

event object handle

The event listener returned as an event object handle. Delete the listener once the operation completes.

## See Also

### **Functions**

`addAnalogInputChannel` | `addAnalogOutputChannel` | `daq.createSession` | `startBackground`

### **Properties**

`DataAvailable` Event | `DataRequired` Event | `ErrorOccurred` Event



## **Topics**

“Session Creation Workflow”

**Introduced in R2010b**

## addTriggerConnection

Add trigger connection

### Syntax

```
addTriggerConnection(s,source,destination,type)
tc = addTriggerConnection(s,source,destination,type)
[tc,idx] = addTriggerConnection(s,source,destination,type)
```

### Description

`addTriggerConnection(s,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.

---

**Note** You cannot use triggers with audio devices.

---

---

**Tip** Before adding trigger connections, create a session using `daq.createSession`, and add channels to the session.

---

`tc = addTriggerConnection(s,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] = addTriggerConnection(s,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 in `idx`.

### Examples

## Add External Start Trigger Connection

Create a session and add an analog input channel from Dev1 to the session.

```
s = daq.createSession('ni')
addAnalogInputChannel(s, 'Dev1', 'ai0', 'Voltage');
```

Add a trigger connection from an external device to terminal PFI1 on Dev1 using the 'StartTrigger' connection type.

```
addTriggerConnection(s, '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')
addAnalogInputChannel(s, 'Dev1', 'ai0', 'Voltage');
```

Add a trigger from terminal PFI1 on Dev1 to an external device using the 'StartTrigger' connection type.

```
addTriggerConnection(s, 'Dev1/PFI1', 'external', 'StartTrigger')
```

## Save Trigger Connection

Add a trigger connection from terminal PFI1 on Dev1 to terminal PFI0 on Dev2 using the 'StartTrigger' connection type and store it in tc.

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')
addAnalogInputChannel(s, 'Dev1', 'ai0', 'Voltage');
addAnalogInputChannel(s, 'Dev2', 'ai1', 'Voltage');
```

Save the trigger connection in tc.

```
tc = addTriggerConnection(s, 'Dev1/PFI1', 'Dev2/PFI0', 'StartTrigger');
```

## Input Arguments

### **s — Data acquisition session**

session object

Data acquisition session, specified as a session object. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

### **source — Source of trigger connection**

character vector or string

Source for the trigger connection, specified as a character vector or string. Valid values are:

- `'external'` — for a trigger based on an external event. A session with an external trigger source has a timeout determined by the `ExternalTriggerTimeout` property; to disable the timeout, set the `ExternalTriggerTimeout` value to `Inf`.
- `'deviceID/terminal'` — for a trigger sourced 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'` — for a trigger sourced 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 — Destination of trigger connection**

character vector or string

Destination for the trigger connection, specified as a character vector or string. Valid values are:

- `'external'` — for a trigger source connected to an external device.
- `'deviceID/terminal'` — for a trigger source 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'` — for a trigger source 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** — Trigger connection type

character vector or string

The trigger connection type, specified as a character vector or string. `'StartTrigger'` is the only connection type available for trigger connections at this time.

## **Output Arguments**

### **tc** — Trigger connection

1-by-n object array

The trigger connection that you add, returned as an object of trigger connection information. The object contains the following properties.

Destination	Indicates trigger destination terminal
ExternalTriggerTimeout	Specify maximum wait time for external trigger
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

### **idx** — Channel index

numeric

Channel index returned as a numeric value. Through the index you can access the array of the session object Channels property.

## See Also

### Functions

`addClockConnection` | `daq.createSession` | `removeConnection`

### Topics

“Trigger Connections”

“Synchronization”

“Acquire Voltage Data Using a Digital Trigger”

“Multiple-Device Synchronization Using USB or PXI Devices”

“Multiple-Chassis Synchronization with CompactDAQ Devices”

### Introduced in R2012a

# Analog Input Recorder

Acquire and visualize analog input signals

## Description

The Analog Input Recorder provides a graphical interface to data acquisition devices.

Using this app, you can:

- Configure device channels and acquisition properties.
- Preview signals on several analog input channels for a selected device.
- Record analog input data for a finite period (foreground) or continuously (background).
- Create scripts in the Live Editor from the app session.

## Open the Analog Input Recorder App

- MATLAB Toolstrip: On the **Apps** tab, under **Test and Measurement**, click the app.
- MATLAB command prompt: Enter `analogInputRecorder`.

---

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

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

---

## See Also

**Apps**  
**Analog Output Generator**

## Topics

“Acquire Data with the Analog Input Recorder”

**Introduced in R2017b**



# Analog Output Generator

Define and generate analog output signals

## Description

The Analog Output Generator provides a graphical interface to data acquisition devices.

Using this app, you can:

- Configure device channels and properties.
- Define waveforms in a workspace variable as a vector of double values, or as a timetable.
- Preview defined signals on several analog output channels for a selected device.
- Generate analog or audio output signals for a finite period (foreground) or continuously (background).
- Create scripts in the Live Editor from the app session.

## Open the Analog Output Generator App

- MATLAB Toolstrip: On the **Apps** tab, under **Test and Measurement**, click the app.
- MATLAB command prompt: Enter `analogOutputGenerator`.

---

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

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

---

## See Also

**Apps**  
**Analog Input Recorder**

## **Topics**

“Generate Signals with the Analog Output Generator”

**Introduced in R2019a**

# binaryVectorToDecimal

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 .

## Examples

### Convert a 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
```

## 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 a character vector or string. Accepted values are:

- 'MSBFirst' — The first element of the binary vector is the most significant bit.
- 'LSBFirst' — The first element of the binary vector is the least significant bit.

Data Types: char | string

## See Also

### Functions

`binaryVectorToHex` | `decimalToBinaryVector` | `hexToBinaryVector`

### Topics

“Generate Signals Using Decimal Data Across Multiple Lines”

**Introduced in R2012b**

## binaryVectorToHex

Convert binary vector value to hexadecimal

### Syntax

```
hexval = binaryVectorToHex(binaryVector)
hexval = binaryVectorToHex(binaryVector,bitOrder)
```

### Description

`hexval = binaryVectorToHex(binaryVector)` converts the input binary vector to a hexadecimal.

`hexval = binaryVectorToHex(binaryVector,bitOrder)` converts the input binary vector using the specified bit orientation.

### Examples

#### Convert a Binary Vector to a Hexadecimal

```
hexval = binaryVectorToHex([0 0 1 1 1 1 0 1])
hexval =
    3D
```

#### Convert an Array of Binary Vectors to a Hexadecimal

```
hexval = binaryVectorToHex([1 1 0 0 0 1 0 0 ; 0 0 0 0 1 0 1 0])
hexval =
    'C4'
    '0A'
```

The output is appended with 0s to make all hex values the same length character vectors.

### Convert a Binary Vector with LSB First

```
hexval = binaryVectorToHex([0 0 1 1 1 1 0 1], 'LSBFirst')
```

```
hexval =
```

```
    BC
```

### Convert a Binary Vector Array with LSB First

```
hexval = binaryVectorToHex([1 1 0 0 0 1 0 0 ; 0 0 0 0 1 0 1 0], 'LSBFirst')
```

```
hexval =
```

```
    '23'
```

```
    '50'
```

If necessary, the output is appended with 0s to make all hex values the same length character vectors.

---

**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 character vector.

---

## Input Arguments

### **binaryVector** — Binary vector to convert to hexadecimal

numeric vector of 1s and 0s

The binary vector to convert to hexadecimal specified as a numeric vector with 0s and 1s. The vector can be a column or row vector.

### **bitOrder** — Bit order for binary vector representation

'MSBFirst' (default) | 'LSBFirst'

Bit order for the binary vector representation, specified as a character vector or string. Accepted values are:

- 'MSBFirst' — The first element of the binary vector is the most significant bit.
- 'LSBFirst' — The first element of the binary vector is the least significant bit.

Data Types: `char` | `string`

## Output Arguments

### **hexval** — Hexadecimal value

character vector

Hexadecimal value returned as a character vector.

## See Also

### **Functions**

`binaryVectorToDecimal` | `decimalToBinaryVector` | `hexToBinaryVector`

### **Topics**

“Acquire Digital Data in Hexadecimal Values”

**Introduced in R2012b**



# daq.createSession

Create data acquisition session for specific vendor hardware

## Syntax

```
session = daq.createSession(vendor)
```

## Description

`session = daq.createSession(vendor)` creates a session object for configuring and operating data acquisition devices from the specified vendor.

## Examples

### Create Data Acquisition Session for National Instruments Devices

Create a data acquisition session object `s`, for National Instruments® devices.

```
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.
```

## Input Arguments

### **vendor** — Vendor name

character vector or string

Vendor name for the device you want to create a session for, specified as a character vector. Valid vendors are:

- 'ni'
- 'diligent'
- 'directsound'
- 'adi'
- 'mcc'

Data Types: char | string

## Output Arguments

### **session — Data acquisition session**

session object

Data acquisition session, returned as a session object. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

The session has the following properties:

Channels	Array of channel objects associated with session object
Connections	Array of connections in session
DurationInSeconds	Specify duration of acquisition
IsContinuous	Specify if operation continues until manually stopped
IsDone	Indicate if session operation is complete
IsLogging	Indicate if hardware is acquiring or generating data
IsNotifyWhenDataAvailableExceedsAuto	Control if <code>NotifyWhenDataAvailableExceeds</code> is set automatically
IsNotifyWhenScansQueuedBelowAuto	Control if <code>NotifyWhenScansQueuedBelow</code> is set automatically
NotifyWhenDataAvailableExceeds	Control firing of <code>DataAvailable</code> event
NotifyWhenScansQueuedBelow	Control firing of <code>DataRequired</code> 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
UserData	Custom data
Vendor	Vendor information associated with session object

## See Also

### Functions

addAnalogInputChannel | addAnalogOutputChannel | addAudioInputChannel |  
addAudioOutputChannel | addCounterInputChannel |  
addCounterOutputChannel | addDigitalChannel | daq.getDevices |  
daq.getVendors

### Topics

“Data Acquisition Toolbox Properties” on page 2-2

### Introduced in R2010b

# daq.getDevices

Display available data acquisition devices

## Syntax

```
daq.getDevices  
device = daq.getDevices
```

## Description

daq.getDevices lists devices available to your system.

---

### Tips

- Devices not supported by the toolbox are denoted in the output list with an asterisk (\*). For a complete list of supported devices, see <https://www.mathworks.com/hardware-support/data-acquisition-software.html>.
  - To suppress diagnostic information from daq.getDevices about inoperational vendors, run the function disableVendorDiagnostics. To turn these diagnostics back on, run enableVendorDiagnostics.
- 

device = daq.getDevices assigns the device list to the variable device.

## Examples

### Get a List of Devices

Get a list of all devices available to your system and store it in the variable d.

```
d = daq.getDevices
```

```
d =
```

```
index  Vendor      Device ID      Description
-----
1      directsound Audio0      DirectSound Primary Sound Capture Driver
2      directsound Audio1      DirectSound Digital Audio (S/PDIF) (High Definition Audio Device)
3      directsound Audio3      DirectSound HP 4120 (2- HP 4120)
4      ni              cDAQ1Mod1     National Instruments NI 9205
5      ni              cDAQ1Mod2     National Instruments NI 9263
6      ni              cDAQ1Mod3     National Instruments NI 9234
7      ni              cDAQ2Mod1     National Instruments NI 9402
8      ni              cDAQ2Mod2     National Instruments NI 9205
9      ni              cDAQ2Mod3     National Instruments NI 9375
10     ni              Dev1          National Instruments USB-6211
11     ni              Dev2          National Instruments USB-6218
12     ni              Dev3          National Instruments PCI-6255
13     ni              PXI1Slot2     National Instruments PXI-4461
14     ni              PXI1Slot3     National Instruments PXI-4461
```

To get detailed information about a particular device or a module in a chassis, type `d(index)`. For example, to get information about the NI 9402, which has the index 7, type:

```
d(7)
ans =
ni: National Instruments NI 9402 (Device ID: 'cDAQ2Mod1')
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 1 of the 'cDAQ-9178' chassis with the name 'cDAQ2'.
```

You can also click on the name of the device in the list to access detailed device information, which includes:

- subsystem type
- rate
- number of available channels
- measurement type

## Output Arguments

### device — Device list

array of `DeviceInfo` objects

Device list, returned as an array of DeviceInfo objects.

## See Also

### Functions

daq.createSession | daq.getVendors

### Topics

“Hardware Discovery and Session Setup”

**Introduced in R2010b**

## daq.getVendors

Display available vendors

### Syntax

```
daq.getVendors
vendor = daq.getVendors
```

### Description

daq.getVendors lists vendors available to your machine and MATLAB.

vendor = daq.getVendors assigns the output list to the variable vendor.

### Examples

#### Get the List of Available Vendors

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: 5
```

index	ID	Operational	Comment
1	ni	true	National Instruments
2	adi	true	Analog Devices Inc.
3	directsound	true	DirectSound
4	digilent	true	Digilent Inc.
5	mcc	true	Measurement Computing Corp.

Programmatically determine if 'adi' is an operational vendor.

```
for idx = 1:length(v)
    if strcmp(v(idx).ID,'adi')
```



```
        v(idx).IsOperational
    end
end
ans =
    logical
    1
```

## Output Arguments

### **vendor** — Vendor list

array of VendorInfo objects

Vendor list, returned as an array of VendorInfo objects. This represents the vendor information available to your system.

For a list of vendors currently supported by Data Acquisition Toolbox, and instructions for installing necessary support packages, see “Data Acquisition Toolbox Supported Hardware”. For information on using the SDK to extend support for other vendors, see “Build Custom Adaptors”.

## See Also

### **Functions**

daq.createSession | daq.getDevices

### **Topics**

“Hardware Discovery and Session Setup”

“Data Acquisition Toolbox Supported Hardware”

“Build Custom Adaptors”

### **Introduced in R2010b**

## daqhelp

Help for toolbox interface

### Syntax

```
daqhelp
daqhelp('functionname')
helptext = daqhelp('functionname')
```

### Description

`daqhelp` displays a comprehensive listing of Data Acquisition Toolbox functions along with a brief description of each. Links in the output provide access to more detailed information.

`daqhelp('functionname')` returns help for the function specified as a character vector or string.

`helptext = daqhelp('functionname')` assigns the help text output to the variable `out`.

### Examples

#### Get Toolbox Help

Get overview help for Data Acquisition Toolbox.

```
daqhelp
```

```
Data Acquisition Toolbox
Version 3.12 (R2018a Prerelease) 08-Aug-2017
Data Acquisition Toolbox Session-Based Interface Support
```

```
To control these data acquisition devices, you use a daq.Session object
to configure and control one or more devices.
```

In a typical workflow,

- (1) Discover hardware devices using `daq.getDevices`
- (2) Create a daq Session using `daq.createSession`
- (3) Add device channels
- (4) Add device connections
- (5) Set session and channel properties
- (6) Perform on demand operations using `inputSingleScan/outputSingleScan`
- (7) Perform clocked operations using `startForeground/startBackground`

(1) Device enumeration and discovery:

- `daq.getDevices` - Show data acquisition devices available
- `daq.getVendors` - Show known data acquisition vendors
- `daq.reset` - Reinitialize all data acquisition devices and sessions.

This represents only a partial view of the total output.

## Get Function Help

Get help for a specified function.

```
daqhelp('daq.createSession')
```

```
daq.createSession Create a data acquisition session for a given vendor
Returns a daq.Session object that represents a session with hardware
from the specific vendor.
```

```
SESSION = daq.createSession(VENDOR_ID) returns a session object specific
to the vendor ID specified by the string VENDOR_ID that you can configure to
perform operations. Type daq.getVendors() for a list of available
vendors.
```

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

See also `daq.getDevices`, `daq.getVendors`, `daq.reset`

## Return Function Help Text to Variable

Get help for a specified function, assigning the help text to a variable.

```
helptext = daqhelp('daq.createSession');
```

## Input Arguments

**'functionname'** — Function for which you want help

char vector or string

Function for which you want help, specified as a character vector or string.

Example: 'daq.createSession'

Data Types: char | string

## Output Arguments

**helptext** — Help text

char vector

Help text, returned as a character vector.

**Introduced before R2006a**

# daqreset

Reset Data Acquisition Toolbox

## Syntax

```
daqreset
```

## Description

daqreset resets Data Acquisition Toolbox and deletes all data acquisition session and device objects.

## Examples

### Reset the Toolbox

Create a data acquisition session, then reset the toolbox.

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

```
s =
```

```
handle to deleted Session
```

## See Also

### Functions

daq.createSession

**Introduced before R2006a**

## decimalToBinaryVector

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.

### 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
```

## 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

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. By default minimum number of bits needed to represent the value is specified, unless you specify a value

### **bitOrder** — Bit order for binary vector representation

'MSBFirst' (default) | 'LSBFirst'

Bit order for the binary vector representation, specified as a character vector or string. Accepted values are:

- 'MSBFirst' — The first element of the binary vector is the most significant bit.
- 'LSBFirst' — The first element of the binary vector is the least significant bit.

Data Types: `char` | `string`

## See Also

### Functions

`binaryVectorToDecimal` | `binaryVectorToHex` | `hexToBinaryVector`



## **Topics**

“Generate Signals Using Decimal Data Across Multiple Lines”

**Introduced in R2012b**

## hexToBinaryVector

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.

### 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 of 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 1
```

### 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

## Input Arguments

### **hexNumber** — Hexadecimal to convert to binary vector

hexadecimal value

Hexadecimal number to convert to a binary vector, specified as a character vector or string.

Data Types: `char` | `string`

### **numberOfBits** — Number of bits to represent the decimal number

numeric

Number of bits to represent the decimal number, specified as a numeric value. 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 a character vector or string. Accepted values are:

- 'MSBFirst' — The first element of the binary vector is the most significant bit.
- 'LSBFirst' — The first element of the binary vector is the least significant bit.

Data Types: `char` | `string`

## See Also

### Functions

`binaryVectorToDecimal` | `binaryVectorToHex` | `decimalToBinaryVector`

### Topics

“Acquire Digital Data in Hexadecimal Values”

**Introduced in R2012b**

# inputSingleScan

Acquire single scan from all input channels

## Syntax

```
data = inputSingleScan(s);  
[data,triggerTime] = inputSingleScan(s);
```

## Description

`data = inputSingleScan(s)`; returns an immediately acquired single scan from each input channel in the session as a 1-by-n array of doubles. The value is stored in `data`, where `n` is the number of input channels in the session.

---

**Tip** To acquire more than a single scan, use `startForeground`.

---

`[data,triggerTime] = inputSingleScan(s)`; returns an immediately acquired single scan from each input channel in the session as a 1-by-n array of doubles. The value is stored in `data`, where `n` is the number of input channels in the session and the MATLAB serial date timestamp representing the time the data is acquired is returned in `triggerTime`.

## Examples

### Acquire Single Analog Input Scan

Acquire a single input from an analog channel.

Create a session and add two analog input channels:

```
s = daq.createSession('ni');  
addAnalogInputChannel(s,'cDAQ1Mod1',1:2,'Voltage');
```

Input a single scan:

```
data = inputSingleScan(s)
data =
    -0.1495    0.8643
```

### Acquire Single Digital Input Scan

Acquire a single input from a digital channel and get data and the trigger time of the acquisition.

Create a session and add two digital channels with `InputOnly` measurement type:

```
s = daq.createSession('ni');
addDigitalChannel(s, 'dev1', 'Port0/Line0:1', 'InputOnly');
```

Input a single scan:

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

### Acquire Single Counter Input Scan

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');
addCounterInputChannel(s, 'Dev1', 0, 'EdgeCount');
```

Input a single edge count:

```
data = inputSingleScan(s)
```

## Input Arguments

**s** — Data acquisition session  
session object

Data acquisition session, specified as a session object. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

## Output Arguments

### **data** — Value from acquired data

array of double

Value from acquired data, returned as a 1-by-n array of doubles.

### **triggerTime** — Timestamp of acquired data

numeric

Timestamp of acquired data which is a MATLAB serial date timestamp representing the absolute time when `timeStamps = 0`.

## See Also

### Functions

`addAnalogInputChannel` | `addCounterInputChannel` | `addDigitalChannel` | `daq.createSession` | `startForeground`

### Topics

“Acquire Non-Clocked Digital Data”

“Acquire Counter Input Data”

**Introduced in R2010b**



# outputSingleScan

Generate single scan on all output channels

## Syntax

```
outputSingleScan(s,data)
```

## Description

`outputSingleScan(s,data)` outputs a single scan of data on one or more analog output channels.

## 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');  
addAnalogOutputChannel(s,'cDAQ1Mod2',0:1, 'Voltage');
```

Create an output value and output a single scan for each channel added.

```
outputSingleScan(s,[1.5 4]);
```

### Digital Output

Output one value on each of two lines on a digital channel

Create a session and add two digital channels from port 0 that measures output only:

```
s = daq.createSession('ni');  
addDigitalChannel(s, 'dev1', 'Port0/Line0:1', 'OutputOnly')
```

Output one value each on the two lines:

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

## Input Arguments

### **s — Data acquisition session**

session object

Data acquisition session, specified as a session object. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

### **data — Data to output**

doubles

Data to output, represented as a 1-by-n matrix of doubles, where n is the number of output channels in the session.

## See Also

### **Functions**

`addAnalogOutputChannel` | `addDigitalChannel` | `daq.createSession` | `inputSingleScan`

**Introduced in R2010b**

# prepare

Prepare session for operation

## Syntax

```
prepare(s)
```

## Description

`prepare(s)` configures and allocates hardware resources for the session `s` and reduces the latency of `startBackground` and `startForeground` functions. There must be at least one channel in the session before you can call this function. Use of this function is optional; it is automatically called as needed.

## Examples

### Prepare Session

Create a session with one channel, and prepare it for operation.

```
s = daq.createSession('directsound');  
ch = addAudioInputChannel(s, 'Audio1', 1);  
prepare(s)
```

## Input Arguments

### **s** — Data acquisition session

session object

Data acquisition session, specified as a session object. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

## See Also

### Functions

`daq.createSession` | `release` | `startBackground` | `startForeground`

**Introduced in R2010b**

# queueOutputData

Queue data to be output

## Syntax

```
queueOutputData(s,data)
```

## Description

`queueOutputData(s,data)` queues data to be output. When generating output signals, you must queue data before you call `startForeground` or `startBackground`.

## Examples

### Queue Output Data for a Single Channel

Create a session, add an analog output channel, and queue some data to output.

```
s = daq.createSession('ni');  
addAnalogOutputChannel(s,'cDAQ1Mod2','ao0','Voltage');  
queueOutputData(s,linspace(-1,1,1000)');  
startForeground(s)
```

### Queue Output Data for Multiple Channels

```
s = daq.createSession('ni');  
addAnalogOutputChannel(s,'cDAQ1Mod2',0:1,'Voltage');  
data0 = linspace(-1,1,1000)';  
data1 = linspace(-2,2,1000)';
```

```
queueOutputData(s,[data0 data1]);  
startBackground(s);
```

## Input Arguments

### **s — Data acquisition session**

session object handle

Data acquisition session, specified as a session object handle. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

### **data — Output data values**

array of doubles

Output data values, specified as an  $m$ -by- $n$  matrix of doubles, where  $m$  is the number of scans to generate, and  $n$  is the number of output channels in the session.

## See Also

### Functions

`addAnalogOutputChannel` | `addDigitalChannel` | `daq.createSession` | `startBackground` | `startForeground`

**Introduced in R2010b**

# release

Release session hardware resources

## Syntax

```
release(s)
```

## Description

`release(s)` releases all reserved hardware resources in the session `s`, and flushes any data you have queued in the hardware in that session.

A session might reserve exclusive access to the hardware associated with it. If you need to use the hardware in another session or by applications other than MATLAB, use `release(s)` to unreserve the hardware and clear its data.

Hardware resources associated with a session are automatically released when you delete the session object or assign a different value to the variable containing the session object.

## Examples

### Release Session Hardware

Create a session and add an analog input voltage channel and acquire data in the foreground:

```
s1 = daq.createSession('ni');  
addAnalogInputChannel(s1, 'cDAQ3Mod1', 'ai0', 'Voltage');  
startForeground(s1)
```

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:

```
release(s1);  
s2 = daq.createSession('ni');
```

```
addAnalogInputChannel(s2, 'cDAQ3Mod1', 'ai2', 'Voltage');  
startForeground(s2);
```

## Input Arguments

### **s** — Data acquisition session

session object

Data acquisition session, specified as a session object. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

## See Also

### Functions

`daq.createSession` | `prepare` | `startBackground` | `startForeground`

**Introduced in R2010b**



# removeChannel

Remove channel from session object

## Syntax

```
removeChannel(s,idx);
```

## Description

`removeChannel(s,idx)`; removes the channel specified by `idx` from the session object `s`.

## Examples

### Remove Channels from a Session

Start with a session `s`, to which you add two analog input and two analog output voltage channels and display the 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	MeasurementType	Range	Name
1	ai	cDAQ1Mod4	ai0	Voltage (SingleEnd)	-10 to +10 Volts	
2	ai	cDAQ1Mod4	ai1	Voltage (SingleEnd)	-10 to +10 Volts	
3	ao	cDAQ1Mod2	ao0	Voltage (Diff)	-10 to +10 Volts	
4	ao	cDAQ1Mod2	ao1	Voltage (Diff)	-10 to +10 Volts	

Remove channel 'ai0' with the index 1 from the session:

```
removeChannel(s,1)
```

To see how the indices 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 MeasurementType Range Name
-----
  1     ai  cDAQ1Mod4 ai1    Voltage (SingleEnd) -10 to +10 Volts
  2     ao  cDAQ1Mod2 ao0    Voltage (Diff)      -10 to +10 Volts
  3     ao  cDAQ1Mod2 ao1    Voltage (Diff)      -10 to +10 Volts
```

Remove the first output channel 'ao0' at index 2:

```
removeChannel(s,2);
```

The session now displays one input and one output channel:

`s.Channels`

`ans =`

```
Number of channels: 2
  index Type Device Channel MeasurementType Range Name
-----
  1     ai  cDAQ1Mod4 ai1    Voltage (SingleEnd) -10 to +10 Volts
  2     ao  cDAQ1Mod2 ao1    Voltage (Diff)      -10 to +10 Volts
```

## Input Arguments

### **s** — Data acquisition session

session object

Data acquisition session, specified as a session object. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

### **idx** — Index of channel

numeric

Channel index, specified as a numeric value. Use the index of the channel that you want to remove from the session.

## See Also

### Functions

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

**Introduced in R2010b**

## removeConnection

Remove clock or trigger connection

### Syntax

```
removeConnection(s,idx)
```

### Description

`removeConnection(s,idx)` removes the specified clock or trigger with the index `idx`, from the session. The connected device remains in the session, but is no longer synchronized with other connected devices in the session.

### Examples

#### Remove a Clock and Trigger Connection

Create clock and trigger connection in the session `s`.

```
s = daq.createSeion('ni');  
addAnalogInputChannel(s,'Dev1','ai0','Voltage')  
addAnalogInputChannel(s,'Dev2','ai0','Voltage')  
addAnalogInputChannel('Dev3','ai0','Voltage')  
addTriggerConnection(s,'Dev1/PFI0',{ 'Dev2/PFI0', 'Dev3/PFI0' }, 'StartTrigger');  
addClockConnection(s,'Dev1/PFI1',{ 'Dev2/PFI1', 'Dev3/PFI1' }, 'ScanClock');
```

View existing synchronization connection .

```
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	Deination
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:

```
removeConnection(s,2);
```

View updated connection

```
s.Connections
```

```
an=
```

Start Trigger is provided by 'Dev1' at 'PFI0' and will be received by  
 'Dev2' 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	Deination
1	StartTrigger	Dev1/PFI0	Dev2/PFI0
2	ScanClock	Dev1/PFI1	Dev2/PFI1
3	ScanClock	Dev1/PFI1	Dev3/PFI1

Notice that the connections are re-indexed.

## Input Arguments

### **s** — Data acquisition session

session object

Data acquisition session, specified as a session object. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation

operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

**idx — Index of connection**

numeric value

Index of the connection you want to remove, specified as a numeric value.

## See Also

### Functions

`addClockConnection` | `addTriggerConnection` | `daq.createSession`

### Topics

“Trigger Connections”

“Clock Connections”

“Synchronization”

**Introduced in R2012a**

# resetCounters

Reset counter channel to initial count

## Syntax

```
resetCounters(s)
```

## Description

`resetCounters(s)` resets the current value of counter channels configured in the session object, `s`, to the value specified by the `InitialCount` property on each channel.

---

### Tips

- Reset counters only if you are performing on-demand operations using `inputSingleScan` or `outputSingleScan`.
  - Create an acquisition session and add a channel before you use this function. See `daq.createSession` for more information.
- 

## Examples

### Reset Counters

Create a session, then add a counter channel with an `EdgeCount` measurement type and acquire data.

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

```
ans =
```

```
756
```

Reset the counter to the default value, 0, and acquire data again.

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

```
ans =
```

```
    303
```

## Input Arguments

### **s** — Data acquisition session

session object

Data acquisition session, specified as a session object. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

## See Also

### Functions

`addCounterInputChannel` | `daq.createSession` | `inputSingleScan`

### Topics

“Acquire Counter Input Data”

“Generate Pulse Data on a Counter Channel”

**Introduced in R2011a**



# startBackground

Start background operations

## Syntax

```
startBackground(s);
```

## Description

`startBackground(s)`; starts the operation of the session object, `s`, without blocking MATLAB command line and other code. To block MATLAB execution, use `startForeground`.

When you use `startBackground(s)` 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 “Event and Listener Concepts” (MATLAB).

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 `stop` to stop background operations explicitly.

Use `wait` to block MATLAB execution until a background operation is complete.

---

## Tips

- Create an acquisition session and add a channel before you use this method. See `daq.createSession` for more information.
- 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.
  - Call `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

Create a session and add a listener. Use the listener callback function to access the acquired data.

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

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

Start the session and perform other MATLAB operations.

```
startBackground(s);
```

Perform other MATLAB operations.

### Generate Data Continuously

For a continuous background generation, add a listener event to queue additional data to be output.

```
s = daq.createSession('ni');  
addAnalogOutputChannel(s, 'cDAQ1Mod2', 0, 'Voltage');  
s.IsContinuous = true;  
s.Rate=10000;  
data=linspace(-1,1,5000)';  
lh = addlistener(s, 'DataRequired', ...
```

```
@(src,event) src.queueOutputData(data);  
queueOutputData(s,data)  
startBackground(s);
```

Perform other MATLAB operations during the generation.

## Input Arguments

### **s** — Data acquisition session

session object

Data acquisition session, specified as a session object. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

## See Also

`DataAvailable` | `DataRequired` | `ErrorOccurred` | `addAnalogInputChannel` | `addAnalogOutputChannel` | `addAudioInputChannel` | `addDigitalChannel` | `addListener` | `daq.createSession` | `queueOutputData` | `startForeground`

## Topics

“Acquire Data in the Background”

“Generate Signals in the Background”

“Generate Signals in the Background Continuously”

## Introduced in R2010b

## startForeground

Start foreground operations

### Syntax

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

### Description

`startForeground(s)`; starts operations of the session object, `s`, and blocks MATLAB command line and other code until the session operation is complete.

`data = startForeground(s)`; returns the data acquired in the output parameter, `data`.

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

### Examples

#### Acquire Analog Data

Acquire data by creating a session with an analog input channel.

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

Start the acquisition and save the acquired data into the variable `data`:

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

## Generate Analog Data

Generate a signal by creating a session with an analog output channel.

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

Create and queue an output signal and start the generation:

```
outputSignal = linspace(-1,1,1000)';  
queueOutputData(s,outputSignal);  
startForeground(s);
```

## Acquire Analog Input Data and Timestamps

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

Start the acquisition and save the acquired data in the variable `data`, the acquisition timestamp in `timestamps`, and the trigger time in `triggerTime`:

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

# Input Arguments

### **s** — Data acquisition session

session object

Data acquisition session, specified as a session object. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

# Output Arguments

### **data** — Values of acquired data

array of doubles

Values of acquired data, returned as an m-by-n array of doubles, where m is the number of scans acquired, and n is the number of input channels in the session.

### **timeStamps — Recorded timestamp**

numeric

Recorded timestamp relative to the time the operation is triggered, returned as an m-by-1 array, where m is the number of scans.

### **triggerTime — Timestamp of acquired data**

numeric

Timestamp of acquired data which is a MATLAB serial date timestamp representing the absolute time when `timeStamps = 0`.

## See Also

`addAnalogInputChannel` | `addAnalogOutputChannel` | `addDigitalChannel` | `daq.createSession` | `startBackground`

## Topics

“Acquire Data in the Foreground”

“Generate Pulse Data on a Counter Channel”

“Hardware Discovery and Session Setup”

**Introduced in R2010b**

# stop

Stop background operation

## Syntax

```
stop(s);
```

## Description

`stop(s)`; stops the session and all associated hardware operations in progress. Stopping the session flushes all undelivered data that is below the threshold defined by the property `NotifyWhenDataAvailableExceeds`, and will not fire any more `DataAvailable` events.

## Examples

### Stop Background Signal Generation

Create a continuous signal in background mode, and generate output until you explicitly stop it.

Generate output data.

```
s = daq.createSession('ni');  
addAnalogOutputChannel(s, 'cDAQ1Mod2', 0, 'Voltage');  
s.IsContinuous = true;  
s.Rate = 10000;  
data = linspace(-1, 1, 5000)';  
lh = addlistener(s, 'DataRequired', ...  
    @(src, event) src.queueOutputData(data));  
queueOutputData(s, data)  
startBackground(s);
```

Perform other MATLAB operations during signal generation, then stop the session when you no longer need the signal.

```
stop(s);
```

## Input Arguments

**s — Data acquisition session**

session object

Data acquisition session, specified as a session object. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

## See Also

### Functions

`startBackground` | `startForeground` | `wait`

**Introduced in R2010b**



---

## wait

Block MATLAB until background operation completes

### Syntax

```
wait(s)
wait(s,timeout)
```

### Description

`wait(s)` blocks MATLAB until the background operation completes. To abort the wait, press **Ctrl+C**.

---

**Tips** You cannot call `wait` if you have set the session `IsContinuous` property to `true`. To terminate the operation in this case, use the `stop` function.

---

`wait(s,timeout)` blocks MATLAB until the operation completes or the specified timeout occurs. If the session operation does not complete before this timeout occurs, MATLAB is unblocked, an error is thrown, and the data acquisition session operation continues running.

## Examples

### Wait for Session to Complete Data

Create a session and add an analog output channel.

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

Queue some output data.

```
queueOutputData(s, zeros(10000,1));
```

Start the session, then issue a `wait`. This blocks MATLAB until all data is output.

```
startBackground(s);  
% Perform other MATLAB operations.  
wait(s)
```

Queue more data and wait for up to 15 seconds.

```
queueOutputData(s, zeros(10000,1));  
startBackground(s);  
% Perform other MATLAB operations.  
wait(s,15)
```

## Input Arguments

### **s** — Data acquisition session

session object

Data acquisition session, specified as a session object. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

### **timeout** — Session timeout value

numeric

Session timeout value in seconds, specified as a numeric value. This value is the maximum time in seconds to wait.

## See Also

### Functions

`startBackground` | `stop`

**Introduced in R2010b**

# DataAvailable

Notify when acquired data is available to process

## Syntax

```
lh = addlistener(session,'DataAvailable',callbackfct);  
lh = addlistener(session,'DataAvailable',@(src,event) expr)
```

## Description

`lh = addlistener(session,'DataAvailable',callbackfct);` creates a listener for the `DataAvailable` event. When data is available to process, the callback executes. The callback can be any MATLAB function with the `(src,event)` signature.

---

**Tip** The frequency with which the `DataAvailable` event is fired, is controlled by `NotifyWhenDataAvailableExceeds`

---

`lh = addlistener(session,'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 in a separate file. For more information see [Anonymous Functions \(MATLAB\)](#).

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  $m$ -by- $n$  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  $m$ -by-1 array where  $m$  is the number of scans acquired.

### TriggerTime

A MATLAB serial date time stamp representing the absolute time the acquisition trigger occurs.

## Examples

### Create DataAvailable Function

This example shows how to create an event that triggers a callback function to plot data.

Create a session, add an analog input channel, and change the duration of the acquisition.

```
s = daq.createSession('ni');  
addAnalogInputChannel(s, 'cDAQ1Mod1', 'ai0', 'Voltage');  
s.DurationInSeconds = 5;
```

Add a listener for the `DataAvailable` event to trigger the plotting callback.

```
lh = addlistener(s, '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.

```
startBackground(s);  
wait(s)
```

Delete the listener.

```
delete(lh)
```

## Create Anonymous DataAvailable Function

This example shows how to create an event using an anonymous function call to plot data when an event occurs.

Create a session, add an analog input channel, and change the duration of the acquisition.

```
s = daq.createSession('ni');  
addAnalogInputChannel(s, 'cDAQ1Mod1', 'ai0', 'Voltage');  
s.DurationInSeconds = 5;
```

Add a listen with an anonymous function call.

```
lh = s.addlistener('DataAvailable', ...  
    @(src,event) plot(event.TimeStamps, event.Data));
```

Acquire data.

```
s.startBackground();
```

Delete the listener.

```
delete(lh)
```

## Input Arguments

### **session** — Data acquisition session

session object

Data acquisition session, specified as a session object. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

### **callbackfct** — Callback function

function handle

Callback function, specified as a function handle.

### **expr** — Anonymous callback function

MATLAB operation

Anonymous callback function, specified as a MATLAB operation. The expression executes when the trigger occurs.

## See Also

### Functions

`addlistener` | `daq.createSession` | `startBackground`

### Properties

`IsNotifyWhenDataAvailableExceedsAuto` | `NotifyWhenDataAvailableExceeds`

### Topics

“Acquire Data in the Background”

### Introduced in R2010b

# DataRequired Event

Notify when additional data is required for output on continuous generation

## Syntax

```
lh = addlistener(session, 'DataRequired', callbackfct);  
lh = addlistener(session, 'DataRequired', @(src, event) expr);
```

## Description

`lh = addlistener(session, 'DataRequired', callbackfct);` creates a listener for the `DataRequired` event. When more data is required, the callback is executed. The callback is typically used to queue more data to the device. The callback can be any MATLAB function with the `(src, event)` signature.

---

**Tips** Frequency is controlled by `NotifyWhenScansQueuedBelow`.

---

`lh = addlistener(session, '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 in a separate file. For more information see [Anonymous Functions \(MATLAB\)](#).

The callback has two required parameters: `src` and `event`. `src` is the session object for the listener and `event` is a `daq.DataRequiredInfo` object.

## Examples

### Add an Anonymous Listener to a Signal Generation Session

Create a session and add two analog output channels.

```
s = daq.createSession('ni');
s.IsContinuous = true;
addAnalogOutputChannel(s, '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, add an anonymous listener, and generate the signal in the background.

```
queueOutputData(s, [outputData0, outputData1]);
lh = addlistener(s, 'DataRequired', ...
    @(src,event) src.queueOutputData([outputData0, outputData1]));
```

Generate the output data and pause for up to 15 seconds.

```
startBackground(s);
pause(15)
```

Delete the listener.

```
delete(lh)
```

## Input Arguments

### **session** — Data acquisition session

session object

Data acquisition session, specified as a session object. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

### **callbackfct** — Callback function

function handle

Callback function, specified as a function handle.

### **expr** — Anonymous callback function

MATLAB operation



Anonymous callback function, specified as a MATLAB operation. The expression executes when the trigger occurs.

## See Also

### Functions

`addlistener` | `daq.createSession` | `startBackground`

### Properties

`IsContinuous` | `IsNotifyWhenScansQueuedBelowAuto` | `NotifyWhenScansQueuedBelow`

### Introduced in R2010b

## ErrorOccurred Event

Notify when device-related errors occur

### Syntax

```
lh = addlistener(session, 'ErrorOccurred', callbackfct);  
lh = addlistener(session, 'ErrorOccurred', @(src, event) expr);
```

### Description

`lh = addlistener(session, 'ErrorOccurred', callbackfct);` creates a listener for the `ErrorOccurred` event. When an error occurs, the callback is executed. The callback can be any MATLAB function with the `(src, event)` signature.

---

**Note** In background mode, errors and exceptions are not displayed by default. Use the `ErrorOccurred` event listener to display the errors.

---

`lh = addlistener(session, '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 requiring that your function be saved in a separate file. For more information, see [Anonymous Functions \(MATLAB\)](#).

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 can use the `MException.getReport` method to return a formatted message that uses the same format as errors thrown by internal MATLAB code.

## Examples

### Add a Listener to Display an Error Report

Create a session, and add an analog input channel.

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

Get a formatted report of the error.

```
lh = addlistener(s, 'ErrorOccurred', @(src,event) disp(getReport(event.Error)));
```

Acquire data, wait, and delete the listener.

```
startBackground(s);  
wait(s)  
delete(lh)
```

## Input Arguments

### **session** — Data acquisition session

session object

Data acquisition session, specified as a session object. Create the session object using `daq.createSession`. Use the data acquisition session for acquisition and generation operations. Create one session per vendor and use that vendor session to perform all data acquisition operations.

### **callbackfct** — Callback function

function handle

Callback function, specified as a function handle.

### **expr** — Anonymous callback function

MATLAB operation

Anonymous callback function, specified as a MATLAB operation. The expression executes when the trigger occurs.

## See Also

### Functions

`addListener` | `daq.createSession` | `startBackground`

### Classes

`MException`

**Introduced in R2010b**

# Properties – Alphabetical List

---

## Data Acquisition Toolbox Properties

Data Acquisition Toolbox uses the following properties.

- ActiveEdge
- ActivePulse
- ADCTimingMode
- AutoSyncDSA
- BitsPerSample
- BridgeMode
- Channels
- Connections
- CountDirection
- Coupling
- Destination
- Device
- Direction
- DurationInSeconds
- DutyCycle
- EncoderType
- EnhancedAliasRejectionEnable
- ExcitationCurrent
- ExcitationSource
- ExcitationVoltage
- ExternalTriggerTimeout
- Frequency
- FrequencyLimit
- Gain
- ID
- IdleState
- InitialCount

- InitialDelay
- IsContinuous
- IsDone
- IsLogging
- IsNotifyWhenDataAvailableExceedsAuto
- IsNotifyWhenScansQueuedBelowAuto
- IsRunning
- IsSimulated
- IsWaitingForExternalTrigger
- MaxSoundPressureLevel
- MeasurementType
- Name
- NominalBridgeResistance
- NotifyWhenDataAvailableExceeds
- NotifyWhenScansQueuedBelow
- NumberOfScans
- Offset
- Phase
- R0
- Range
- Rate
- RateLimit
- RTDConfiguration
- RTDType
- ScansAcquired
- ScansOutputByHardware
- ScansQueued
- Sensitivity
- ShuntLocation
- ShuntResistance

- Source
- StandardSampleRates
- Terminal
- TerminalConfig
- Terminals
- ThermocoupleType
- TriggerCondition
- TriggersPerRun
- TriggersRemaining
- TriggerType
- Units
- UserData
- UseStandardSampleRates
- Vendor
- WaveformType
- ZResetCondition
- ZResetEnable
- ZResetValue



# ActiveEdge

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 = addCounterInputChannel(s, '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

### Functions

`addCounterInputChannel`, `addCounterOutputChannel`

# ActivePulse

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 = addCounterInputChannel(s, 'cDAQ1Mod5', 0, 'PulseWidth')
```

```
ch =
```

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

```
    ActivePulse: High  
    Terminal: 'PFI4'  
    Name: empty  
    ID: 'ctrl'  
    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 'cDAQ1Mod5':

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

## See Also

`addCounterInputChannel`

# ADCTimingMode

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');
ch = addAnalogInputChannel(s, 'cDAQ1Mod1', 'ai1', 'Voltage');

ch
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'`:

```
ch.ADCTimingMode = 'HighResolution';
```

## See Also

`addAnalogInputChannel`

# AutoSyncDSA

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')
addAnalogInputChannel(s, 'PXI1Slot2', 0, 'Voltage');
addAnalogInputChannel(s, 'PXI1Slot3', 1, 'Voltage');
```

Enable automatic synchronization and acquire data”

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

## See Also

[addAnalogInputChannel](#)

# BitsPerSample

Display bits per sample

## Description

This property displays the maximum value of bits per sample of the device, based on the device specifications. By default this read-only value is 24.

## Example

### View BitsPerSample Property

Create an audio input session and display session properties.

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

```
s =
```

```
Data acquisition session using DirectSound hardware:  
  Will run for 1 second (44100 scans) at 44100 scans/second.  
  No channels have been added.
```

Properties, Methods, Events

Click on the **Properties** link.

```
UseStandardSampleRates: true  
    BitsPerSample: 24  
    StandardSampleRates: [1x15 double]  
    NumberOfScans: 44100  
    DurationInSeconds: 1  
    Rate: 44100  
    IsContinuous: false  
    NotifyWhenDataAvailableExceeds: 4410  
IsNotifyWhenDataAvailableExceedsAuto: true  
    NotifyWhenScansQueuedBelow: 22050  
IsNotifyWhenScansQueuedBelowAuto: true
```



```
ExternalTriggerTimeout: 10
  TriggersPerRun: 1
    Vendor: DirectSound
    Channels: ''
    Connections: ''
    IsRunning: false
    IsLogging: false
    IsDone: false
IsWaitingForExternalTrigger: false
  TriggersRemaining: 1
    RateLimit: ''
    ScansQueued: 0
  ScansOutputByHardware: 0
  ScansAcquired: 0
```

## See Also

[StandardSampleRates](#) | [UseStandardSampleRates](#) | [addAudioInputChannel](#) | [addAudioOutputChannel](#)

# BridgeMode

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.

## Examples

### Set BridgeMode Property

Set the BridgeMode property of an analog input Bridge measurement type channel.

Create a session and add an analog input Bridge channel.

```
s = daq.createSession('ni');  
ch = addAnalogInputChannel(s, 'cDAQ1Mod7', 0, 'Bridge');
```

Set the BridgeMode property to 'Full' and view the channel properties.

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

```
ch =
```

```
Data acquisition analog input channel 'ai0' on device 'cDAQ1Mod7':
```

```
    BridgeMode: Full  
    ExcitationSource: Internal
```

```
ExcitationVoltage: 2.5
NominalBridgeResistance: 'Unknown'
    Range: -0.063 to +0.063 VoltsPerVolt
    Name: ''
    ID: 'ai0'
    Device: [1x1 daq.ni.CompactDAQModule]
MeasurementType: 'Bridge'
ADCTimingMode: HighResolution
```

## See Also

`addAnalogInputChannel`

# Channels

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 “Hardware Discovery and Session Setup”.

---

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

---

## Values

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

## Example

### Access Channels Property

Create both analog and digital channels in a session and display the Channels property.

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

```
s = daq.createSession('ni');  
aich = addAnalogInputChannel(s, 'cDAQ1Mod7', 0, 'Bridge');  
  
aich =
```

```
Data acquisition analog input channel 'ai0' on device 'cDAQ1Mod7':
```

```
    BridgeMode: Unknown
```

```

    ExcitationSource: Internal
    ExcitationVoltage: 2.5
    NominalBridgeResistance: 'Unknown'
        Range: -0.025 to +0.025 VoltsPerVolt
        Name: ''
        ID: 'ai0'
        Device: [1x1 daq.ni.CompactDAQModule]
    MeasurementType: 'Bridge'
    ADCTimingMode: HighResolution

```

Add an analog output channel and view the Channels property.

```
aoch = addAnalogOutputChannel(s, 'cDAQ1Mod2', 'ao1', 'Voltage')
```

```
aoch =
```

```
Data acquisition analog output voltage channel 'ao1' on device 'cDAQ1Mod2':
```

```

    TerminalConfig: SingleEnded
        Range: -10 to +10 Volts
        Name: ''
        ID: 'ao1'
        Device: [1x1 daq.ni.CompactDAQModule]
    MeasurementType: 'Voltage'

```

Add a digital channel with 'InputOnly'.

```
dich = addDigitalChannel(s, 'dev1', 'Port0/Line0:1', 'InputOnly')
```

```
dich =
```

```
Number of channels: 2
```

index	Type	Device	Channel	MeasurementType	Range	Name
1	dio	Dev1	port0/line0	InputOnly	n/a	
2	dio	Dev1	port0/line1	InputOnly	n/a	

Change the TerminalConfig property of the input channel to 'SingleEnded'.

```
aich.TerminalConfig = 'SingleEnded';
```

You can use the channel object to access and edit the Channels property.

### **See Also**

#### **Functions**

`addAnalogInputChannel`, `addAnalogOutputChannel`

# Connections

Array of connections in 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 `addTriggerConnection` and `addClockConnection` to add connections. Use `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')
addAnalogInputChannel(s, 'Dev1', 0, 'voltage');
addAnalogInputChannel(s, 'Dev2', 0, 'voltage');
addAnalogInputChannel(s, 'Dev3', 0, 'voltage');
addTriggerConnection(s, 'Dev1/PFI4', 'Dev2/PFI0', 'StartTrigger');
addTriggerConnection(s, 'Dev1/PFI4', 'Dev3/PFI0', 'StartTrigger');
addClockConnection(s, '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.

```
removeConnection(s,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' 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

## See Also

### Function

addTriggerConnection, addClockConnection,



# CountDirection

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 decrements.

## Examples

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

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

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

## **See Also**

`addCounterInputChannel`

# Coupling

Specify input coupling mode

## Description

The `Coupling` property indicates the coupling mode used for the analog input signal connection. You cannot change the value for devices that support only one mode. For devices that support both AC and DC coupling, you can specify the mode by changing this property value.

If `Coupling` is set to `'DC'`, the signal input is connected directly to the amplifier, allowing measurement of the complete signal including its DC bias component. This is typically used with slowly changing signals such as temperature, pressure, or voltage readings.

If `Coupling` is set to `'AC'`, a series capacitor is inserted between the input connector and the amplifier, filtering out the DC bias component of the measured signal. This is typically used with dynamic signals such as audio.

## Values

- |                   |   |
|-------------------|---|
| <code>'DC'</code> | Direct input connection to amplifier. Default for any device that supports DC coupling.   |
| <code>'AC'</code> | Series capacitor inserted between the input connector and the amplifier. Default for any device that supports only AC coupling. |

## Examples

Create a session and add an analog input channel. Then change the coupling mode to `'AC'`.

```
s = daq.createSession('ni');  
ch = addAnalogInputChannel(s, 'Dev4', 'ai1', 'Voltage')
```

```
ch.Coupling = 'AC'
```

### See Also

#### Functions

[addAnalogInputChannel](#)

#### Properties

[Range](#) | [TerminalConfig](#)

# Destination

Indicates trigger destination terminal

## Description

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

## Example

### Examine a Trigger Connection Destination

Create a session with a trigger connection and examine the connection properties.

Create a session and add 2 analog input channels from different devices.

```
s = daq.createSession('ni');  
addAnalogInputChannel(s, 'Dev1', 0, 'voltage');  
addAnalogInputChannel(s, 'Dev2', 0, 'voltage');
```

Add a trigger connection and examine the connection properties.

```
addTriggerConnection(s, 'Dev1/PFI4', 'Dev2/PFI0', 'StartTrigger')
```

```
ans =
```

```
Start Trigger is provided by 'Dev1' at 'PFI4' and will be received by 'Dev2' at terminal 'PFI0'.
```

```
    TriggerType: 'Digital'  
TriggerCondition: RisingEdge  
    Source: 'Dev1/PFI4'  
Destination: 'Dev2/PFI0'  
    Type: StartTrigger
```

## See Also

Source, `addTriggerConnection`

# Device

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 = addCounterInputChannel(s, '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

`addCounterInputChannel`, `addCounterOutputChannel`

## Direction

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 messages. By default the channel is set to `Unknown`. Change the direction to output signal 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';
```

## Change the Direction of a Digital Channel

Change the direction of a bidirectional digital channel to `Input`.

Create a session and add a bidirectional digital channel.

```
s = daq.createSession('ni')
ch = addDigitalChannel(s, 'dev6', 'Port0/Line0', 'Bidirectional')
```

```
ch =
```

```
Data acquisition digital bidirectional (unknown) channel 'port0/line0' on device 'Dev6':
```

```
    Direction: Unknown
```

```
        Name: ''
        ID: 'port0/line0'
        Device: [1x1 daq.ni.DeviceInfo]
MeasurementType: 'Bidirectional (Unknown)'
```

Change the channels direction to 'Input'.

```
ch.Direction = 'Input'
```

```
ch =
```

```
Data acquisition digital bidirectional (input) channel 'port0/line0' on device 'Dev6':
```

```
        Direction: Input
        Name: ''
        ID: 'port0/line0'
        Device: [1x1 daq.ni.DeviceInfo]
MeasurementType: 'Bidirectional (Input)'
```

```
Properties, Methods, Events
```



# DurationInSeconds

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 analog, digital, or audio output channels, `DurationInSeconds` is a read-only property whose value is determined by  $\frac{s.ScansQueued}{s.Rate}$ .

If the session contains only counter output channels with `PulseGeneration` measurement type, then `DurationInSeconds` represents the duration of the pulse train signal generation.

## Values

In a session with only input channels or counter output 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, `DurationInSeconds` is set to 1 second.

## Examples

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

```
s = daq.createSession('ni');
addAnalogInputChannel(s,'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	MeasurementType	Range	Name
1						

-----  
1 ai cDAQ1Mod1 ai0 Voltage (Diff) -10 to +10 Volts

### **See Also**

### **Properties**

NumberOfScans, Rate

### **Functions**

addCounterInputChannel

# DutyCycle

Duty cycle of 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, the time the signal is high is equal to the time the signal is low.

For function generation channels using Digilent devices, each waveform adopts the duty cycle

## Examples

### Specify Duty Cycle

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

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

```
ch =
```

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

```
    IdleState: Low  
    InitialDelay: 2.5e-08  
    Frequency: 100  
    DutyCycle: 0.5  
    Terminal: 'PFI0'  
    Name: ''  
    ID: 'ctr0'  
    Device: [1x1 daq.ni.CompactDAQModule]  
    MeasurementType: 'PulseGeneration'
```

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

### ch.DutyCycle

ch =

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

```
IdleState: Low
InitialDelay: 2.5e-08
Frequency: 100
DutyCycle: 0.25
Terminal: 'PFI0'
Name: ''
ID: 'ctr0'
Device: [1x1 daq.ni.CompactDAQModule]
MeasurementType: 'PulseGeneration'
```

You can change the channel duty cycle while the session is running when using counter output channels.

## See Also

### Functions

`addCounterOutputChannel`

### Properties

Gain | Offset | Phase

# EncoderType

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'`

## Example

### Change Encoder Type Property

Change the `EncoderType` property of a counter input channel with a `Position` measurement type.

Create a session and add a counter input channel with `Position` measurement type.

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

```
ch =
```

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

```
EncoderType: X1  
ZResetEnable: 0  
ZResetValue: 0  
ZResetCondition: BothHigh  
TerminalA: 'PFI0'  
TerminalB: 'PFI2'  
TerminalZ: 'PFI1'
```

```
Name: ''
ID: 'ctr0'
Device: [1x1 daq.ni.CompactDAQModule]
MeasurementType: 'Position'
```

Change the channels encoder type to X2.

```
ch.EncoderType = 'X2'
```

```
ch =
```

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

```
EncoderType: X2
ZResetEnable: 0
ZResetValue: 0
ZResetCondition: BothHigh
TerminalA: 'PFI0'
TerminalB: 'PFI2'
TerminalZ: 'PFI1'
Name: ''
ID: 'ctr0'
Device: [1x1 daq.ni.CompactDAQModule]
MeasurementType: 'Position'
```

## See Also

`addCounterInputChannel`

# EnhancedAliasRejectionEnable

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.

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

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

## Example

### Enable Enhanced Alias Rejection

Enable enhanced alias rejection on a DSA device.

Create a session and add an analog input voltage channel using a DSA device.

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

```
ch =
```

```
Data acquisition analog input voltage channel 'ai0' on device 'PXI1Slot2':
```

```
    Coupling: DC
TerminalConfig: PseudoDifferential
    Range: -42 to +42 Volts
    Name: ''
        ID: 'ai0'
        Device: [1x1 daq.ni.PXIDSAModule]
    MeasurementType: 'Voltage'
```

```
EnhancedAliasRejectionEnable: 0
```

Enable enhanced alias rejection.

`ch.EnhancedAliasRejectionEnable = 1`

`ch =`

Data acquisition analog input voltage channel 'ai0' on device 'PXI1Slot2':

```
    Coupling: DC
    TerminalConfig: PseudoDifferential
    Range: -42 to +42 Volts
    Name: ''
    ID: 'ai0'
    Device: [1x1 daq.ni.PXIDSAModule]
    MeasurementType: 'Voltage'
    EnhancedAliasRejectionEnable: 1
```

## See Also

`AutoSyncDSA`



# ExcitationCurrent

Current of external source of excitation

## Description

When working with the session-based interface, the `ExcitationCurrent` property indicates the current in amps 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 a range of excitation currents, the default will be the lowest available value in the range.

## Example

### Change Excitation Current Value

Change the excitation current value of a microphone channel.

Create a session and add an analog input microphone channel.

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

```
ch =
```

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

```
    Sensitivity: 'Unknown'
MaxSoundPressureLevel: 'Unknown'
ExcitationCurrent: 0.002
ExcitationSource: Internal
    Coupling: AC
    TerminalConfig: PseudoDifferential
        Range: -5.0 to +5.0 Volts
        Name: ''
        ID: 'ai0'
    Device: [1x1 daq.ni.CompactDAQModule]
MeasurementType: 'Microphone'
ADCTimingMode: ''
```

Change the excitation current value to 0.0040.

```
ch.ExcitationCurrent = .0040
```

```
ch =
```

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

```
    Sensitivity: 'Unknown'  
    MaxSoundPressureLevel: 'Unknown'  
    ExcitationCurrent: 0.004  
    ExcitationSource: Internal  
    Coupling: AC  
    TerminalConfig: PseudoDifferential  
    Range: -5.0 to +5.0 Volts  
    Name: ''  
    ID: 'ai0'  
    Device: [1x1 daq.ni.CompactDAQModule]  
    MeasurementType: 'Microphone'  
    ADCTimingMode: ''
```

## See Also

### Properties

ExcitationSource

### Functions

addAnalogInputChannel

# ExcitationSource

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`.

## Example

### Change Excitation Source

Change the excitation source of a microphone channel.

Create a session and add an analog input microphone channel.

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

```
ch =
```

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

```
    Sensitivity: 'Unknown'  
MaxSoundPressureLevel: 'Unknown'  
ExcitationCurrent: 0.004  
ExcitationSource: Unknown  
    Coupling: AC  
TerminalConfig: PseudoDifferential  
    Range: -5.0 to +5.0 Volts
```

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

Change the excitation source value to 'Internal'.

```
ch.ExcitationSource = 'Internal'
```

```
ch =
```

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

```
    Sensitivity: 'Unknown'
MaxSoundPressureLevel: 'Unknown'
ExcitationCurrent: 0.004
ExcitationSource: Internal
    Coupling: AC
    TerminalConfig: PseudoDifferential
    Range: -5.0 to +5.0 Volts
    Name: ''
    ID: 'ai0'
    Device: [1x1 daq.ni.CompactDAQModule]
MeasurementType: 'Microphone'
ADCTimingMode: ''
```

## See Also

### Properties

ExcitationCurrent

ExcitationVoltage

### Functions

addAnalogInputChannel

# ExcitationVoltage

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 available value in the range.

## See Also

## Properties

`ExcitationSource`

# ExternalTriggerTimeout

Specify maximum wait time for external trigger

## Description

The data acquisition session `ExternalTriggerTimeout` property specifies the maximum amount of time in seconds the session waits for an external trigger before timing out. To disable the timeout, set `ExternalTriggerTimeout` to a value of `Inf`.

## Examples

### Specify External Trigger Timeout

Specify how long the session waits for an external trigger before timing out.

Create a session and click on the [Properties](#) link to display session properties.

```
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.
```

Properties, Methods, Events

```
AutoSyncDSA: false  
NumberOfScans: 1000  
DurationInSeconds: 1  
Rate: 1000  
IsContinuous: false  
NotifyWhenDataAvailableExceeds: 100  
IsNotifyWhenDataAvailableExceedsAuto: true  
NotifyWhenScansQueuedBelow: 500  
IsNotifyWhenScansQueuedBelowAuto: true  
ExternalTriggerTimeout: 10
```

```
TriggersPerRun: 1
  Vendor: National Instruments
  Channels: ''
  Connections: ''
  IsRunning: false
  IsLogging: false
  IsDone: false
IsWaitingForExternalTrigger: false
  TriggersRemaining: 1
  RateLimit: ''
  ScansQueued: 0
  ScansOutputByHardware: 0
  ScansAcquired: 0
```

Change the timeout to 15 seconds.

```
s.ExternalTriggerTimeout = 15;
```

## Specify External Trigger with Disabled Timeout

Set an external trigger on a session, without a timeout.

Create a session with an external trigger, then set its `ExternalTriggerTimeout` to `Inf`.

```
s = daq.createSession('ni');
addAnalogInputChannel(s, 'Dev1', 'ai0', 'Voltage');
addTriggerConnection(s, 'External', 'Dev1/PFI0', 'StartTrigger');
s.ExternalTriggerTimeout = Inf;
```

## See Also

### Functions

`addTriggerConnection`

# Frequency

Frequency of generated output

## Description

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

When working with function generation channel, data acquisition sessions, the rate of a waveform is controlled by the channel `Frequency` property. To synchronize all operation in the session, set each channel generation rate individually, and change the session `Rate` to match the channel generation rate.

The frequency value must fall within the specified `FrequencyLimit` values.

## Values

Specify the frequency in hertz.

## Examples

### Set the Frequency of a Counter Input Channel

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

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

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

```
ch.Frequency = 200;  
  
ch  
  
ans =
```



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

```

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

```

## Set the Frequency of a Function Generator Channel

Create a waveform generation channel, and change the generation rate to 20000 scans per second.

```

s = daq.createSession('digilent'):
fgenCh = addFunctionGeneratorChannel(s, 'AD1', 1, 'Sine'
fgenCh.Frequency = 20000

```

```
fgenCh =
```

Data acquisition sine waveform generator '1' on device 'AD1':

```

Phase: 0
Range: -5.0 to +5.0 Volts
TerminalConfig: SingleEnded
Gain: 1
Offset: 0
Frequency: 20000
WaveformType: Sine
FrequencyLimit: [0.0 25000000.0]
Name: ''
ID: '1'
Device: [1x1 daq.di.DeviceInfo]
MeasurementType: 'Voltage'

```

---

**Tip** You can change the channel frequency while the session is running when using counter output channels.

---

## See Also

### Functions

`addCounterInputChannel` | `addFunctionGeneratorChannel`

### Properties

`FrequencyLimit`

# FrequencyLimit

Limit of rate of operation based on hardware configuration

## Description

In the session-based interface, the read-only `FrequencyLimit` property displays the minimum and maximum rates that the function generation channel supports.

---

**Tip** `FrequencyLimit` changes dynamically as the channel configuration changes.

---

## Example

View waveform function generation channel's generation rate limit.

```
s = daq.createSession('digilent')
fgenCh = addFunctionGeneratorChannel(s, 'AD1', 1, 'Sine')
fgenCh.FrequencyLimit
```

```
ans =
```

```
[0.0 25000000.0]
```

## See Also

### Properties

Frequency

# Gain

Waveform output gain

## Description

When using waveform function generation channels, **Gain** represents the value by which the scaled waveform data is multiplied to get the output data.

## Values

The waveform gain can be between  $-5$  and  $5$ . Ensure that  $\text{Gain} \times \text{Voltage} + \text{Offset}$  falls within the valid ranges of output voltage of the device.

## Example

Change the gain of the waveform function generation channel to 2 volts.

```
s = daq.createSession('digilent');  
fgenCh = addFunctionGeneratorChannel(s, 'AD1', 1, 'Sine');  
fgenCh.Gain = 2
```

```
fgenCh =
```

```
Data acquisition sine waveform generator '1' on device 'AD1':
```

```
    Phase: 0  
    Range: -5.0 to +5.0 Volts  
TerminalConfig: SingleEnded  
    Gain: 2  
    Offset: 0  
    Frequency: 4096  
    WaveformType: Sine  
FrequencyLimit: [0.0 25000000.0]  
    Name: ''  
    ID: '1'
```

```
Device: [1x1 daq.di.DeviceInfo]  
MeasurementType: 'Voltage'
```

## See Also

### Functions

`addFunctionGeneratorChannel`

### Properties

`DutyCycle` | `Offset` | `Phase`

# ID

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.

## Examples

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

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

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

## See Also

`addCounterInputChannel`

# IdleState

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 'cDAQ1Mod5':
```

```
    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

`addCounterOutputChannel`

# InitialCount

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.

## Examples

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

```
s = daq.createSession('ni');  
ch = addCounterInputChannel(s, '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 `InitialCount` to 15:

```
ch.InitialCount = 15
```

```
ch =
```

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

```
    ActiveEdge: Rising  
    CountDirection: Increment  
    InitialCount: 15  
    Terminal: 'PFI8'  
    Name: empty  
    ID: 'ctr0'  
    Device: [1x1 daq.ni.DeviceInfo]  
    MeasurementType: 'EdgeCount'
```

## See Also

### Functions

`addCounterInputChannel` | `resetCounters`

# InitialDelay

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.

## Example

### Specify Initial Delay

Set the initial delay on a counter output channel to 3.

Create a session and add a counter input channel.

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

Set the initial delay.

```
ch.InitialDelay = 3
```

```
ch =
```

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

```
  IdleState: Low  
  InitialDelay: 3  
  Frequency: 100  
  DutyCycle: 0.5  
  Terminal: 'PFI0'  
  Name: ''  
  ID: 'ctr0'
```

```
Device: [1x1 daq.ni.CompactDAQModule]  
MeasurementType: 'PulseGeneration'
```

## See Also

`addCounterOutputChannel`

# IsContinuous

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 `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 `stop`.

## Examples

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

```
s = daq.createSession('ni');
addAnalogInputChannel(s, '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	MeasurementType	Range	Name
1	ai	cDAQ1Mod1	ai0	Voltage (Diff)	-10 to +10 Volts	

## **See Also**

### **Properties**

IsDone

### **Functions**

stop,startBackground

# IsDone

Indicate if session operation is complete

## Description

The read-only `IsDone` property indicates that the session operation is complete.

---

### Tip

- `IsRunning` indicates the session has started, but the hardware might not be acquiring or generating data. It is still true while the hardware is waiting for a trigger, and while transferring data in the process of stopping.
  - `IsLogging` indicates the hardware is actively acquiring or generating data.
  - `IsDone` indicates the session object has completed its operation, including all necessary transfer of data.
- 

## Values

`true`

Value is logical 1 (`true`) when the session operation is complete.

`false`

Value is logical 0 (`false`) while the session operation is not complete.

## Examples

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

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

```
wait(s)  
s.IsDone
```

```
ans =
```

```
    1
```

## See Also

### Properties

IsLogging | IsRunning

### Functions

startBackground

# IsLogging

Indicate if hardware is acquiring or generating data

## Description

The read-only `IsLogging` property indicates if the hardware is actively acquiring or generating data.

---

### Tip

- `IsRunning` indicates the session has started, but the hardware might not be acquiring or generating data. It is still true while the hardware is waiting for a trigger, and while transferring data in the process of stopping.
  - `IsLogging` indicates the hardware is actively acquiring or generating data.
  - `IsDone` indicates the session object has completed its operation, including all necessary transfer of data.
- 

## Values

`true`

Value is logical 1 (`true`) if the device is acquiring or generating data.

`false`

Value is logical 0 (`false`) if the device is not acquiring or generating data.

## Examples

Create and start a session.

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

```
startBackground(s);  
s.IsRunning
```

```
ans =
```

```
1
```

The session is running, so check for device logging.

```
s.IsLogging
```

```
ans =
```

```
0
```

This result might indicate that the device is waiting for an external trigger. After triggering, wait until logging is complete.

```
wait(s)  
s.IsDone
```

```
ans =
```

```
1
```

## See Also

### Properties

IsDone | IsRunning

### Functions

startBackground

# IsNotifyWhenDataAvailableExceedsAuto

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.

## Example

### Enable Data Exceeds Notification

Change the `IsNotifyWhenDataAvailableExceedsAuto` to be able to set the `NotifyWhenDataAvailableExceeds` property to a specific value.

Create a session and display the properties by clicking the `Properties` link.

```
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.
```

```
Properties, Methods, Events
```

```
      AutoSyncDSA: false  
      NumberOfScans: 1000  
      DurationInSeconds: 1  
          Rate: 1000  
      IsContinuous: false  
      NotifyWhenDataAvailableExceeds: 100  
IsNotifyWhenDataAvailableExceedsAuto: true  
      NotifyWhenScansQueuedBelow: 500  
IsNotifyWhenScansQueuedBelowAuto: true  
      ExternalTriggerTimeout: 10  
      TriggersPerRun: 1  
          Vendor: National Instruments  
          Channels: ''  
          Connections: ''  
          IsRunning: false  
          IsLogging: false  
          IsDone: false  
IsWaitingForExternalTrigger: false  
      TriggersRemaining: 1  
          RateLimit: ''  
          ScansQueued: 0  
      ScansOutputByHardware: 0  
      ScansAcquired: 0
```

```
Change the IsNotifyWhenDataAvailableExceedsAuto to
```

```
s.IsNotifyWhenDataAvailableExceedsAuto = false
```

```
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**

#### **Properties**

NotifyWhenDataAvailableExceeds

#### **Events**

DataAvailable

# IsNotifyWhenScansQueuedBelowAuto

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.

## Example

### Enable Notification When Scans Reach Below Specified Range

Change the `IsNotifyWhenScansQueuedBelowAuto` to be able to set the `NotifyWhenScansQueuedBelow` property to a specific value.

Create a session and display the properties by clicking the `Properties` link.

```
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.

Properties, Methods, Events

```
                AutoSyncDSA: false
                NumberOfScans: 1000
                DurationInSeconds: 1
                    Rate: 1000
                IsContinuous: false
                NotifyWhenDataAvailableExceeds: 100
IsNotifyWhenDataAvailableExceedsAuto: true
                NotifyWhenScansQueuedBelow: 500
IsNotifyWhenScansQueuedBelowAuto: true
                ExternalTriggerTimeout: 10
                TriggersPerRun: 1
                    Vendor: National Instruments
                    Channels: ''
                    Connections: ''
                    IsRunning: false
                    IsLogging: false
                    IsDone: false
                IsWaitingForExternalTrigger: false
                TriggersRemaining: 1
                    RateLimit: ''
                    ScansQueued: 0
                ScansOutputByHardware: 0
                ScansAcquired: 0
```

Change the `IsNotifyWhenDataAvailableExceedsAuto` to

```
s.IsNotifyWhenScansQueuedBelowAuto = false
```

```
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

### Properties

`NotifyWhenScansQueuedBelow`, `ScansQueued`



## **Events**

DataRequired

# IsRunning

Indicate if session operation is in progress

## Description

The read-only `IsRunning` property indicates the session operation is started and in progress, whether or not the hardware is acquiring or generating data at the time.

---

### Tip

- `IsRunning` indicates the session has started, but the hardware might not be acquiring or generating data. It is still true while the hardware is waiting for a trigger, and while transferring data in the process of stopping.
  - `IsLogging` indicates the hardware is actively acquiring or generating data.
  - `IsDone` indicates the session object has completed its operation, including all necessary transfer of data.
- 

## Values

`true`

Value is logical 1 (`true`) while the session operation is in progress.

`false`

Value is logical 0 (`false`) while the session operation is not in progress, that is, before it starts or after it stops.

## Examples

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

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

```
lh = s.addlistener('DataAvailable', @plotData);  
  
function plotData(src,event)  
    plot(event.TimeStamps, event.Data)  
end  
startBackground(s);
```

See if the session is in progress.

```
s.IsRunning
```

```
ans =
```

```
1
```

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

```
wait(s)  
s.IsRunning
```

```
ans =
```

```
0
```

## See Also

### Properties

IsDone | IsLogging

### Functions

startBackground

# IsSimulated

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 'cDAQ1'.

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

## **Functions**

startBackground

## IsWaitingForExternalTrigger

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

`addTriggerConnection`

# MaxSoundPressureLevel

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 `MaxSoundPressureLevel` 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.

## Example

### Change Maximum Sound Pressure of Microphone

Change the `Sensitivity` of a microphone channel and set the maximum sound pressure level to 10.

Create a session and add a microphone channel.

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

```
ch =
```

```
Data acquisition analog input microphone channel 'ai0' on device 'cDAQ1Mod3':
    Sensitivity: 'Unknown'
    MaxSoundPressureLevel: 'Unknown'
    ExcitationCurrent: 0.002
    ExcitationSource: Internal
```



```

Coupling: AC
TerminalConfig: PseudoDifferential
  Range: -5.0 to +5.0 Volts
  Name: ''
  ID: 'ai0'
Device: [1x1 daq.ni.CompactDAQModule]
MeasurementType: 'Microphone'
ADCTimingMode: ''

```

Set the channel's sensitivity to 3 0 .037.

```
ch.Sensitivity = 0.037
```

```
ch =
```

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

```

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

```

Set the channel's maximum sound pressure to 10 db.

```
ch.MaxSoundPressureLevel = 10
```

```
ch =
```

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

```

Sensitivity: 0.037
MaxSoundPressureLevel: 10
ExcitationCurrent: 0.002
ExcitationSource: Internal
Coupling: AC
TerminalConfig: PseudoDifferential
  Range: -135 to +135 Pascals
  Name: ''
  ID: 'ai0'
Device: [1x1 daq.ni.CompactDAQModule]
MeasurementType: 'Microphone'
ADCTimingMode: ''

```

# MeasurementType

Channel measurement type

## Description

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

## Values

You can only use `Audio` measurement type with multichannel audio devices.

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 = addCounterInputChannel (s, 'cDAQ1Mod5', 0, 'EdgeCount')
```

```
ch =
```

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

```
    ActiveEdge: Rising  
    CountDirection: Increment  
    InitialCount: 0  
    Terminal: 'PF18'  
    Name: empty  
    ID: 'ctr0'  
    Device: [1x1 daq.ni.DeviceInfo]  
    MeasurementType: 'EdgeCount'
```

## See Also

`addAnalogInputChannel`, `addAnalogOutputChannel`, `addCounterInputChannel`,  
`addCounterOutputChannel`,

### Name

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 character vector 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');  
ch = addAnalogInputChannel(s, 'Dev1', 0, 'Voltage')
```

```
ch =
```

```
Data acquisition analog input voltage channel 'ai0' on device 'Dev1':
```

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

Change Name to 'AI-Voltage'.

```
ch.Name = 'AI-Voltage'
```

ch =

Data acquisition analog input voltage channel 'ai0' on device 'Dev1':

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

## See Also

`addAnalogInputChannel`

## **NominalBridgeResistance**

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**

`addAnalogInputChannel`

# NotifyWhenDataAvailableExceeds

Control firing of DataAvailable event

## Description

The DataAvailable event is triggered when the number of scans available to the session object exceeds the quantity specified in the NotifyWhenDataAvailableExceeds property.

You cannot set the NotifyWhenDataAvailableExceeds property when the session is in the prepared state, which can happen after running startForeground. In this case, call release on the session before setting this property value.

## Values

By default the DataAvailable event triggers when 1/10 second worth of data is available for analysis. To specify a different threshold, change the value of NotifyWhenDataAvailableExceeds.

## Examples

### Control Firing of Data Available Event

Add an event listener to display the total number of scans acquired and fire the event when the data available exceeds specified amount.

Create the session and add an analog input voltage channel.

```
s = daq.createSession('ni');  
addAnalogInputChannel(s, 'Dev4', 1, 'Voltage');  
lh = addlistener(s, '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.

Increase the Rate to 800,000 scans per second, while 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 = startForeground(s);
```

```
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
```



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

```
data = startForeground(s);  
  
          160000  
          320000  
          480000  
          640000  
          800000
```

Set `IsNotifyWhenDataAvailableExceedsAuto` back to `true`.

```
s.IsNotifyWhenDataAvailableExceedsAuto = true;  
s.NotifyWhenDataAvailableExceeds  
  
ans =  
          80000
```

This causes `NotifyWhenDataAvailableExceeds` to set automatically when `Rate` changes.

```
s.Rate = 50000;  
s.NotifyWhenDataAvailableExceeds  
  
ans =  
          5000
```

## See Also

### Properties

`IsNotifyWhenDataAvailableExceedsAuto`

### Events

`DataAvailable`

## **Functions**

`addListener, startBackground`

# NotifyWhenScansQueuedBelow

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 this property value to control when the event is fired.

## Example

### Control When DataRequired Event Is Fired

Specify a threshold below which the DataRequired event fires.

Create a session and add an analog output channel.

```
s = daq.createSession('ni')
addAnalogOutputChannel(s, 'cDAQ1Mod2', 0, 'Voltage')
```

Queue some output data.

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

Set the threshold of scans queued to 100.

```
s.NotifyWhenScansQueuedBelow = 100;
```

Add an anonymous listener and generate the signal in the background:

```
lh = s.addListener('DataRequired', ...  
@(src,event) src.queueOutputData(outputData));  
  
startBackground(s);
```

## See Also

### Properties

ScansQueued, IsNotifyWhenScansQueuedBelowAuto

### Events

DataRequired

# NumberOfScans

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 `queueOutputData`.
- 

## Values

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

## Example

### Change Number of Scans

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

```
s = daq.createSession('ni');  
addAnalogInputChannel(s, '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	MeasurementType	Range	Name
1	ai	cDAQ1Mod1	ai0	Voltage (Diff)	-10 to +10 Volts	

## See Also

### Properties

ScansQueued, DurationInSeconds

### Functions

startForeground, startBackground, queueOutputData

## Offset

Specify DC offset of waveform

## Description

When using waveform function generation channels, `Offset` represents offsetting of a signal from zero, or the mean value of the waveform.

## Values

The waveform offset can be between  $-5$  and  $5$ . Ensure that  $\text{Gain} \times \text{Voltage} + \text{Offset}$  falls within the valid ranges of output voltage of the device.

## Example

Change the offset of the waveform function generation channel to 2 volts.

```
s = daq.createSession('digilent');
fgenCh = addFunctionGeneratorChannel(s, 'AD1', 1, 'Sine');
fgenCh.Offset = 2
```

```
fgenCh =
```

```
Data acquisition sine waveform generator '1' on device 'AD1':
```

```
    Phase: 0
    Range: -5.0 to +5.0 Volts
TerminalConfig: SingleEnded
    Gain: 0
    Offset: 2
    Frequency: 4096
    WaveformType: Sine
FrequencyLimit: [0.0 25000000.0]
    Name: ''
    ID: '1'
```

```
Device: [1x1 daq.di.DeviceInfo]  
MeasurementType: 'Voltage'
```

## See Also

### Functions

`addFunctionGeneratorChannel`

### Properties

`DutyCycle` | `Gain` | `Phase`



# Phase

Waveform phase

## Description

In a function generation channel, the Phase property specifies the period of waveform cycle from its point of origin. Specify the values for Phase in time units.

## Example

Set the phase of a waveform function generation channel to 33.

```
s = daq.createSession('digilent')
fgenCh = addFunctionGeneratorChannel(s, 'AD1', 1, 'Sine')
fgenCh.Phase = 33
```

```
fgenCh =
```

```
Data acquisition sine waveform generator '1' on device 'AD1':
```

```
        Phase: 33
        Range: -5.0 to +5.0 Volts
TerminalConfig: SingleEnded
        Gain: 1
        Offset: 0
        Frequency: 4096
        WaveformType: Sine
        FrequencyLimit: [0.0 25000000.0]
        Name: ''
        ID: '1'
        Device: [1x1 daq.di.DeviceInfo]
MeasurementType: 'Voltage'
```

## See Also

### Functions

`addFunctionGeneratorChannel`

**Properties**

DutyCycle | Gain | Offset

## R0

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 R0 property displays Unknown. You must change this value to set the resistance of this device to the temperature you want.

## Example

### Set RTD Channels Resistance

Create a session and add an RTD channel.

```
s = daq.createSession('ni');  
ch = addAnalogInputChannel(s, 'cDAQ1Mod7', 3, 'RTD');
```

Change the channels resistance to 100°C.

```
ch.R0 = 100
```

```
ch =
```

```
Data acquisition analog input RTD channel 'ai3' on device 'cDAQ1Mod7':
```

```
        Units: Celsius  
        RTDType: Unknown  
    RTDConfiguration: Unknown  
            R0: 100  
ExcitationCurrent: 0.0005  
ExcitationSource: Internal  
        Coupling: DC  
    TerminalConfig: Differential  
            Range: -200 to +660 Celsius
```

```
Name: ''  
ID: 'ai3'  
Device: [1x1 daq.ni.CompactDAQModule]  
MeasurementType: 'RTD'  
ADCTimingMode: HighResolution
```

## See Also

### Properties

RTDConfiguration, RTDType

# Range

Specify channel measurement range

## Description

When working with the session-based interface, use the `Range` property 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.

## Examples

### Set Channel Range

Specify the range of an analog input voltage channel.

Create a session and add an analog input channel.

```
s = daq.createSession('ni');  
ch = addAnalogInputChannel(s, 'cDAQ1Mod7', 3, 'voltage');
```

Set a range of -60 to +60 volts.

```
ch.Range = [-60, 60];
```

### Display Ranges Available

See what ranges your channel supports before you set the channel range.

Create a session and add an analog input channel.

```
s = daq.createSession('ni');  
ch = addAnalogInputChannel(s, 'Dev1', 3, 'voltage');
```

Display channel device.

```
ch.Device
```

```
ans =
```

```
ni: National Instruments USB-6211 (Device ID: 'Dev1')  
  Analog input subsystem supports:  
    4 ranges supported  
    Rates from 0.1 to 250000.0 scans/sec  
    16 channels ('ai0' - 'ai15')  
    'Voltage' measurement type  
  
  Analog output subsystem supports:  
    -10 to +10 Volts range  
    Rates from 0.1 to 250000.0 scans/sec  
    2 channels ('ao0', 'ao1')  
    'Voltage' measurement type  
  
  Digital subsystem supports:  
    8 channels ('port0/line0' - 'port1/line3')  
    'InputOnly', 'OutputOnly' measurement types  
  
  Counter input subsystem supports:  
    Rates from 0.1 to 80000000.0 scans/sec  
    2 channels ('ctr0', 'ctr1')  
    'EdgeCount', 'PulseWidth', 'Frequency', 'Position' measurement types  
  
  Counter output subsystem supports:  
    Rates from 0.1 to 80000000.0 scans/sec  
    2 channels ('ctr0', 'ctr1')  
    'PulseGeneration' measurement type
```

Create a subsystems object.

```
sub = ch.Device.Subsystems
```

```
sub =
```

```
Analog input subsystem supports:  
  4 ranges supported  
  Rates from 0.1 to 250000.0 scans/sec
```

```
    16 channels ('ai0' - 'ai15')
    'Voltage' measurement type
    Properties, Methods, Events

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

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

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

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

Display the ranges available on the analog input subsystem.

```
sub(1).RangesAvailable
```

```
ans =
```

```
-0.20 to +0.20 Volts, -1.0 to +1.0 Volts, -5.0 to +5.0 Volts, -10 to +10 Volts
```

## See Also

`daq.createSession, addAnalogInputChannel`

# Rate

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

### Change Session Rate

Create a session and add an analog input channel.

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

Change the rate to 10000.

```
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	MeasurementType	Range	Name
1	ai	cDAQ1Mod1	ai1	Voltage (Diff)	-10 to +10 Volts	

## See Also

### Properties

[DurationInSeconds](#) | [NumberOfScans](#) | [RateLimit](#) | [StandardSampleRates](#) | [UseStandardSampleRates](#)

### Topics

“Multichannel Audio Session Rate”

# RateLimit

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.

---

## Example

### Display Sessions Rate Limit

Create session and add an analog input channel.

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

Examine the session's rate limit.

```
s.RateLimit
```

```
ans =
```

```
1.0e+05 *
```

0.0000 2.5000

## **See Also**

### **Properties**

Rate

## RTDConfiguration

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`

### Example

#### Specify Channel's RTD Configuration

Specify an RTD channels wiring configuration.

Create a session and add an RTD channel to it.

```
s = daq.createSession('ni');  
ch = addAnalogInputChannel(s, 'cDAQ1Mod7', 3, 'RTD');
```

Change the `RTDConfiguration` to `ThreeWire`.

```
ch.RTDConfiguration = 'ThreeWire'
```

```
ch =
```

```
Data acquisition analog input RTD channel 'ai3' on device 'cDAQ1Mod7':
```

```
Units: Celsius  
RTDType: Unknown
```

```
RTDConfiguration: ThreeWire
                   R0: 'Unknown'
ExcitationCurrent: 0.0005
ExcitationSource: Internal
                   Coupling: DC
TerminalConfig: Differential
                  Range: -200 to +660 Celsius
                  Name: ''
                  ID: 'ai3'
                  Device: [1x1 daq.ni.CompactDAQModule]
MeasurementType: 'RTD'
ADCTimingMode: HighResolution
```

## See Also

### Properties

R0, RTDType

# RTDType

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

## Example

### Set RTD Sensor Type

Set an RTD sensor's sensitivity type.

Create a session and add an RTD channel.

```
s = daq.createSession('ni');  
ch = addAnalogInputChannel(s, 'cDAQ1Mod7', 3, 'RTD');
```

Set the RTDType to Pt3851.

```
ch.RTDType = 'Pt3851'  
  
ch =
```

Data acquisition analog input RTD channel 'ai3' on device 'cDAQ1Mod7':

```
    Units: Celsius
    RTDType: Pt3851
    RTDConfiguration: ThreeWire
                   R0: 'Unknown'
ExcitationCurrent: 0.0005
ExcitationSource: Internal
    Coupling: DC
    TerminalConfig: Differential
                   Range: -200 to +660 Celsius
                   Name: ''
                   ID: 'ai3'
                   Device: [1x1 daq.ni.CompactDAQModule]
MeasurementType: 'RTD'
    ADCTimingMode: HighResolution
```

## See Also

### Functions

`addAnalogInputChannel`

### Properties

`R0` | `RTDConfiguration`

# ScansAcquired

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 `startBackground`.

## Values

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

## Example

### Display Number of Scans Acquired

Acquire analog input data and display the number of scans acquired.

Create a session, add an analog input channel,

```
s = daq.createSession('ni');  
ch = addAnalogInputChannel(s, 'Dev1', 'ai1', 'voltage');
```

See how many scan the session had acquired.

```
s.ScansAcquired
```

```
ans =
```

```
0
```

Start the acquisition and see how many scans the session has acquired

```
startForeground(s);  
s.ScansAcquired
```



ans =

1000

## **See Also**

### **Properties**

NumberOfScans, ScansOutputByHardware

### **Functions**

startBackground

## ScansOutputByHardware

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 `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.

### Example

#### Display Scans Output by Hardware

Generate data on an analog output channel and to see how many scans are output by the hardware.

Create a session and add an analog output channel.

```
s = daq.createSession('ni');  
ch = addAnalogOutputChannel(s, 'Dev1', 'ao1', 'voltage');
```

Queue some output data and start the generation.

```
s.queueOutputData(linspace(-1, 1, 1000)');  
startForeground(s);
```

Examine the `ScansOutputByHardware` property.

s.ScansOutputByHardware

ans =

1000

## See Also

### Properties

ScansAcquired, ScansQueued

### Functions

queueOutputData, startBackground

# ScansQueued

Indicate number of scans queued for output

## Description

In the session-based interface, the `ScansQueued` property displays the number of scans queued for output `queueOutputData`. The `ScansQueued` property increases when you successfully call `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.

## Example

### Display Scans Queued

Queue some output data to an analog output channel and examine the session properties to see how many scans are queued.

Create a session and add an analog output channel.

```
s = daq.createSession('ni');  
ch = addAnalogOutputChannel(s, 'Dev1', 'ao1', 'voltage');
```

Queue some output data and call the `ScansQueued` property to see number of data queued.

```
s.queueOutputData(linspace(-1,1,1000)');  
s.ScansQueued  
  
s.ScansQueued
```

ans =

1000

## See Also

### Properties

ScansOutputByHardware

### Functions

queueOutputData

# Sensitivity

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:

```
ch1 = s.Channels(1)
ch1.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

`addAnalogInputChannel`

# ShuntLocation

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.

## Example

### Specify Shunt Location

Set the shunt location of an analog input current channel.

Create a session and add an analog input current channel.

```
s = daq.createSession('ni')
ch = addAnalogInputChannel(s, 'cDAQ1Mod7', 0, 'Current');
```

Set the `ShuntLocation` to `Internal`.

```
ch.ShuntLocation = 'Internal'
```

```
ch =
```

```
Data acquisition analog input current channel 'ai0' on device 'cDAQ1Mod7':
```

```
    ShuntLocation: Internal
ShuntResistance: 20
    Coupling: DC
TerminalConfig: Differential
```



Range: -0.025 to +0.025 A  
Name: ''  
ID: 'ai0'  
Device: [1x1 daq.ni.CompactDAQModule]  
MeasurementType: 'Current'  
ADCTimingMode: HighResolution

## See Also

ShuntResistance

# ShuntResistance

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.

---

## Example

### Specify Shunt Resistance

Set the shunt resistance of an analog input current channel.

Create a session and add an analog input current channel.

```
s = daq.createSession('ni')
ch = addAnalogInputChannel(s, 'cDAQ1Mod7', 0, 'Current');
```

Set the `ShuntLocation` to `External` and the `ShuntResistance` to 20.

```
ch.ShuntLocation = 'External';
ch.ShuntResistance = 20
```

```
ch =
```

```
Data acquisition analog input current channel 'ai0' on device 'cDAQ1Mod7':
```

```
    ShuntLocation: External
ShuntResistance: 20
    Coupling: DC
TerminalConfig: Differential
    Range: -0.025 to +0.025 A
    Name: ''
```

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

## **See Also**

ShuntLocation

# Source

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.

# Example

## View Clock Connection Source

Create an external clock connection and view the connection properties.

Create a session and add a digital input channel.

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

Add an external scan clock connection.

```
s.addClockConnection('External', 'Dev1/PFI0', 'ScanClock')
```

```
ans =
```

Scan Clock is provided externally and will be received by 'Dev1' at terminal 'PFI0'.

```
Source: 'External'  
Destination: 'Dev1/PFI0'  
Type: ScanClock
```

# See Also

[DestinationaddTriggerConnection](#)

# StandardSampleRates

Display standard rates of sampling

## Description

This property displays the standard sample rates supported by your audio device. You can choose to use the standard rates or use values within the given range. See `UseStandardSampleRates` for more information.

Standard sample rates for DirectSound audio devices are:

- 8000
- 8192
- 11025
- 16000
- 22050
- 32000
- 44100
- 47250
- 48000
- 50000
- 88200
- 96000
- 176400
- 192000
- 352800

## Example

### Set Rate of an Audio Session

Specify a nonstandard sample rate for a session with multichannel audio devices.

Create a session and add an audio channel.

```
s = daq.createSession('directsound')
ch = addAudioInputChannel(s, 'Audio1', 1);
```

Specify the session to use nonstandard sample rates.

```
s.UseStandardSampleRates = false
```

```
Data acquisition session using DirectSound hardware:
```

```
Will run for 1 second (44100 scans) at 44100 scans/second.
```

```
Number of channels: 1
```

index	Type	Device	Channel	MeasurementType	Range	Name
1	audi	Audio1	1	Audio	-1.0 to +1.0	

Change the session rate to 85000.

```
s.Rate = 85000
```

```
s =
```

```
Data acquisition session using DirectSound hardware:
```

```
Will run for 1 second (85000 scans) at 85000 scans/second.
```

```
Number of channels: 1
```

index	Type	Device	Channel	MeasurementType	Range	Name
1	audi	Audio1	1	Audio	-1.0 to +1.0	

### See Also

[BitsPerSample](#) | [Rate](#) | [UseStandardSampleRates](#) | [addAudioInputChannel](#) | [addAudioOutputChannel](#)

## **Topics**

“Multichannel Audio Session Rate”

# Terminal

PFI terminal of counter subsystem

## Description

The `Terminal` property indicates the counter subsystem's corresponding PFI terminal.

## Example

### Determine Counter Input Channel Terminal

Determine the terminal on the counter channel connected to your input signal.

Create a session and add a counter input channel.

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

Examine the `Terminal` property of the channel.

```
ch.Terminal
```

```
ans =
```

```
PFI1
```

## See Also

`addCounterInputChannel`, `addCounterOutputChannel`



# TerminalConfig

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
- SingleEndedNonReferenced
- Differential
- PseudoDifferential

## Example

### Change Analog Channel Terminal Configuration

Change the terminal configuration of an analog input channel.

Create a session and add an analog input voltage channel.

```
s = daq.createSession('ni');  
ch = addAnalogInputChannel(s, 'dev5', 0, 'voltage')
```

```
ch =
```

```
Data acquisition analog input voltage channel 'ai0' on device 'Dev5':
```

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

Change the `TerminalConfig` of the channel to `SingleEnded`.

```
ch.TerminalConfig = 'SingleEnded'
```

```
ch =
```

```
Data acquisition analog input voltage channel 'ai0' on device 'Dev5':
```

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

### See Also

[addAnalogInputChannel](#) | [addAnalogOutputChannel](#)

## Terminals

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

#### Display Device Terminals

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

### Functions

daq.getDevices, addTriggerConnection, addClockConnection

# ThermocoupleType

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, according to the NIST Thermocouple Types Definitions.

## Values

You can set the `ThermocoupleType` to:

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

By default the thermocouple type is 'Unknown'.

## Example

### Specify Thermocouple Type

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

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

ch =

Data acquisition analog input thermocouple channel 'ai1' on device 'cDAQ1Mod6':

    Units: Celsius
ThermocoupleType: Unknown
    Range: -210 to +1200 Celsius
    Name: ''
    ID: 'ai1'
    Device: [1x1 daq.ni.CompactDAQModule]
MeasurementType: 'Thermocouple'
ADCTimingMode: HighResolution
```

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

```
ch.ThermocoupleType = 'J'

ch =

Data acquisition analog input thermocouple channel 'ai1' on device 'cDAQ1Mod6':

    Units: Celsius
ThermocoupleType: J
    Range: -210 to +1200 Celsius
    Name: ''
    ID: 'ai1'
    Device: [1x1 daq.ni.CompactDAQModule]
MeasurementType: 'Thermocouple'
ADCTimingMode: HighResolution
```

## See Also

### Functions

`addAnalogInputChannel`

### External Websites

NIST ITS-90 Thermocouple Database

# TriggerCondition

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 “Synchronization”.

## Values

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

## Examples

### Specify Session Connection Trigger Condition

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

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

Change the trigger condition to `FallingEdge`.

```
connection = s.Connections(1)
connection.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
```

Click on **(Details)** to see the connection details.

Start Trigger is provided by 'Dev1' at 'PFI4' and will be received by 'Dev2' at terminal 'PFI0'.

```
TriggerType: 'Digital'
TriggerCondition: FallingEdge
Source: 'Dev1/PFI4'
Destination: 'Dev2/PFI0'
Type: StartTrigger
```

## See Also

`addTriggerConnection`

## Properties

`TriggerType`



# TriggersPerRun

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.

## Examples

### Specify Number of Triggers Per Operation

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

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

Display Session's `TriggersPerRun` Property.

```
s.TriggersPerRun
```

```
ans =
```

```
1
```

Set the trigger to run twice during the operation.

```
s.TriggersPerRun = 2
```

```
s =
```

```
Data acquisition session using National Instruments hardware:  
Will run 2 times 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

`addTriggerConnection`

# TriggersRemaining

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`.

## Examples

### Display Number of Triggers Remaining in Operation

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

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

Display Session's `TriggersRemaining` Property.

```
s.TriggersRemaining
```

```
ans =
```

```
1
```

## See Also

`addTriggerConnection`

## **TriggerType**

Type of trigger executed

### **Description**

This read-only property displays the type of trigger that the source device executes to synchronize operations in the session. Currently all trigger types are **digital**.

### **See Also**

#### **Functions**

`addTriggerConnection`

#### **Properties**

`TriggerCondition`

## Units

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

## Example

### Change RTD Unit

Change the unit of an RTD channel.

Create a session, add an analog input RTD channel, and display channel properties.

```
s = daq.createSession('ni');  
ch = addAnalogInputChannel(s, 'cDAQ1Mod7', 0, 'RTD')
```

```
ch =
```

```
Data acquisition analog input RTD channel 'ai0' on device 'cDAQ1Mod7':
```

```
          Units: Celsius  
          RTDType: Unknown  
    RTDConfiguration: Unknown  
                R0: 'Unknown'  
ExcitationCurrent: 0.0005  
ExcitationSource: Internal
```

```
Coupling: DC
TerminalConfig: Differential
  Range: -200 to +660 Celsius
  Name: ''
  ID: 'ai0'
  Device: [1x1 daq.ni.CompactDAQModule]
MeasurementType: 'RTD'
ADCTimingMode: HighResolution
```

Change the Units property from Celsius to Fahrenheit.

```
ch.Units = 'Fahrenheit'
```

```
ch =
```

Data acquisition analog input RTD channel 'ai0' on device 'cDAQ1Mod7':

```
Units: Fahrenheit
RTDType: Unknown
RTDConfiguration: Unknown
  R0: 'Unknown'
ExcitationCurrent: 0.0005
ExcitationSource: Internal
  Coupling: DC
  TerminalConfig: Differential
  Range: -328 to +1220 Fahrenheit
  Name: ''
  ID: 'ai0'
  Device: [1x1 daq.ni.CompactDAQModule]
MeasurementType: 'RTD'
ADCTimingMode: HighResolution
```

## See Also

### Class

`addAnalogInputChannel`

# UserData

Custom data

## Description

Manipulate custom data for a data acquisition session object using the `UserData` property. The property is never read-only. Its value can be any MATLAB data type and format.

## Examples

Create a session and define its `UserData` property fields.

```
s = daq.createSession('ni');  
s.UserData.Data = [];  
s.UserData.TimeStamps = [];  
s.UserData.StartTime = [];
```

Set the start time, and append event information to the log fields stored in `UserData`.

```
s.UserData.StartTime = eventData.TriggerTime;  
s.UserData.Data = [s.UserData.Data; eventData.Data];  
s.UserData.TimeStamps = [s.UserData.TimeStamps; eventData.TimeStamps];
```

## See Also

### Functions

`daq.createSession`

## UseStandardSampleRates

Configure session to use standard sample rates

### Description

Use this property to specify if your audio channel uses standard sample rates supported by your device or a user-specified value. To use non-standard sample rates, set the value to `false` and set the session's `Rate` to the desired value.

### Example

#### Change Acquisition Rate

Add an audio channel to a session and change the `UseStandardSampleRates` property.

```
s = daq.createSession('directsound');
addAudioInputChannel(s,Audio1,1);
s.UseStandardSampleRates = false
```

```
s =
```

```
Data acquisition session using DirectSound hardware:
```

```
Will run for 1 second (44100 scans) at 44100 scans/second.
```

```
Number of channels: 1
```

index	Type	Device	Channel	MeasurementType	Range	Name
1	audi	Audio1	1	Audio	-1.0 to +1.0	

Specify a different scan rate.

```
s.Rate = 8500
```

```
s =
```

```
Data acquisition session using DirectSound hardware:
```

```
Will run for 1 second (8500 scans) at 8500 scans/second.
```

```
Number of channels: 1
```



index	Type	Device	Channel	MeasurementType	Range	Name
1	audi	Audio3	1	Audio	-1.0 to +1.0	

## See Also

Rate | StandardSampleRates | addAudioInputChannel | addAudioOutputChannel

## Topics

“Multichannel Audio Session Rate”

# Vendor

Vendor information associated with session object

## Description

In the session-based interface, the `Vendor` property displays information about the vendor.

## Values

a `daq.Vendor` object that represents the vendor associated with the session.

## 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 support packages. Open the Support Package Installer to install additional vendors.

## See Also

`daq.createSession`

# WaveformType

Function generator channel waveform type

## Description

This read-only property displays the channel waveform type that you specified while creating a function generator channel in a session. Supported waveform types are:

- 'Sine'
- 'Square'
- 'Triangle'
- 'RampUp'
- 'RampDown'
- 'DC'
- 'Arbitrary'

## Example

Display the channel's waveform type.

```
fgenCh.WaveformType
```

```
ans =
```

```
    Sine
```

# ZResetCondition

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'

## Example

### Change Counter Channel Z Reset Condition

Create a session and add a counter input Position channel.

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

```
ch =
```

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

```
EncoderType: X1  
ZResetEnable: 0  
ZResetValue: 0  
ZResetCondition: BothHigh  
TerminalA: 'PFI0'  
TerminalB: 'PFI2'  
TerminalZ: 'PFI1'  
Name: ''  
ID: 'ctr0'  
Device: [1x1 daq.ni.CompactDAQModule]  
MeasurementType: 'Position'
```

Change the ZResetCondition to BothLow.

```
ch.ZResetCondition = 'BothLow'
```

```
ch =
```

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

```
EncoderType: X1
ZResetEnable: 0
ZResetValue: 0
ZResetCondition: BothLow
TerminalA: 'PFI0'
TerminalB: 'PFI2'
TerminalZ: 'PFI1'
Name: ''
ID: 'ctr0'
Device: [1x1 daq.ni.CompactDAQModule]
MeasurementType: 'Position'
```

## See Also

[addCounterInputChannel](#)

# ZResetEnable

Enable reset for Z-indexing

## Description

Use the ZResetEnable property to allow the Z-indexing to be reset on a counter input 'Position' channel.

## Example

### Reset Z Indexing on Counter Channel

Create a session and add a counter input Position channel.

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

```
ch =
```

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

```
    EncoderType: X1  
    ZResetEnable: 0  
    ZResetValue: 0  
ZResetCondition: BothHigh  
    TerminalA: 'PFI0'  
    TerminalB: 'PFI2'  
    TerminalZ: 'PFI1'  
    Name: ''  
    ID: 'ctr0'  
    Device: [1x1 daq.ni.CompactDAQModule]  
MeasurementType: 'Position'
```

Change the ZResetEnable property value to 1.

```
ch.ZResetEnable = 1
```

```
ch =
```

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

```
EncoderType: X1
ZResetEnable: 1
ZResetValue: 0
ZResetCondition: BothHigh
  TerminalA: 'PFI0'
  TerminalB: 'PFI2'
  TerminalZ: 'PFI1'
  Name: ''
  ID: 'ctr0'
  Device: [1x1 daq.ni.CompactDAQModule]
MeasurementType: 'Position'
```

## See Also

### Class

`addCounterInputChannel`



# ZResetValue

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.

## Example

### Specify Z Indexing Value

Create a session and add a counter input `Position` channel.

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

```
ch =
```

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

```
    EncoderType: X1
    ZResetEnable: 0
    ZResetValue: 0
ZResetCondition: BothHigh
    TerminalA: 'PFI0'
    TerminalB: 'PFI2'
    TerminalZ: 'PFI1'
    Name: ''
    ID: 'ctr0'
    Device: [1x1 daq.ni.CompactDAQModule]
MeasurementType: 'Position'
```

Change the `ZResetValue` to 62.

```
ch.ZResetValue = 62
```

```
ch =
```

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

```
EncoderType: X1
ZResetEnable: 1
ZResetValue: 62
ZResetCondition: BothHigh
  TerminalA: 'PFI0'
  TerminalB: 'PFI2'
  TerminalZ: 'PFI1'
  Name: ''
  ID: 'ctr0'
  Device: [1x1 daq.ni.CompactDAQModule]
MeasurementType: 'Position'
```

## See Also

### Class

`addCounterInputChannel`

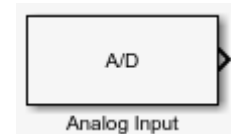
# Block Reference

---

## Analog Input

Acquire data from multiple analog channels of data acquisition device

**Library:** Data Acquisition Toolbox



### Description

The Analog Input block opens, initializes, configures, and controls an analog data acquisition device. The opening, initialization, and configuration of the device occur once at the start of the model execution. During the model run time, the block acquires data either synchronously (deliver the current block of data the device is providing) or asynchronously (stream buffered incoming data).

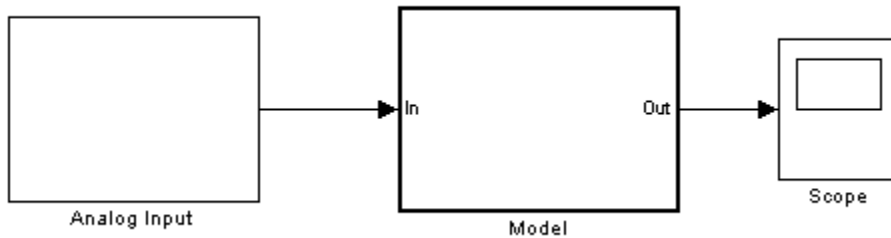
The block has no input ports. It has one or more output ports, depending on the configuration you choose in its dialog box.

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
- Controller design

The following diagram shows the basic analog input usage configuration, with which you can:

- Read acquired 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.



---

**Notes** To use this block, you need both Data Acquisition Toolbox and Simulink software.

Some devices are not supported by the Simulink blocks in Data Acquisition Toolbox. To see if your device supports Simulink, refer to Supported Hardware.

You can use the Analog Input block only with devices that support clocked acquisition. To acquire data using devices that do not support clocking, use the Analog Input (Single Sample) block.

---

## Other Supported Features

- If you have DSP System Toolbox™, you can use this block for signal applications.
- This block supports the use of Simulink Accelerator™ mode, but not Rapid Accelerator or code generation.
- The block supports the use of model referencing, so that your model can include other Simulink models as modular components.

For more information on these features, see the “Simulink” documentation.

## Ports

### Output

**Data — Acquired analog input**

double

Acquired analog input data, returned as doubles. If using only one output port for all channels, each scan is available as a matrix of scan blocksize by number of channels, M-by-N. If using a port for each channel, each scan results in a blocksize-by-1 column vector on each port. Multiple ports are named by channel names or device specified channel IDs.

Data Types: double

### **Relative timestamp – Relative timestamps of scans**

Relative timestamp of each scan, returned as a double. This port is available when you check the Output relative timestamps on page 3-0 parameter.

Data Types: double

## **Parameters**

Use the Block Parameters dialog box to select your acquisition mode and to set other configuration options.

### **Device – Device from which you want to acquire data**

The 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 or vendor name and unique device ID, followed by the model name of the device, for example, ni Dev1 (USB-6255). The first available device is selected by default. A CompactDAQ chassis would be shown as a single device identified by vendor name, chassis ID, and chassis model; for example, ni cDAQ2 (cDAQ-9172).

### **Acquisition Mode – Synchronous setting**

**Asynchronous** | **Synchronous**

Synchronous setting, specified as one of the following options.

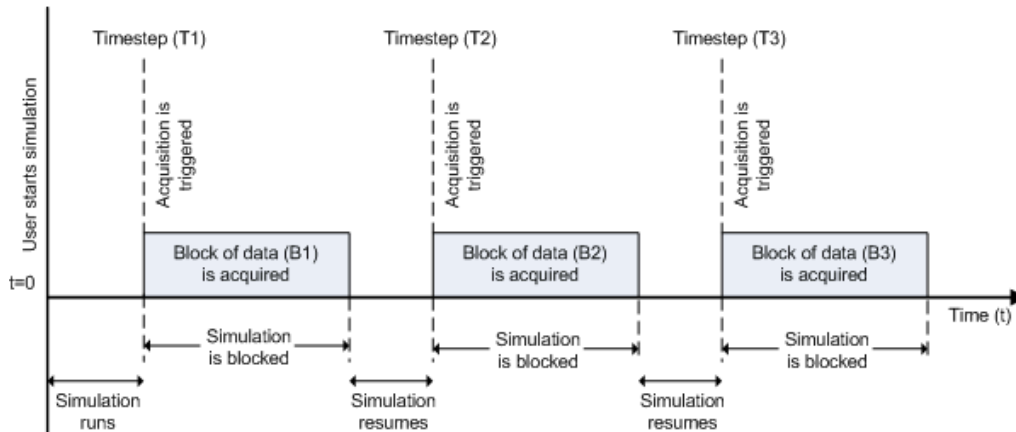
**Asynchronous** — In asynchronous mode, the data acquisition from the device and the simulation happen in parallel. The model initiates the acquisition from the device when the simulation starts. Data from the device is continuously acquired into a FIFO (first in, first out) buffer in parallel as the simulation runs. At each time step, the model fetches data from the FIFO buffer and outputs a block of data. The data in the FIFO buffer is contiguous according to the hardware acquisition clock.

**Synchronous** — In synchronous mode, the simulation is blocked while acquiring data from the device. The model initiates the acquisition from the device at each time step and

immediately enters a wait state until the acquisition request has completed. This is unbuffered input; the block outputs the latest block of data at each time step.

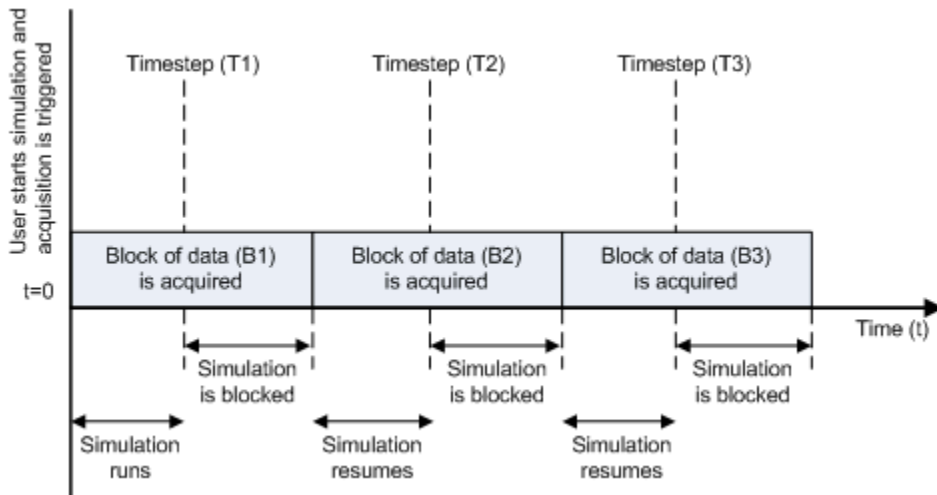
The following diagrams show the difference between synchronous and asynchronous modes for the Analog Input block.

### Synchronous Analog Input



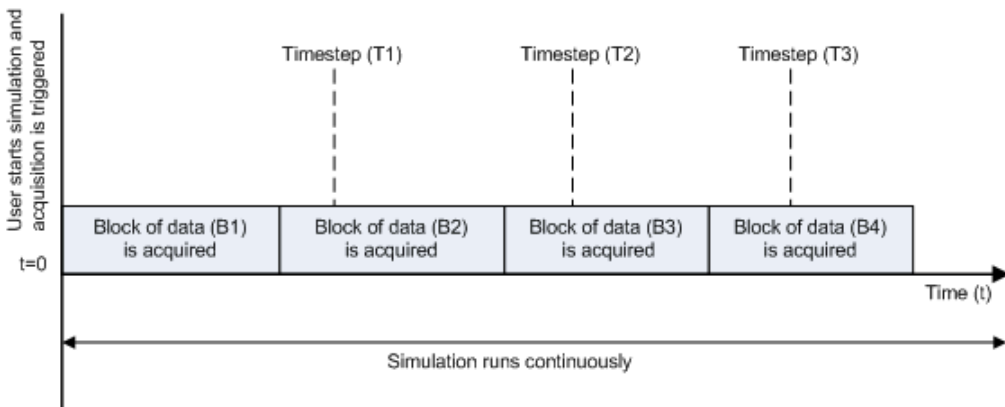
At the first time step (T1), the acquisition is initiated for the required block of data (B1). The simulation does not continue until B1 is completely acquired.

### 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.

---

### **Channels — Device channel selection and configuration**

options depend on device

Device channel selection and configuration table. The channel configuration table lists the hardware channels of your device, and lets you select and configure them. Specify which channels to acquire data from (by default all the channels are selected). The following parameters are specified for each selected channel:

**Channel ID** — Hardware channel ID specified by the device. The Channel ID column is read-only, and the parameters are defined when the device is selected.

**Name** — 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`.

**Module** — Device ID the channel belongs to. The Module column is read-only. If a CompactDAQ chassis is selected, it shows the ID of the CompactDAQ module which the channel belongs to; otherwise the ID of the device.

**Measurement Type** — Measurement type of the channel. This block supports only voltage measurement types. (For other measurement types, use a data acquisition session in MATLAB.)

**Input Range** — Input ranges available for each channel supported by the hardware, defined when a device is selected.

**Terminal Configuration** — Specifies the hardware terminal configuration, such as single-ended, differential, etc. The terminal configuration options are defined by the capabilities of the selected channel.

**Coupling** — Hardware coupling configuration, such as AC or DC. The coupling type is defined when a device is selected

**Number of ports — Number of output data ports**  
**1 for all channels | 1 per channel**

Number of output data ports, specified as:

**1 for all channels** — Output data from a single port as a matrix, with a size of blocksize by number of channels selected.

**1 per channel** — Output data from N ports, where N is equal to the number of selected channels. Each output port is a column vector with a size of blocksize-by-1. For naming, each output port uses the channel name if one was specified, otherwise the channel ID, for example, ai0.

**Input sample rate — Device sampling rate**

numeric value

The rate at which samples are acquired from the device, in samples per second. This is the sampling rate for the hardware. The sample rate must be a positive real number within the range supported by the selected hardware.

**Block size — Number of scans per time step**

integer value

The number of data samples to read from the block output at each time step for each channel. It must be a positive integer greater than or equal to 2, within the range supported by the selected hardware.

**Output relative timestamps — Add timestamp output port**

Select this option to output the relative data timestamps, one for each sample. This option adds a new output port to the block. The data type of this port is double, and corresponds to the time offset in seconds of the sample related to the start of acquisition. For asynchronous acquisition, the acquisition is initiated once at the start of model execution, the relative timestamp is a monotonically-increasing number relative to the start of simulation. For synchronous acquisition, an acquisition is initiated at every time step; as a result, the relative timestamp is reset to zero every time an acquisition is initiated.

## See Also

### Blocks

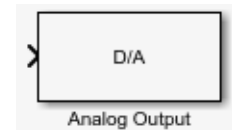
Analog Input (Single Sample) | Analog Output | Analog Output (Single Sample) | Digital Input (Single Sample) | Digital Output (Single Sample)

**Introduced in R2016b**

# Analog Output

Output data to multiple analog channels of data acquisition device

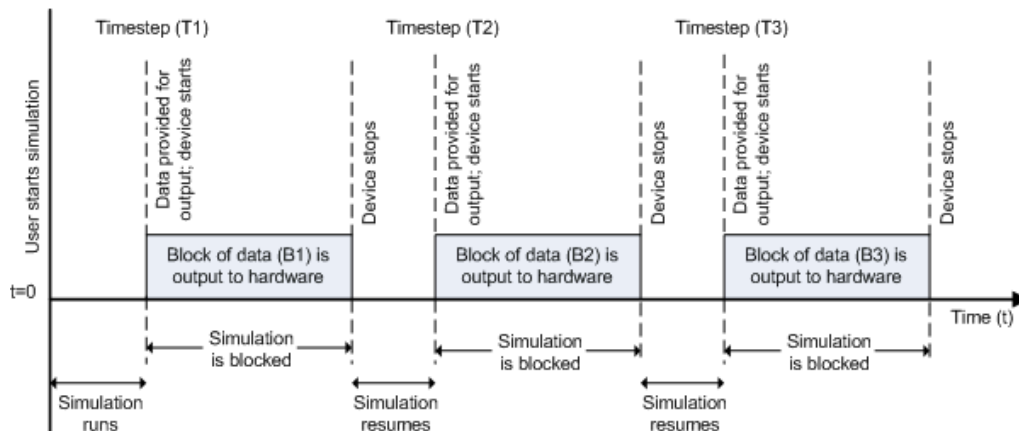
**Library:** Data Acquisition Toolbox



## Description

The Analog Output block opens, initializes, configures, and controls an analog data acquisition device. The opening, initialization, and configuration of the device occur once at the start of the model execution. During the model run time, the block outputs data to the hardware synchronously (outputs the block of data as it is provided). On every time step, the block performs a blocking synchronous write to the hardware, outputting the entire input data.

The following diagram shows the timing of the 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.

The block has one or more input ports, depending on the option you choose in its parameters dialog box. It has no output ports.

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.

---

**Notes** To use this block, you need both Data Acquisition Toolbox and Simulink software.

You can use the Analog Output block only with devices that support clocked generation. To generate data using devices that do not support clocking, use the Analog Output (Single Sample) block.

Some devices are not supported by the Simulink blocks in Data Acquisition Toolbox. To see if your device supports Simulink, refer to Supported Hardware.

---

### Other Supported Features

- This block supports the use of Simulink Accelerator mode, but not Rapid Accelerator or code generation.
- The block supports the use of model referencing, so that your model can include other Simulink models as modular components.

For more information on these features, see the “Simulink” documentation.

## Ports

### Input

#### Data — Analog output to generate

double

Analog output to generate, specified as doubles. If using only one input port for all channels, specify an M-by-N matrix for a blocksize of M scans on N channels. Each scan is a row across N channels. Each channel outputs a column of M scans.

If using a port for each channel, specify a column of data for each channel on each port. Multiple ports are named by channel names or device specified channel IDs.

Data Types: double

## Parameters

### **Device — Device through which you want to output data**

select available device

The device from which you want to generate 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 model name of the device, for example, `ni Dev1 (USB-6255)`. The first available device is selected by default. A CompactDAQ chassis is shown as a single device; vendor name, chassis ID, and chassis model would be shown in the list, for example, `ni cDAQ2 (cDAQ-9172)`.

### **Channels — Device channel selection and configuration**

options depend on device

Device channel selection and configuration table. The channel configuration table lists the hardware channels of your device, and lets you select and configure them. Specify which channels to acquire data from (by default all the channels are selected). The following parameters are specified for each selected channel:

**Channel ID** — Hardware channel ID specified by the device. The Channel ID column is read-only, and the parameters are defined when the device is selected.

**Name** — 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`.

**Module** — Device ID the channel belongs to. The Module column is read-only. If a CompactDAQ chassis is selected, it shows the ID of the CompactDAQ module which the channel belongs to; otherwise the ID of the device.

**Measurement Type** — Measurement type of the channel. This block supports only voltage measurement types. (For other measurement types, use a data acquisition session in MATLAB.)

**Output Range** — Output ranges available for each channel supported by the hardware, defined when a device is selected.

### **Number of ports — Number of input data ports** **1 for all channels | 1 per channel**

Number of input data ports, specified as:

**1 for all channels** (default) — One input port on the block for all channels. Provide data as a matrix, with a size of scan blocksize by number of channels, M-by-N.

**1 per channel** — N input ports on the block, where N is equal to the number of selected channels. Provide each port data as a column vector with a size of blocksize-by-1. For naming, each output port uses the channel name if one was specified, otherwise the channel ID, for example, ao1.

### **Output sample rate — Device sampling rate** numeric value

The rate at which samples are output from Simulink to the device, in samples per second. This is the sampling rate for the hardware. The default is defined when a device is selected. The sample rate must be a positive real number within the range allowed for the selected hardware.

## See Also

### **Blocks**

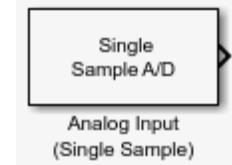
Analog Input | Analog Input (Single Sample) | Analog Output (Single Sample) | Digital Input (Single Sample) | Digital Output (Single Sample)

**Introduced in R2016b**

## Analog Input (Single Sample)

Acquire single sample from multiple analog channels of data acquisition device

**Library:** Data Acquisition Toolbox



### Description

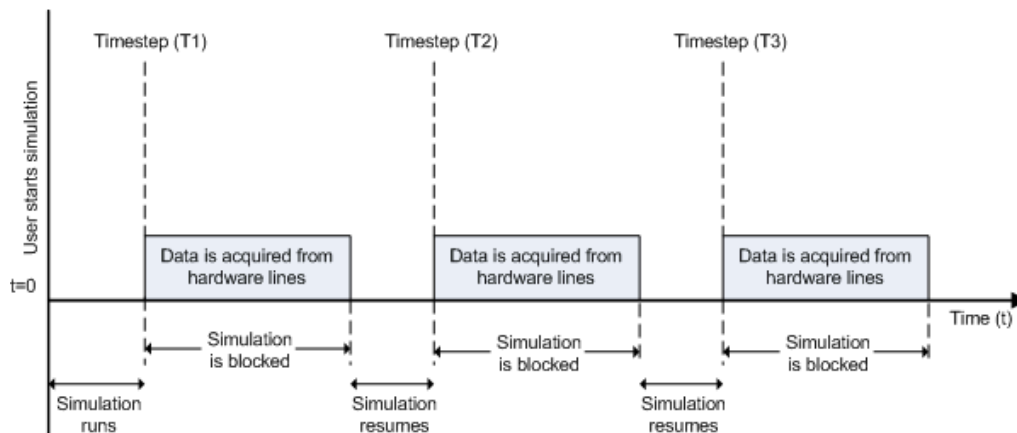
The Analog Input (Single Sample) block opens, initializes, configures, and controls an analog data acquisition device. The opening, initialization, and configuration of the device occur once at the start of the model execution. The block acquires a single sample every time step, synchronously from the device, during the model run time.

The block has no input ports. It has one or more output ports, depending on the configuration you choose in its dialog box.

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
- Controller design

Analog input acquisition is done synchronously, according to the following diagram.



At the first time step (T1), data is acquired from the selected hardware channels. The simulation does not continue until data is read from all channels.

---

**Notes** To use this block, you need both Data Acquisition Toolbox and Simulink software.

Some devices are not supported by the Simulink blocks in Data Acquisition Toolbox. To see if your device supports Simulink, refer to Supported Hardware.

You can use Analog Input (Single Sample) block only with devices that support single sample acquisition. If the device does not support single sample acquisition, the model generates an error. To acquire data from devices that do not support acquisition of a single sample (such as devices designed for sound and vibration), use the Analog Input block.

---

### Other Supported Features

- If you have DSP System Toolbox, you can use this block for signal applications.
- This block supports the use of Simulink Accelerator mode, but not Rapid Accelerator or code generation.
- The block supports the use of model referencing, so that your model can include other Simulink models as modular components.

For more information on these features, see the “Simulink” documentation.



## Ports

### Output

#### Data — Acquired analog input

double

Acquired analog input data, returned as doubles. If using only one output port for all channels, the output is an array of data. If using a port for each channel, each scan results in a single value on each port. Multiple ports are named by channel names or device specified channel IDs.

Data Types: double

#### Timestamp — Timestamp of scan

Timestamp of scan, returned as a double. This port is available when you check the Output timestamp on page 3-0 parameter.

Data Types: double

## Parameters

Use the Block Parameters dialog box to select your device and to set other configuration options.

#### Device — Device from which you want to acquire data

The 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 or vendor name and unique device ID, followed by the model name of the device, for example, ni\_Dev1 (USB-6255). The first available device is selected by default. A CompactDAQ chassis would be shown as a single device identified by vendor name, chassis ID, and chassis model; for example, ni\_cDAQ2 (cDAQ-9172).

#### Channels — Device channel selection and configuration

options depend on device

Device channel selection and configuration table. The channel configuration table lists the hardware channels of your device, and lets you select and configure them. Specify which

channels to acquire data from (by default all the channels are selected). The following parameters are specified for each selected channel:

**Channel ID** — Hardware channel ID specified by the device. The Channel ID column is read-only, and the parameters are defined when the device is selected.

**Name** — 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.

**Module** — Device ID the channel belongs to. The Module column is read-only. If a CompactDAQ chassis is selected, it shows the ID of the CompactDAQ module which the channel belongs to; otherwise the ID of the device.

**Measurement Type** — Measurement type of the channel. This block supports only voltage measurement types. (For other measurement types, use a data acquisition session in MATLAB.)

**Input Range** — Input ranges available for each channel supported by the hardware, defined when a device is selected.

**Terminal Configuration** — Specifies the hardware terminal configuration, such as single-ended, differential, etc. The terminal configuration options are defined by the capabilities of the selected channel.

**Coupling** — Hardware coupling configuration, such as AC or DC. The coupling type is defined when a device is selected

**Number of ports — Number of output data ports**  
**1 for all channels | 1 per channel**

Number of output data ports, specified as:

**1 for all channels** — Outputs the acquired data from a single port as a 1-by-N vector with a length equal to the number of channels selected.

**1 per channel** — Outputs the acquired data from N ports, where N is equal to the number of selected channels. Each port output is a 1-by-1 double. For naming, each output port uses the channel name if one was specified, otherwise the channel ID, for example, ai0.

**Sample time — Block execution rate**  
1 (default)

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 1 (seconds). For more information, see “What Is Sample Time?” (Simulink).

### **Output timestamp — Add timestamp output port**

Select this option to output the absolute timestamp of the scan. This option adds a new output port to the block. The data type of this port is double (datenum), which corresponds to a serial date number. You can convert the datenum into a datetime value with the `datetime` function.

## **See Also**

### **Blocks**

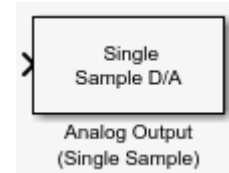
Analog Input | Analog Output | Analog Output (Single Sample) | Digital Input (Single Sample) | Digital Output (Single Sample)

### **Introduced in R2016b**

## Analog Output (Single Sample)

Output single sample to multiple analog channels of data acquisition device

**Library:** Data Acquisition Toolbox

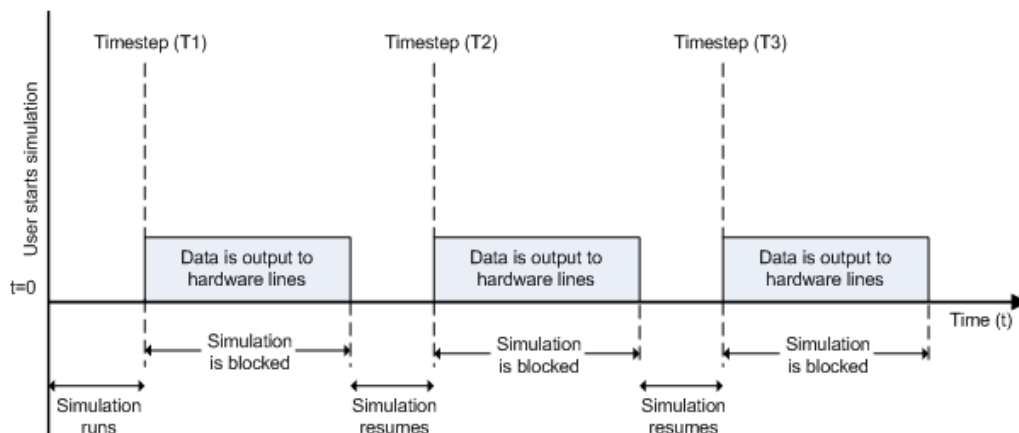


### Description

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 execution. The block outputs a single sample every time step, synchronously to the hardware, during the model run time.

The block has one or more input ports, depending on the option you choose in its dialog box. It has no output ports. The valid data type of the signal at the input port is double.

The Analog Output (Single Sample) block inherits the sample time from the driving block connected to the input port. Analog output is done synchronously, according to the following diagram.



At the first time step (T1), data is output to the selected hardware channels. The simulation does not continue until data is output to all channels.

---

**Notes** To use this block, you need both Data Acquisition Toolbox and Simulink software.

You can use the Analog Output (Single Sample) block only with devices that support single sample output. To send data using devices that do not support acquisition of a single sample (such as devices designed for sound and vibration), use the Analog Output block.

Some devices are not supported by the Simulink blocks in Data Acquisition Toolbox. To see if your device supports Simulink, refer to Supported Hardware.

---

## Other Supported Features

- This block supports the use of Simulink Accelerator mode, but not Rapid Accelerator or code generation.
- The block supports the use of model referencing, so that your model can include other Simulink models as modular components.

For more information on these features, see the “Simulink” documentation.

## Ports

### Input

#### Data — Analog output to generate

double

Analog output to generate, specified as doubles. If using only one input port for all channels, provide a 1-by-N vector for a single scan on all N channels.

If using a port for each channel, provide a double value to each port. Multiple ports are named by channel names or device specified channel IDs.

Data Types: double

## Parameters

### **Device — Device through which you want to output data**

select available device

The device from which you want to generate 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 model name of the device, for example, `ni Dev1 (USB-6255)`. The first available device is selected by default. A CompactDAQ chassis is shown as a single device; vendor name, chassis ID, and chassis model would be shown in the list, for example, `ni cDAQ2 (cDAQ-9172)`.

### **Channels — Device channel selection and configuration**

options depend on device

Device channel selection and configuration table. The channel configuration table lists the hardware channels of your device, and lets you select and configure them. Specify which channels to acquire data from (by default all the channels are selected). The following parameters are specified for each selected channel:

**Channel ID** — Hardware channel ID specified by the device. The Channel ID column is read-only, and the parameters are defined when the device is selected.

**Name** — 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`.

**Module** — Device ID the channel belongs to. The Module column is read-only. If a CompactDAQ chassis is selected, it shows the ID of the CompactDAQ module which the channel belongs to; otherwise the ID of the device.

**Measurement Type** — Measurement type of the channel. This block supports only voltage measurement types. (For other measurement types, use a data acquisition session in MATLAB.)

**Output Range** — Output ranges available for each channel supported by the hardware, defined when a device is selected.

### **Number of ports — Number of input data ports**

1 for all channels | 1 per channel

Number of input data ports, specified as:

**1 for all channels** (default) — One input port on the block provides data for all channels. Provide data as a 1-by-N vector for N channels.

**1 per channel** — N input ports on the block, where N is equal to the number of selected channels. Provide data as a double value to each port. For naming, each output port uses the channel name if one was specified, otherwise the channel ID, for example, `ao1`.

**Sample time — Block sample time**

numeric value

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 1. For more information, see “What Is Sample Time?” (Simulink).

## See Also

### Blocks

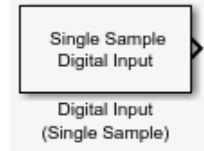
Analog Input | Analog Input (Single Sample) | Analog Output | Digital Input (Single Sample) | Digital Output (Single Sample)

**Introduced in R2016b**

## Digital Input (Single Sample)

Acquire single sample from multiple digital lines of data acquisition device

**Library:** Data Acquisition Toolbox

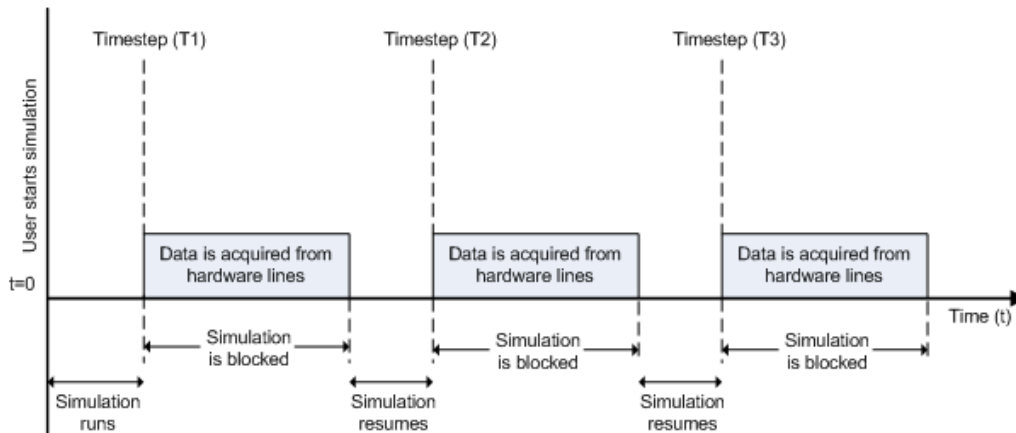


### Description

The Digital Input (Single Sample) 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 delivers this as a vector of boolean values.

The block has no input ports. It has one or more output ports, depending on the option you choose in its dialog box.

The block inherits the sample time of the model. Digital input acquisition is done synchronously, according to the following diagram.



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.



**Note** To use this block, you need both Data Acquisition Toolbox and Simulink software.

Some devices are not supported by the Simulink blocks in Data Acquisition Toolbox. To see if your device supports Simulink, refer to Supported Hardware.

---

## Other Supported Features

- This block supports the use of Simulink Accelerator mode, but not Rapid Accelerator or code generation.
- The block supports the use of model referencing, so that your model can include other Simulink models as modular components.

For more information on these features, see the “Simulink” documentation.

## Ports

### Output

#### **Data — Acquired digital input**

boolean

Acquired digital input data, returned as booleans. If using only one output port for all lines, the output is a 1-by-N vector for N channels. If using a port for each line, each scan results in a single boolean on each port. Multiple ports are named by line names or device specified line IDs.

Data Types: Boolean

#### **Timestamp — Timestamp of scan**

double

Timestamp of scan, returned as a double. This port is available when you check the Output timestamp on page 3-0 parameter.

Data Types: double

## Parameters

### **Device — Device from which you want to acquire data**

The 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 or vendor name and unique device ID, followed by the model name of the device, for example, `ni Dev1 (USB-6255)`. The first available device is selected by default. A CompactDAQ chassis would be shown as a single device identified by vendor name, chassis ID, and chassis model; for example, `ni cDAQ2 (cDAQ-9172)`.

### **Lines — Device line selection and configuration**

options depend on device

**Line ID** — ID of the hardware line (for example, `port0/line0`). This is automatically detected and filled in by the selected device, and is read-only.

**Name** — Hardware line name. This is automatically detected and filled in from the hardware, though you can edit the name.

**Module** — Device ID that the line belongs to. The Module column is read-only. If a CompactDAQ chassis is selected, it shows the ID of the CompactDAQ module which the line belongs to; otherwise the ID of the device.

### **Number of ports — Number of output data ports**

**1 for all lines | 1 per line**

Number of output data ports, specified as:

**1 for all lines** — The block has only one output port for all of the lines that are selected in the table. Acquired data is returned as a 1-by-N vector of boolean values, whose size is the number of lines.

**1 per line** — The block has one output port per selected line. Data is returned as a 1-by-1 boolean value on each port. The name of each output port is the name specified in the table for each line. If no name is provided, the name is the Line ID. For example, if line 2 of hardware port 3 is selected, and you did not specify a name in the line table, `port3/line2` appears in the block. Data size for each line is 1-by-1.

### **Sample time — Block execution rate**

1 (default)

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 1 (seconds). For more information, see “What Is Sample Time?” (Simulink).

### **Output timestamp — Add timestamp output port**

Select this option to output the absolute timestamp of the scan. This option adds a new output port to the block. The data type of this port is double (datenum), which corresponds to a serial date number. You can convert the datenum into a datetime value with the `datetime` function.

## **See Also**

### **Blocks**

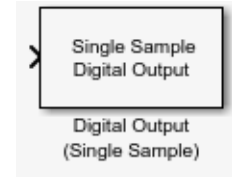
Analog Input | Analog Input (Single Sample) | Analog Output | Analog Output (Single Sample) | Digital Output (Single Sample)

### **Introduced in R2016b**

## Digital Output (Single Sample)

Output single sample to multiple digital lines of data acquisition device

**Library:** Data Acquisition Toolbox

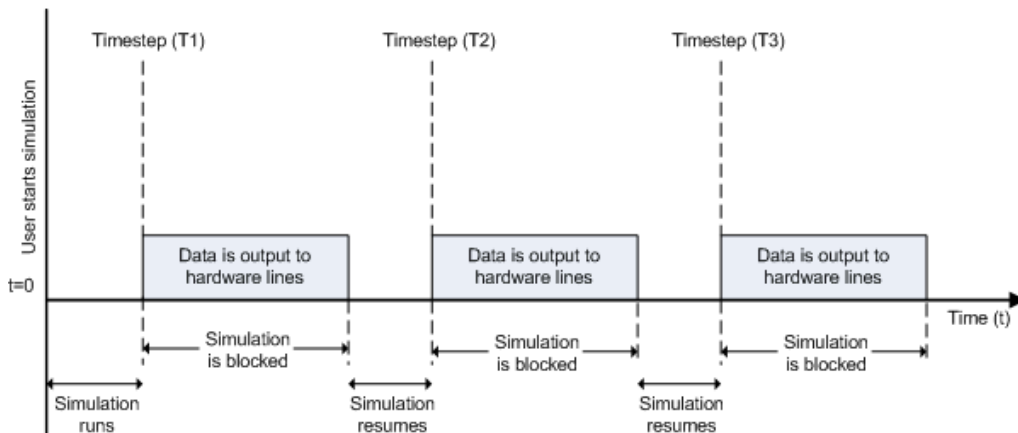


### Description

The Digital Output (Single Sample) block synchronously outputs the latest set of data to the hardware at each simulation time step. It outputs unbuffered digital data. Specify the output data as a vector of boolean values.

The block has no output ports. It can have one or more input ports, depending on the option you choose in its dialog box. The data type of the signal at the input port must be a boolean data type.

The Digital Output (Single Sample) block inherits the sample time from the driving block connected to the input port. Digital output is done synchronously, according to the following diagram.



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.

---

**Note** To use this block, you need both Data Acquisition Toolbox and Simulink software.

Some devices are not supported by the Simulink blocks in Data Acquisition Toolbox. To see if your device supports Simulink, refer to Supported Hardware.

---

## Other Supported Features

- This block supports the use of Simulink Accelerator mode, but not Rapid Accelerator or code generation.
- The block supports the use of model referencing, so that your model can include other Simulink models as modular components.

For more information on these features, see the “Simulink” documentation.

## Ports

### Input

#### Data — Generated digital output data

boolean

Generated digital output data, specified as booleans. If using only one input port for all lines, provide a 1-by-N vector of data. If using a port for each line, provide a single value on each port. Multiple ports are named by line names or device specified line IDs.

Data Types: Boolean

## Parameters

#### Device — Device with which you want to generate data

The 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 or vendor name and unique device ID, followed by the model name of the device, for example, `ni Dev1 (USB-6255)`. The first available device is selected by default. A CompactDAQ chassis would be shown as a single device identified by vendor name, chassis ID, and chassis model; for example, `ni cDAQ2 (cDAQ-9172)`.

### **Lines — Device line selection and configuration**

options depend on device

**Line ID** — ID of the hardware line (for example, `port0/line0`). This is automatically detected and filled in by the selected device, and is read-only.

**Name** — Hardware line name. This is automatically detected and filled in from the hardware, though you can edit the name.

**Module** — Device ID that the line belongs to. The Module column is read-only. If a CompactDAQ chassis is selected, it shows the ID of the CompactDAQ module which the line belongs to; otherwise the ID of the device.

### **Number of ports — Number of input data ports**

**1 for all lines | 1 per line**

Number of input data ports, specified as:

**1 for all lines** — The block has only one input port for all of the lines that are selected in the table. Generated data is defined as a 1-by-N row vector of boolean values, whose size is the number of lines.

**1 per line** — The block has 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 the Line ID. For example, if line 2 of hardware port 3 is selected, and you did not specify a name in the line table, `port3/line2` appears in the block. Data size for each line is 1-by-1.

### **Sample time — Block execution rate**

1 (default)

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 1 (seconds). For more information, see “What Is Sample Time?” (Simulink).

## See Also

### Blocks

[Analog Input](#) | [Analog Input \(Single Sample\)](#) | [Analog Output](#) | [Analog Output \(Single Sample\)](#) | [Digital Input \(Single Sample\)](#)

**Introduced in R2016b**

