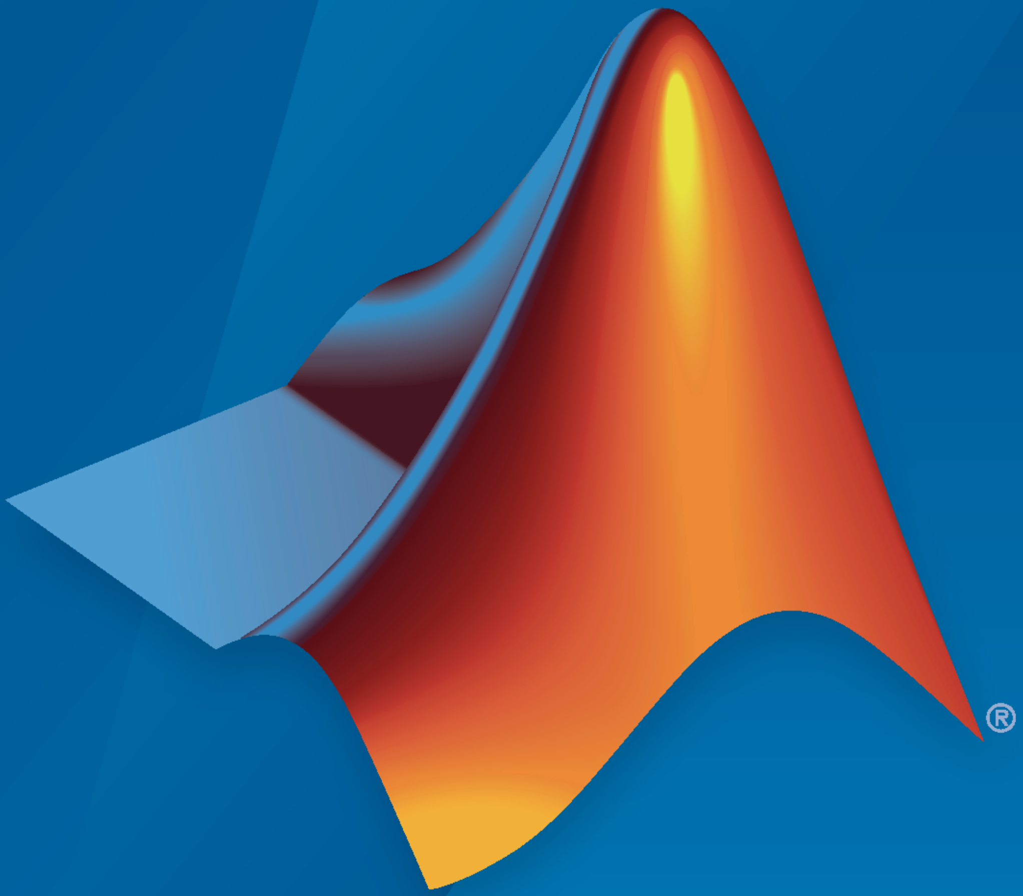
Data Acquisition Toolbox™

Session Interface Reference



MATLAB® & SIMULINK®

R2023a

MathWorks®

How to Contact MathWorks

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

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

Data Acquisition Toolbox™ Session Interface Reference

Revision History

September 2010	Online only	Revised for Version 2.17 (Release 2010b)
April 2011	Online only	Revised for Version 2.18 (Release 2011a)
September 2011	Online only	Revised for Version 3.0 (Release 2011b)
March 2012	Online only	Revised for Version 3.1 (Release 2012a)
September 2012	Online only	Revised for Version 3.2 (Release 2012b)
March 2013	Online only	Revised for Version 3.3 (Release 2013a)
September 2013	Online only	Revised for Version 3.4 (Release 2013b)
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)
March 2020	Online only	Revised for Version 4.1 (Release 2020a)
September 2020	Online only	Revised for Version 4.2 (Release 2020b)
March 2021	Online only	Revised for Version 4.3 (Release 2021a)
September 2021	Online only	Revised for Version 4.4 (Release 2021b)
March 2022	Online only	Revised for Version 4.5 (Release 2022a)
September 2022	Online only	Revised for Version 4.6 (Release 2022b)
March 2023	Online only	Revised for Version 4.7 (Release 2023a)

Contents

1 Functions

2 Properties

Functions

addAnalogInputChannel

(Not recommended) Add analog input channel

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

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.

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.

Version History

Introduced in R2010b

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see "Transition Your Code from Session to DataAcquisition Interface".

See Also

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

Topics
daq.Session Properties
daq.Channel Properties

addAnalogOutputChannel

(Not recommended) Add analog output channel to session

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

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.

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.

Version History

Introduced in R2010b

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

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

Topics
daq.Session Properties
daq.Channel Properties

addAudioInputChannel

(Not recommended) Add audio input channel to session

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See "Compatibility Considerations".

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.

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.

Version History

Introduced in R2014a

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

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

Topics
daq.Session Properties
daq.Channel Properties

addAudioOutputChannel

(Not recommended) Add audio output channel to session

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

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.

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.

Version History

Introduced in R2014a

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

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

Topics
daq.Session Properties
daq.Channel Properties

addClockConnection

(Not recommended) Add clock connection

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

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.

Version History

Introduced in R2012a

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

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

Topics
daq.Session Properties

addCounterInputChannel

(Not recommended) Add counter input channel

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See "Compatibility Considerations".

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 daq.Channel Properties.

`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.

Version History

Introduced in R2011a

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see "Transition Your Code from Session to DataAcquisition Interface".

See Also

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

Topics
daq.Session Properties
daq.Channel Properties

addCounterOutputChannel

(Not recommended) Add counter output channel

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

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.

`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.

Version History
Introduced in R2011a

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

Functions
`addCounterInputChannel` | `startForeground` | `startBackground` | `removeChannel`

Topics
daq.Session Properties
daq.Channel Properties

addDigitalChannel

(Not recommended) Add digital channel

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

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.

Version History

Introduced in R2012b

R2020a: `session` object interface is not recommended

Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see "Transition Your Code from Session to DataAcquisition Interface".

See Also

Functions

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

Topics

daq.Session Properties

addFunctionGeneratorChannel

(Not recommended) Add function generator channel

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

Syntax

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

Description

`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.

Version History

Introduced in R2014b

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see "Transition Your Code from Session to DataAcquisition Interface".

See Also

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

Topics
daq.Session Properties

addlistener

Package: daq

(Not recommended) Create event listener

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See "Compatibility Considerations".

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.

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.

Version History

Introduced in R2010b

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

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

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

Topics
daq.Session Properties

addTriggerConnection

(Not recommended) Add trigger connection

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

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 — Trigger destination terminal

char

This property is read-only.

Device and terminal to which you connect a trigger destination.

Example

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

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

Source — Trigger source terminal

char

This property is read-only.

Device and terminal to which you added a trigger source.

Example

Create an external clock connection and view the connection properties.

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

TriggerCondition — Condition that must be satisfied before trigger executes

'RisingEdge' (default) | 'FallingEdge'

Specify the signal condition that executes the trigger, which synchronizes operations on devices in a session. Set the trigger condition to `RisingEdge` or `FallingEdge`.

For more information, see “Synchronization”.

Example

Create a session and add channels and a trigger to the session. Change the trigger condition to `FallingEdge`.

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

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 `(Details)` to see the trigger 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
```

`TriggerType` — Type of trigger executed
`'digital'` (default)

This property is read-only.

Type of trigger that the source device executes to synchronize operations in the session. Currently all trigger types are `digital`.

`Type` — Operation of trigger executed
`'StartTrigger'` (default)

This property is read-only.

Operation of the trigger that the source device executes to synchronize operations in the session. Currently the only value is `'StartTrigger'`.

`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.

Version History

Introduced in R2012a

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

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

Topics
daq.Session Properties

daq.createSession

(Not recommended) Create data acquisition session for specific vendor hardware

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

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'`
- `'digilent'`
- `'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 properties described in daq.Session Properties.

Version History

Introduced in R2010b

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see "Transition Your Code from Session to DataAcquisition Interface".

See Also

Functions
`daq.getDevices` | `daq.getVendors` | `addAnalogInputChannel` | `addAnalogOutputChannel` | `addAudioInputChannel` | `addAudioOutputChannel` | `addDigitalChannel` | `addCounterInputChannel` | `addCounterOutputChannel` | `addTriggerConnection` | `addClockConnection`

Topics
daq.Session Properties
daq.Channel Properties

daq.getDevices

(Not recommended) 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-acquistion-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

Examine Device Properties

View information about devices by examining device object properties.

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 the NI 9201, with the device id `cDAQ1Mod1` at 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 object.

```
  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.

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 the NI 9205 at index `1`.

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

```
ans =

    'cDAQ1/PFI0'
    'cDAQ1/PFI1'
    'cDAQ1/20MHzTimebase'
    'cDAQ1/80MHzTimebase'
    'cDAQ1/ChangeDetectionEvent'
    'cDAQ1/AnalogComparisonEvent'
    'cDAQ1/100kHzTimebase'
    'cDAQ1/SyncPulse0'
    'cDAQ1/SyncPulse1'
         ⋮
```

Output Arguments

device — Device list
array of `DeviceInfo` objects

Device list, returned as an array of `DeviceInfo` objects.

Version History

Introduced in R2010b

R2020a: `daq.getDevices` is not recommended
Not recommended starting in R2020a

Use of this function not recommended. Use `daqlist` instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

Functions
`daq.getVendors` | `daq.createSession`

daq.getVendors

(Not recommended) 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”.

Version History

Introduced in R2010b

R2020a: `daq.getVendors` is not recommended
Not recommended starting in R2020a

Use of this function not recommended. Use `daqvendorlist` instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

Functions
`daq.createSession` | `daq.getDevices`

inputSingleScan

(Not recommended) Acquire single scan from all input channels

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

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`.

Version History

Introduced in R2010b

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

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

Topics
daq.Session Properties

outputSingleScan

(Not recommended) Generate single scan on all output channels

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

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.

Version History

Introduced in R2010b

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

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

Topics
daq.Session Properties

prepare

(Not recommended) Prepare session for operation

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

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.

Version History

Introduced in R2010b

R2020a: `session` object interface is not recommended

Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

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

Topics
daq.Session Properties

queueOutputData

(Not recommended) Queue data to be output

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See "Compatibility Considerations".

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.

Version History

Introduced in R2010b

R2020a: `session` object interface is not recommended

Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

Functions

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

Topics

daq.Session Properties

release

(Not recommended) Release session hardware resources

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

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.

Version History

Introduced in R2010b

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

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

Topics
daq.Session Properties

removeChannel

(Not recommended) Remove channel from session object

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

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.

Version History

Introduced in R2010b

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

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

Topics
daq.Session Properties

removeConnection

(Not recommended) Remove clock or trigger connection

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

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.

Version History

Introduced in R2012a

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

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

Topics
daq.Session Properties

resetCounters

(Not recommended) Reset counter channel to initial count

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

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.

Version History

Introduced in R2011a

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

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

Topics
daq.Session Properties

startBackground

(Not recommended) Start background operations

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

Syntax

```
startBackground(s);
```

Description

`startBackground(s);` starts the operation of the session object, `s`, without blocking the 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”.

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.

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.

Version History

Introduced in R2010b

R2020a: `session` object interface is not recommended

Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

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

Topics
daq.Session Properties

startForeground

(Not recommended) Start foreground operations

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

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`.

Version History

Introduced in R2010b

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

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

Topics
daq.Session Properties

stop

(Not recommended) Stop background operation

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See "Compatibility Considerations".

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.

Version History

Introduced in R2010b

R2020a: `session` object interface is not recommended

Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

Functions

`startBackground` | `startForeground` | `wait`

Topics

daq.Session Properties

wait

(Not recommended) Block MATLAB until background operation completes

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

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.

Version History

Introduced in R2010b

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

Functions
`startBackground` | `stop`

Topics
daq.Session Properties

DataAvailable

(Not recommended) Notify when acquired data is available to process

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

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.

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.

Version History

Introduced in R2010b

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

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

Properties
IsNotifyWhenDataAvailableExceedsAuto | NotifyWhenDataAvailableExceeds

Topics
daq.Session Properties

DataRequired Event

(Not recommended) Notify when additional data is required for output on continuous generation

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

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.

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.

Version History

Introduced in R2010b

R2020a: `session` object interface is not recommended

Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

Functions

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

Properties

IsContinuous | IsNotifyWhenScansQueuedBelowAuto | NotifyWhenScansQueuedBelow

Topics
daq.Session Properties

ErrorOccurred Event

(Not recommended) Notify when device-related errors occur

Note This `session` object function is not recommended. Use `DataAcquisition` object functions instead. See “Compatibility Considerations”.

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.

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 `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.

Version History

Introduced in R2010b

R2020a: `session` object interface is not recommended
Not recommended starting in R2020a

Use of this function with a `session` object is not recommended. To access a data acquisition device, use a `DataAcquisition` object with its functions and properties instead.

For more information about using the recommended functionality, see “Transition Your Code from Session to DataAcquisition Interface”.

See Also

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

Classes
`MException`

Topics
daq.Session Properties

Properties

daq.Session Properties

Session object properties

Description

Alphabetical listing of session object properties. Sessions for some vendors might not have all properties.

Properties

Session Properties

AutoSyncDSA — Automatically Synchronize DSA devices

false (default) | true

Automatically Synchronize DSA devices, specified as a logical true or false. Use this property to enable or disable automatic synchronization between DSA (PXI or PCI) devices in the same DataAcquisition. By default automatic synchronization capability is disabled.

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);
```

Example: true

Data Types: logical

BitsPerSample — Sampling resolution

numeric

This property is read-only.

Sampling resolution indicating 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 the 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
```

`Channels` — Device channels
array of channel objects

This property is read-only.

Device channels, returned as an array of channel objects.

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

`Connections` — Device connections
array of clock and trigger connection objects

This property is read-only.

Device clock and trigger connections, returned as an array of objects.

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

Tip You cannot directly add or remove connections using the `Connections` property. Use `addTriggerConnection` and `addClockConnection` to add connections. Use `removeConnection` to remove connections.

Example: 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
```

`DurationInSeconds` — Specify duration of acquisition

1 (default) | numeric

Duration of an acquisition, specified in seconds. 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.

When the session contains analog, digital, or audio output channels, `DurationInSeconds` is a read-only property whose value is determined by

```
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.

Example: Create a session object, add an analog input channel, and change the session 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     ai   cDAQ1Mod1 ai0       Voltage (Diff)  -10 to +10 Volts
```

Data Types: `double`

`ExternalTriggerTimeout` — Specify maximum wait time for external trigger
numeric

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`.

Example: 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;
```

Example: 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;
```

Data Types: `double`

`IsContinuous` — Specify operation to continue until manually stopped

`false` (default) | `true`

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.

- Set `IsContinuous` to `false` to make the session operation stop automatically. This property is set to `false` by default.
- Set `IsContinuous` to `true` to make the session operation run until you execute `stop`.

Example: 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
```

Data Types: `logical`

`IsDone` — Indicate if session operation is complete

`true` | `false`

This property is read-only.

Indication that the session operation is complete, returned as `true` or `false`.

- Value is logical `1` (`true`) when the session operation is complete.
- Value is logical `0` (`false`) while the session operation is not 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.

Example: 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
```

Data Types: `logical`

`IsLogging` — Indicate if hardware is acquiring or generating data

`true` | `false`

This property is read-only.

Indication if the hardware is actively acquiring or generating data, returned as `true` or `false`.

- Value is logical `1` (`true`) if the device is acquiring or generating data.
- Value is logical `0` (`false`) if the device is not 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.

Example: Check device logging.

Create and start a session.

```
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
```

Data Types: `logical`

`IsNotifyWhenDataAvailableExceedsAuto` — Control if `NotifyWhenDataAvailableExceeds` is set automatically

`true` (default) | `false`

Indication 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.

- When the value is `true` (default), then the NotifyWhenDataAvailableExceeds property is set automatically.
- When the value is `false`, you 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 `false`.

```
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.
```

Data Types: `logical`

`IsNotifyWhenScansQueuedBelowAuto` — Control if `NotifyWhenScansQueuedBelow` is set automatically

`true` (default) | `false`

Indication if the NotifyWhenScansQueuedBelow property is set automatically, or you have set a specific value.

- When the value is `true`, then NotifyWhenScansQueuedBelow is set automatically.
- When the value is `false`, you set NotifyWhenScansQueuedBelow 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 `false`.

```
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.
```

Data Types: `logical`

`IsRunning` — Indicate if session operation is in progress
`true` | `false`

This property is read-only.

Session running indication, returned as `true` or `false`.

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.

Example: 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 the session is in progress.

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

```
ans =

     0
```

Data Types: `logical`

`IsWaitingForExternalTrigger` — Indicates if synchronization is waiting for an external trigger

false (default) | true

This property is read-only.

Indication 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`.

Example: `true`

Data Types: `logical`

`NotifyWhenDataAvailableExceeds` — Specify number of acquired scans to fire `DataAvailable` event

numeric

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

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`.

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.

Example: 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 a 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 `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
```

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`NotifyWhenScansQueuedBelow` — Specify number of queued scans to fire `DataRequired` event
numeric

When generating 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.

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);
```

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`NumberOfScans` — Number of scans for operation when starting
numeric

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 a time length for the acquisition, use DurationInSeconds.
- To control the length of an output operation, use `queueOutputData`.

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
```

Rate — Data scan rate
numeric

Data scan rate, specified as a numeric value of samples per second. You can set the rate to any positive nonzero scalar value supported by the hardware in its current configuration. 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.

Example: Set the 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 scans per second.

```
s.Rate = 10000
```

```
s =

Data acquisition session using National Instruments hardware:
   Will run for 1 second (10000 scans) at 10000 scans/second.
   Operation starts immediately.
      Number of channels: 1
      index Type  Device   Channel  MeasurementType      Range        Name
      ----- ---- --------- ------- ----------------- ---------------- ----
      1     ai   cDAQ1Mod1 ai1       Voltage (Diff)  -10 to +10 Volts
```

Data Types: `double`

`RateLimit` — Lower and upper scan rate limits
array of doubles

This property is read-only.

Lower and upper scan rate limits, returned as a 1-by-2 vector of doubles indicating minimum and maximum allowed scan rates in samples per second. The scan rate limits depend on the hardware and its configurations. In devices that multiplex channels to a converter, the rate limit is impacted by the number of channels you use.

`RateLimit` changes dynamically as the session configuration changes.

Example: Find the Session Rate Limits

Create a session and add an analog input channel.

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

Examine the session rate limits.

```
format("longG")
s.RateLimit

ans =
    0.1        250000
```

Data Types: `double`

`ScansAcquired` — Number of data scans acquired during operation

numeric

This property is read-only.

Number of scans acquired after you start the operation calling `startBackground`. This value is reset each time you call `startBackground`.

Example: Display Number of Acquired Scans

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 scans the session had acquired before and after running.

```
s.ScansAcquired

ans =
    0

startForeground(s);
s.ScansAcquired

ans =
    1000
```

`ScansOutputByHardware` — Number of scans generated as device output

numeric

This property is read-only.

Number of scans output by the hardware after you start the operation using `startBackground`. The value depends on information from the hardware.

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
```

`ScansQueued` — Number of scans prepared for device output
numeric

This property is read-only.

Number of scans queued to the device output channels. Add scans to the queued with `queueOutputData`. The `ScansQueued` property value decreases when the hardware reports that it has successfully output data.

Example: Monitor Scans Queue

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, then examine the `ScansQueued` property to see the number of scans queued.

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

```
ans =
    1000
```

`StandardSampleRates` — Standard available scan rates
double array

This property is read-only.

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

View the standard sample rates for DirectSound audio devices:

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

```
ans =

        8000
        8192
       11025
       16000
       22050
       32000
       44100
       47250
```

```
      48000
      50000
      88200
      96000
     176400
     192000
     352800
```

Data Types: `double`

`TriggersPerRun` — Number of digital triggers per session acquisition run

numeric

Number of digital triggers per session acquisition run, returned as a double. This is the number of times the specified trigger executes for one acquisition or generation session.

Example: 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 the session `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
```

Data Types: `double`

`TriggersRemaining` — Number of digital triggers remaining in acquisition

numeric

This property is read-only.

The number of triggers remaining for this acquisition or generation session, returned as a double. This value depends on the number of triggers set using TriggersPerRun.

Example: Display Number of Triggers Remaining in Operation

Create a session and add channels and a 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 the session `TriggersRemaining` property.

```
s.TriggersRemaining
```

```
ans =

     1
```

Data Types: `double`

`UserData` — Custom data
any data

Custom data, specified as any MATLAB data type and format.

Example: Create a session and define its `UserData` property as a struct with custom 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];
```

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `string` | `struct` | `table` | `cell` | `function_handle` | `categorical` | `datetime` | `duration` | `calendarDuration` | `fi`

`UseStandardSampleRates` — Configure session to use standard sample rates
`true` (default) | `false`

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 Rate to the desired value.

Example: Configure a nonstandard sample 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
```

Data Types: `logical`

`Vendor` — Data acquisition hardware vendor information
vendor object

This property is read-only.

Data acquisition hardware vendor information associated with the session, returned as a vendor object with the following properties:

```
ID
FullName
AdaptorVersion
DriverVersion
IsOperational
```

This object is the same as the corresponding vendor object returned by the `daq.getVendors` function.

Example: Create a session and get information its vendor.

```
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
```

Version History
Introduced in R2010b

See Also

Topics
daq.Channel Properties

daq.Channel Properties

Channel object properties

Description

Alphabetical listing of channel object properties. The type of channel determines which of these properties a particular instance has.

Properties

Channel Properties

ActiveEdge — Rising or falling edges of EdgeCount signals

'Rising' (default) | Falling

Rising or falling edges of the EdgeCount signal, specified as 'Rising' or 'Falling'.

Example: Create an EdgeCount counter channel, and set its ActiveEdge.

```
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 ActiveEdge 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'
```

Data Types: char | string

ActivePulse — Active pulse level for measurement of PulseWidth counter channel

'High' (default) | 'Low'

Pulse width measurement active level, specified as 'High' or 'Low'.

Example: Set active pulse level.

Create a session object, add a counter input channel, with the `'PulseWidth'` `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: 'ctr1'
          Device: [1x1 daq.ni.DeviceInfo]
 MeasurementType: 'PulseWidth'
```

Change the `ActivePulse` 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'
```

Data Types: `char` | `string`

ADCTimingMode — Specify channel timing mode

char

Timing mode of all channels in the device. You can set the `ADCTimingMode` to:

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

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

Example: Set channel timing mode

Create a session and add an analog input channel.

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

```
ch =

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';
```

Data Types: char | string

BridgeMode — Specify analog input device bridge mode
char

Specify the bridge mode, which represents the active gauge of the analog input channel. The default value 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.

Example

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
```

Data Types: char | string

CountDirection — Specify direction of counter channel
'Increment' (default) | 'Decrement'

Direction of the channel counter, specified as `'Increment'` (default) to count up, or `'Decrement'`, to count down.

Example

Create a session object, add a counter input channel, and change its `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'
```

Data Types: `char` | `string`

`Coupling` — Input coupling mode
`'DC'` | `'AC'`

Coupling mode used for the analog input signal connection, specified as `'DC'` or `'AC'`. 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.

Example

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'
```

Data Types: `char` | `string`

`Device` — Channel device information
`DeviceInifo` object

This property is read-only.

Device information for the channel, returned as a `DeviceInfo` object.

Example

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

`Direction` — Specify digital channel direction

`'Unknown'` (default) | `'Input'` | `'Output'`

When you specify a digital channel `MeasurementType` as `Bidirectional`, you can use the channel to input and output data. By default the channel `Direction` is set to `'Unknown'`. Other possible values are `'Input'` and `'Output'`.

Example: Change Bidirectional Channel Direction

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 channel direction to `'Input'`. Note the impact on the display of the channel description, `Direction`, and `MeasurementType`.

```
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)'
```

Data Types: `char` | `string`

`DutyCycle` — Duty cycle of output channel

numeric

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, or duty cycle 0.5.

You can change the duty cycle of counter output channels while the session is running.

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

Example

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`.

```
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'
```

EncoderType — Encoding type of counter channel
`'X1'` | `'X2'` | `'X4'` | `'TwoPulse'`

Specify the encoding type of the counter input `'Position'` channel. Supported encoder types include:

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

Example

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 channel 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
```

Data Types: `char` | `string`

`EnhancedAliasRejectionEnable` — Specify enhanced alias rejection mode
`false` (default) | `true`

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.

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

Example: 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 = true
```

```
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
```

Data Types: `logical`

`ExcitationCurrent` — Current of external source of excitation

numeric

The current in amperes to excite an IEPE accelerometer, IEPE microphone, generic IEPE sensor, or RTD.

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

Example: 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 4 milliamps.

```
ch.ExcitationCurrent = .004
```

```
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: ''
```

Data Types: `double`

`ExcitationSource` — External source of excitation
`'Unknown'` (default) | `'Internal'` | `'External'` | `'None'`

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 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: ''
```

Data Types: `char` | `string`

`ExcitationVoltage` — Voltage of excitation source
numeric

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 is the lowest available value in the range.

Data Types: `double`

`Frequency` — Frequency of generated output
numeric

On counter output channels, use the `Frequency` property to set the pulse repetition rate. You can change the channel frequency while the session is running when using counter output channels.

On function generation channels use the `Frequency` property to specify the waveform frequency. You can set each channel frequency individually.

The frequency value must fall within the specified FrequencyLimit values.

Example: Set the Frequency of a Counter Output 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`.

```
ch.Frequency = 200

ch =

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'
```

Example: 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'
```

Data Types: `double`

`FrequencyLimit` — Limit of rate of operation based on hardware configuration

double array

This property is read-only.

The minimum and maximum rates that the function generation channel supports. `FrequencyLimit` changes dynamically as the channel configuration changes.

Example

View the frequency limits of a waveform function generation channel.

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

```
ans =

[0.0 25000000.0]
```

Data Types: `double`

`Gain` — Waveform output gain

numeric

When using waveform function channels to generate standard waveforms, `Gain` sets the peak amplitude. For arbitrary waveforms, `Gain` prepresents the value by which the data is multiplied to generate the output waveform.

The waveform gain can range from –5 to 5. Be sure that for any instantaneous output point `Gain x Voltage + Offset` falls within the valid output voltage range of the device.

Example

Change the gain of a waveform function generation channel for an amplitude of 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'
```

Data Types: `double`

`ID` — Channel identifier

char | string

This property is read-only.

Identifier of the channel, returned as a character array. The ID specifies a particular channel in the device subsystem. Use the channel ID when you add the channel to a session object.

Example

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'
```

Data Types: `char` | `string`

`IdleState` — Default state of counter output channel

`'Low'` (default) | `'High'`

Default state of the counter output channel with a `'PulseGeneration'` measurement type, when the counter is not active, specified as `'Low'` or `'High'`.

Example

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

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

Change the `IdleState` property to `'High'`.

```
ch.IdleState = 'High'

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'
```

Data Types: `char` | `string`

`InitialCount` — Specify initial count point
0 (default) | numeric

Point from which the device starts the counter, specified as a numeric value.

Example

Create a session object and add a counter input channel.

```
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 the `InitialCount` value 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'
```

`InitialDelay` — Time delay until output channel generates pulses
numeric

Delay before counter output channel generates pulses, specified in seconds.

Example

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

Create a session and add a counter output 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'
```

`MaxSoundPressureLevel` — Sound pressure level for microphone channels

numeric

Specify the maximum sound pressure of the microphone channel in decibels. The maximum sound pressure level is based on the sensitivity and the voltage range of your device. When you set 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 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 `Sensitivity` to `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 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
char

This property is read-only.

Specified measurement type for the channel. Some channels support many measurement types:

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)

Example

Create a session object, add a counter input channel with an `'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: 'PFI8'
            Name: empty
              ID: 'ctr0'
          Device: [1x1 daq.ni.DeviceInfo]
 MeasurementType: 'EdgeCount'
```

Data Types: `char`

Name — Descriptive name for channel
char | string

Descriptive name for the channel, specified as a character vector or string. By default there is no name assigned to a channel. You can change the value of `Name` at any time.

Example

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 the channel `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'
```

Data Types: `char` | `string`

`NominalBridgeResistance` — Resistance of sensor
numeric

Resistance of a bridge based sensor, specified 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.

`Offset` — DC offset of waveform
numeric

When using waveform function generation channels, `Offset` specifies the DC offset voltage of a signal from zero, or the mean value of the waveform.

The waveform offset can be from –5 to 5 volts. Be sure that `Gain x Voltage + Offset` falls within the valid range 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'
```

`Phase` — Waveform phase shift
numeric

In a function generation channel, the `Phase` property specifies the waveform phase shift in degrees.

Example

Set the phase shift of a waveform function generation channel to 90°.

```
s = daq.createSession('digilent');
fgenCh = addFunctionGeneratorChannel(s, 'AD1', 1,'Sine');
fgenCh.Phase = 90
```

```
fgenCh =

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

              Phase: 90
              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'
```

R0 — RTD channel resistance value at 0°C

numeric

Specify the 0° resistance of the device in ohms. When you add an RTD channel, the resistance is initially unknown and the `R0` property displays `Unknown`. Before any measurement, you must change this value to correspond to the resistance of the device. For more information, see https://en.wikipedia.org/wiki/Resistance_thermometer.

Example

Create a session and add an RTD channel.

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

For a standard 100Ω probe, change the channel 0° resistance to 100Ω.

```
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
```

Range — Channel measurement range

numeric vector

Specify the measurement range of a channel, as a vector of numeric values.

For analog channels, the 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. `Range` is not applicable for counter channels.

Example: 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];
```

Example: Examine supported ranges

Create a session and add an analog input channel.

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

Display the 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
```

View the device subsystems.

```
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
```

RTDConfiguration — Specify wiring configuration of RTD device

'TwoWire' | 'ThreeWire' | 'FourWire'

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 an RTD channel wiring configuration

Create a session and add an RTD channel.

```
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
```

RTDType — Specify sensor sensitivity
char | string

Specify the sensitivity of a standard RTD sensor. 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 supported values:

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

Example: Set an RTD sensor sensitivity type

Create a session and add an RTD channel.

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

Set the `RTDType` property 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
```

Data Types: `char` | `string`

`Sensitivity` — Sensitivity of an analog channel
numeric

Specify the accelerometer or microphone sensor channel sensitivity.

Sensitivity in an accelerometer channel is expressed as volts per g-force, V/*g*.

Sensitivity in a microphone channel is expressed as volts per pascal, V/Pa.

Example

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

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

```
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) -5.0 to +5.0 Volts
```

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

```
ch.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
```

`ShuntLocation` — Specify location of channel shunt resistor
`'Internal'` | `'External'`

On an analog input current channel, specify if the shunt resistor is located internally on the device or externally with a property value of `'Internal'` or `'External'`.

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

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
```

Data Types: `char` | `string`

`ShuntResistance` — Resistance of channel shunt resistor
numeric

Analog input current channel resistance in ohms. This value is automatically set if the shunt resistor is located internally on the device and is read-only.

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

Example

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` ohms.

```
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
```

`Terminal` — PFI terminal of counter subsystem

char

This property is read-only.

The `Terminal` property indicates the counter subsystem’s corresponding PFI terminal.

Example: Determine Counter Input Channel Terminal

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
```

`TerminalConfig` — Specify terminal configuration

char | string

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

- `SingleEnded`
- `SingleEndedNonReferenced`
- `Differential`
- `PseudoDifferential`

Example: 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 channel `TerminalConfig` property 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'
```

Data Types: `char` | `string`

`ThermocoupleType` — Specify thermocouple type

char | string

Specify the type of thermocouple you used in making your measurements. Select the type based on the temperature range and sensitivity you need, according to the NIST Thermocouple Types Definitions.

Supported `ThermocoupleType` values are:

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

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

Example

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
```

Data Types: `char` | `string`

Units — Specify unit of RTD measurement
`'Celsius'` (default) | `'Fahrenheit'` | `'Kelvin'` | `'Rankine'`

Specify the temperature unit of the analog input channel with RTD measurement type. Supported temperature units are:

- `Celsius` (Default)
- `Fahrenheit`
- `Kelvin`
- `Rankine`

Example

Set 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` Notice the impact on the `Range` property value.

```
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
```

Data Types: `char` | `string`

`WaveformType` — Function generator channel waveform type
char

This property is read-only.

Indicate the channel waveform type that was specified when the function generator channel was created. Supported waveform types are:

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

Display the channel waveform type.

```
fgenCh.WaveformType
```

```
ans =

    Sine
```

Data Types: `char`

`ZResetCondition` — Reset condition for Z-indexing
char | string

Specify reset conditions for Z-indexing of counter Input `'Position'` channels. Supported values are:

- `'BothHigh'`
- `'BothLow'`

- `'AHigh'`
- `'BHigh'`

Example: Specify 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 channel `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'
```

Data Types: `char` | `string`

`ZResetEnable` — Enable reset for Z-indexing
`false` (default) | `true`

Allow the Z-indexing to be reset on a counter input `'Position'` channel, specified as `false` or `true`.

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 `true`.

```
ch.ZResetEnable = true
```

```
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'
```

Data Types: `logical`

ZResetValue — Reset value for Z-indexing
numeric

Specify the reset value for Z-indexing on a counter input `'Position'` channel.

Example

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`. Notice also the change in `ZResetEnable`.

```
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'
```

Version History

Introduced in R2010b

See Also

Topics

daq.Session Properties