

# NATIONAL INSTRUMENTS™ Data Acquisition

## NI-DAQ™mx Base 3.x C Function Reference Help

June 2008 Edition, Part Number 371164F-01

Thank you for using NI-DAQmx Base 3.x. The NI-DAQmx Base 3.x software contains a C Application Programming Interface (API), which allows you to create applications for your device.

For more information about this help file, refer to the following topics:

[Conventions](#)—formatting and typographical conventions in this help file

[Related Documentation](#)

[Glossary](#)

[Important Information](#)






[Technical Support and Professional Services](#)

To comment on National Instruments documentation, refer to the [National Instruments Web site](#).

©2004–2008 National Instruments Corporation. All rights reserved.

### Conventions

This help file uses the following formatting and typographical conventions: edit this list according to the specific conventions used in your help file.

- < >    Angle brackets that contain numbers separated by an ellipsis represent a range of values associated with a bit or signal name—for example, AO <0..3>.
- [ ]    Square brackets enclose optional items—for example, [response].
- >    The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.
- ◆    The ◆ symbol indicates that the following text applies only to a specific product, a specific operating system, or a specific software version.
-     This icon denotes a tip, which alerts you to advisory information.
-     This icon denotes a note, which alerts you to important information.
-     This icon denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash.
-     When symbol is marked on a product, it denotes a warning advising you to take precautions to avoid electrical shock. Only use in conventions, *not* in text.
-     When symbol is marked on a product, it denotes a component that may be hot. Touching this component may result in bodily injury. Only use in conventions, *not* in text.
- bold**    Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
- green    Underlined text in this color denotes a link to a help topic, help file, or Web address.
- italic*    Italic text denotes variables, emphasis, cross-references, or an introduction to a key concept. Italic text also denotes text that is a placeholder for a word or value that you must supply.

### Related Documentation

The following documents contain information that you might find helpful as you use this help file:

- *NI-DAQmx Base Readme*
- *NI-DAQmx Base 3.x Getting Started Guide*
- *NI-DAQmx Base VI Reference Help*
- *E Series User Manual*
- *M Series User Manual*

- *PCI-DIO-96/PXI-6508/PCI-6503 User Manual*
- *NI CF-6004 User Guide and Specifications*
- *NI USB-6008/6009 User Guide and Specifications*
- *NI USB-6501 User Guide and Specifications*
- *NI USB-621x User Manual*
- *NI USB-621x Specifications*
- *NI USB-9211/9211A User Guide and Specifications*
- *NI USB-9215 Series User Guide and Specifications*
- *NI USB-9233 Series User Guide and Specifications*
- *NI USB-9234 Series User Guide and Specifications*
- Documentation for your Windows Mobile device
- *Getting Started with the LabVIEW Windows Mobile Module*
- *LabVIEW Help*
- *LabVIEW Fundamentals*
- *NI-DAQmx Help*

## Clocks

Periodic digital edges measure time and are called *clocks*. Clocks such as a sample timebase clock and the 20 MHz timebase clock mark the passing of time or are used to align other signals in time. Clocks usually do not cause actions in the sense that triggers do. The names of clocks usually do not refer to actions. The sample clock is a notable exception.

### Clocks in NI-DAQmx

- **AI Convert Clock**—The E Series clock that directly causes ADC conversions.
- **AI Convert Clock Timebase**—The clock that is divided down to produce the AI convert clock.
- **AI Sample Clock**—The clock that controls the time interval between samples. Each time the sample clock ticks (produces a pulse) one sample per channel is acquired.
- **AI Sample Clock Timebase**—The clock used as the onboard clock source of the sample clock. When the source of the sample clock is set to the onboard clock, the Sample Clock Timebase is divided down to produce the sample clock. When the source of the Sample Clock Timebase is also the onboard clock, the master timebase is divided down to produce the Sample Clock Timebase.
- **AO Sample Clock**—The clock that controls the time interval between samples. Each time the sample clock ticks (and produces a pulse), one sample per channel is generated.
- **AO Sample Clock Timebase**—The onboard clock used as the source of the AO sample clock. The AO Sample Clock Timebase is divided down to produce the AO sample clock.
- **Counter Timebase**—The clock connected to the Source terminal of a counter (Ctr0Source, for example).
- **Master Timebase**—An onboard clock used by other counters on the device. The master timebase is divided down to produce a slower clock or to measure elapsed time. This timebase is the onboard clock used as the source of the AI Sample Clock timebase, the AO Sample Clock timebase, and the counter timebases, for example.
- **20 MHz Timebase**—The onboard clock source for the master timebase from which other timebases are derived.
- **100 kHz Timebase**—The clock produced by dividing the 20 MHz Timebase by 200.

### Trigger and Clock Confusion

The distinction between triggers and clocks is blurred when the digital edges used as a trigger are periodic. In such a case, a clock causes the device to perform an action. The sample clock is the primary example. The stimulus for the action of producing a sample is so often a clock that NI-DAQmx Base configures the sample clock instead of the sample trigger. The distinction is made clear when you consider the sample clock is in fact just one way of providing the source of a sample trigger.

## Terminal Names

|                |  |
|----------------|--|
| Onboard Clock  | An alias for the terminal within a device where the default source for a clock can be found. If your application does not set the source of a clock (or uses an empty string as the source), the clock's particular onboard clock is used. For example, the onboard clock for the ai sample clock is the ai Sample Clock Timebase. |
| PFI $n$        | Programmable Function Interface—general-purpose input terminals, fixed-purpose output terminals. The name of the fixed output signal is often placed on the I/O connector next to the terminal as a hint.  |
| PXI $Trig$ $n$ | PXI Trigger bus—general-purpose input/output lines.  |

|  |  |
|--|--|
| RTSI <i>n</i>                          | Real Time System Integration bus—general-purpose input/output lines. RTSI7 is the exception. It is the only line to use for the 20 MHz Timebase signal.  |
| ai/SampleClock                         | A terminal within a device where the analog input sample clock can be found.   |
| ai/StartTrigger                        | A terminal within a device where the analog input Start Trigger can be found.  |
| ai/ReferenceTrigger                    | A terminal within a device where the analog input Reference Trigger can be found.  |
| ao/SampleClock                         | A terminal within a device where the analog output sample clock can be found.  |
| ao/StartTrigger                        | A terminal within a device where the analog output Start Trigger can be found.   |
| 20MHzTimebase                          | A terminal within a device where the onboard clock source for the master timebase can be found.  |
| MasterTimebase                         | A terminal within a device where the master timebase signal can be found. This signal originates either from the 20MHzTimebase terminal or the RTSI7 terminal. This signal is the onboard source for the Sample Clock Timebases and is one of the possible sources for the AI convert clock timebase.  |
| 100kHzTimebase                         | A terminal within a device where the 100 kHz Timebase signal can be found. This signal is created by dividing the signal at the 20MHzTimebase terminal by 200 and is one of the possible sources for the Sample Clock Timebases.   |
| ai/ConvertClock                        | A terminal within a device where the AI Convert Clock can be found.  |
| ai/ConvertClockTimebase                | A terminal within a device where the AI Convert Clock Timebase can be found. This is the onboard clock source for the AI convert clock.  |
| ai/HoldCompleteEvent                   | A terminal within a device where the AI Hold Complete Event signal can be found.   |
| AI Hold Complete                       | The terminal at the I/O connector (external to the device) where the AI Hold Complete Event signal can be emitted.   |
| ai/PauseTrigger                        | A terminal within a device where the analog input pause trigger can be found.  |
| ai/SampleClockTimebase                 | A terminal within a device where the AI Sample Clock Timebase can be found. This is the onboard clock source for the AI sample clock.  |
| AnalogComparisonEvent                  | A terminal within a device where the output of the analog comparison circuit, the Analog Comparison Event signal, can be found. This circuit is active whenever an analog edge or window trigger is configured.  |
| ao/PauseTrigger                        | A terminal within a device where the analog output pause trigger can be found.   |
| ao/SampleClockTimebase                 | A terminal within a device where the AO Sample Clock Timebase can be found. This is the onboard clock source for the AO sample clock.  |
| Ctr0Out, Ctr1Out                       | Terminals at the I/O connector where the output of counter 0 or counter 1 can be emitted. You also can use Ctr0Out also as an input terminal for driving an external signal onto the RTSI bus.   |
| Ctr0Gate, Ctr1Gate                     | Terminals within a device whose purpose depends on the application.  |
| Ctr0Source, Ctr1Source                 | Terminals within a device whose purpose depends on the application.  |
| Ctr0InternalOutput, Ctr1InternalOutput | Terminals within a device where you can choose the pulsed or toggled output of the counters.   |
| PairedCtrInternalOutput                | An alias for one of the counter internal output terminals. For example, if you use counter 1, PairedCtrInternalOutput refers to Ctr0InternalOutput. If you use counter 0, PairedCtrInternalOutput refers to Ctr1InternalOutput.  |
| PairedCtrOutputPulse                   | A terminal within a device that chains counters together without using any external connections. If you configure counter 0, PairedCtrOutputPulse refers to the pulsed output of counter 1. If you configure counter 1, PairedCtrOutputPulse refers to the pulsed output of counter 0. When the counter reaches terminal count (zero when counting down, its maximum count when counting up), the output of the PairedCtrOutputPulse pulses. By using this terminal, you can chain counters together to create a wider counter, perform buffered event counting using the other counter as your clock source, perform finite pulse-train generation, and create other custom applications. |

## DAQmxBaseClearTask

```
int32 DAQmxBaseClearTask (TaskHandle taskHandle);
```

### Purpose

Clears the task. Make sure the task has been stopped by calling `DAQmxBaseStop Task` before calling this VI. Before clearing, this VI releases any resources the task reserved. You cannot use a task after you clear it unless you recreate the task.

If you use the `DAQmxBaseCreateTask` function or any of the NI-DAQmxBase Create Channel functions within a loop, use this function within the loop after you finish with the task to avoid allocating unnecessary memory.

## Parameters

*Input*

| Name                    | Type       | Description        |
|-------------------------|------------|--------------------|
| <code>taskHandle</code> | TaskHandle | The task to clear. |

## Return Value

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

|                     |       |   |
|---------------------|-------|---|
| <code>status</code> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |
|---------------------|-------|---|

## DAQmxBaseCreateTask

```
int32 DAQmxBaseCreateTask (const char taskName[ ], TaskHandle *taskHandle);
```

## Purpose

Creates a task. If you use this function to create a task, you must use `DAQmxBaseClearTask` to destroy it.

If you use this function within a loop, NI-DAQmxBase creates a new task in each iteration of the loop. Use the `DAQmxBaseClearTask` function within the loop after you finish with the task to avoid allocating unnecessary memory.

## Parameters

*Input*

| Name                  | Type           | Description                |
|-----------------------|----------------|----------------------------|
| <code>taskName</code> | const char [ ] | Name assigned to the task. |

*Output*

| Name                    | Type         | Description                                       |
|-------------------------|--------------|---|
| <code>taskHandle</code> | TaskHandle * | A reference to the task created in this function. |

## Return Value

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

|                     |       |   |
|---------------------|-------|---|
| <code>status</code> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |
|---------------------|-------|---|

## DAQmxBaseIsTaskDone

```
int32 DAQmxBaseIsTaskDone (TaskHandle taskHandle, bool32 *isTaskDone);
```

## Purpose

Queries whether the task completed execution. Use this function to ensure that the specified operation is complete before you stop the task.

## Parameters

*Input*

| Name                    | Type       | Description                     |
|-------------------------|------------|---------------------------------|
| <code>taskHandle</code> | TaskHandle | The task used in this function. |

*Output*

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

**isTaskDone** bool32 \* Indicates whether the measurement or generated completed.

## Return Value

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

|               |       |   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |
|---------------|-------|---|

## DAQmxBaseLoadTask

```
int32 DAQmxBaseLoadTask (const char taskName[ ], TaskHandle *taskHandle);
```

### Purpose

Loads an existing named task created by you with the NI-DAQmx Base Task Configuration Utility. If you use this function to load a task, you must use `DAQmxBaseClearTask` to destroy it.

### Parameters

*Input*

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

|                 |                |   |
|-----------------|----------------|---|
| <b>taskName</b> | const char [ ] | A named task in the NI-DAQmx Base Task Configuration Utility. |
|-----------------|----------------|---|

*Output*

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

|                   |              |  |
|-------------------|--------------|--|
| <b>taskHandle</b> | TaskHandle * | A reference to the task returned by this function. |
|-------------------|--------------|--|

### Return Value

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

|               |       |   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |
|---------------|-------|---|

## DAQmxBaseResetDevice

```
int32 DAQmxBaseResetDevice (const char deviceName[ ]);
```

### Purpose

Immediately aborts all tasks associated with a device and returns the device to an initialized state. Aborting a task stops and releases any resources the task reserved.

### Parameters

*Input*

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

|                   |                |   |
|-------------------|----------------|---|
| <b>deviceName</b> | const char [ ] | The name of the device to which this operation applies. |
|-------------------|----------------|---|

### Return Value

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

|               |       |   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |
|---------------|-------|---|

## DAQmxBaseStartTask

```
int32 DAQmxBaseStartTask (TaskHandle taskHandle);
```

### Purpose

Transitions the task from the committed state to the running state, which begins measurement or generation. Using this function is required for all NI-DAQmx Base applications. This function is not required if you are using a `DAQmxBase Write` function with `autoStart` set to `TRUE`.

### Parameters

*Input*

| Name              | Type       | Description        |
|-------------------|------------|--------------------|
| <b>taskHandle</b> | TaskHandle | The task to start. |

**Return Value**

| Name          | Type  | Description   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |

**DAQmxBaseStopTask**

```
int32 DAQmxBaseStopTask (TaskHandle taskHandle);
```

**Purpose**

Stops the task and returns it to the state it was in before you called `DAQmxBaseStartTask` or called an `DAQmxBase Write` function with **autoStart** set to TRUE. Using this function is required for all NI-DAQmx Base applications.

**Parameters***Input*

| Name              | Type       | Description       |
|-------------------|------------|-------------------|
| <b>taskHandle</b> | TaskHandle | The task to stop. |

**Return Value**

| Name          | Type  | Description   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |

**DAQmxBaseCreateAIThrmcpIChan**

```
int32 DAQmxBaseCreateAIThrmcpIChan (TaskHandle taskHandle, const char physicalChannel[ ], const char nameToAssignToChannel[ ], float64 minVal, float64 maxVal, int32 units, int32 thermocoupleType, int32 cjcSource, float64 cjcVal, const char cjcChannel[ ]);
```

**Purpose**

This function is only valid for a NI USB-9211 device. Creates channel(s) that use a thermocouple to measure temperature and adds the channel(s) to the task you specify with **taskHandle**.

**Parameters***Input*

| Name                         | Type               | Description  |      |             |                |                 |                |                    |                   |         |
|------------------------------|--------------------|--|------|-------------|----------------|-----------------|----------------|--------------------|-------------------|---------|
| <b>taskHandle</b>            | TaskHandle         | The task to which to add the channels that this function creates.  |      |             |                |                 |                |                    |                   |         |
| <b>physicalChannel</b>       | const char [ ]     | The names of the physical channels to use to create virtual channels. You can specify a list or range of physical channels such as the following: <code>Dev1/ai0:3</code> or <code>Dev1/ai0,Dev1/ai2</code>  |      |             |                |                 |                |                    |                   |         |
| <b>nameToAssignToChannel</b> | const char [ ]     | Pass NULL for this parameter. NI-DAQmx Base currently ignores this parameter.  |      |             |                |                 |                |                    |                   |         |
| <b>minVal</b>                | float64            | The minimum value, in <b>volts</b> , that you expect to measure.   |      |             |                |                 |                |                    |                   |         |
| <b>maxVal</b>                | float64            | The maximum value, in <b>volts</b> , that you expect to measure.   |      |             |                |                 |                |                    |                   |         |
| <b>units</b>                 | int32              | The units to use to return the measurement.  |      |             |                |                 |                |                    |                   |         |
|                              |                    | <table> <thead> <tr> <th>Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>DAQmx_Val_DegC</td> <td>Degrees Celsius</td> </tr> <tr> <td>DAQmx_Val_DegF</td> <td>Degrees Fahrenheit</td> </tr> <tr> <td>DAQmx_Val_Kelvins</td> <td>Kelvins</td> </tr> </tbody> </table> | Name | Description | DAQmx_Val_DegC | Degrees Celsius | DAQmx_Val_DegF | Degrees Fahrenheit | DAQmx_Val_Kelvins | Kelvins |
| Name                         | Description        |  |      |             |                |                 |                |                    |                   |         |
| DAQmx_Val_DegC               | Degrees Celsius    |  |      |             |                |                 |                |                    |                   |         |
| DAQmx_Val_DegF               | Degrees Fahrenheit |  |      |             |                |                 |                |                    |                   |         |
| DAQmx_Val_Kelvins            | Kelvins            |  |      |             |                |                 |                |                    |                   |         |

|                         |                |  |   |
|-------------------------|----------------|--|---|
|                         |                | DAQmx_Val_DegR   | Degrees Rankin  |
| <b>thermocoupleType</b> | int32          | The type of thermocouple connected to the channel.   |   |
|                         |                | <b>Value</b>   | <b>Description</b>  |
|                         |                | DAQmx_Val_J_Type_TC  | J-type thermocouple.  |
|                         |                | DAQmx_Val_K_Type_TC  | K-type thermocouple.  |
|                         |                | DAQmx_Val_N_Type_TC  | N-type thermocouple.  |
|                         |                | DAQmx_Val_R_Type_TC  | R-type thermocouple.  |
|                         |                | DAQmx_Val_S_Type_TC  | S-type thermocouple.  |
|                         |                | DAQmx_Val_T_Type_TC  | T-type thermocouple.  |
|                         |                | DAQmx_Val_B_Type_TC  | B-type thermocouple.  |
|                         |                | DAQmx_Val_E_Type_TC  | E-type thermocouple.  |
| <b>cjcSource</b>        | int32          | The source of cold junction compensation.  |   |
|                         |                | <b>Value</b>   | <b>Description</b>  |
|                         |                | DAQmx_Val_BuiltIn  | Use a cold-junction compensation channel built into the terminal block. |
|                         |                | DAQmx_Val_ConstVal   | You must specify the cold-junction temperature.                         |
|                         |                | DAQmx_Val_Chan   | Use a channel for cold-junction compensation.                           |
| <b>cjcVal</b>           | float64        | The temperature of the cold junction of the thermocouple if you set <b>cjcSource</b> to DAQmx_Val_ConstVal.                |   |
| <b>cjcChannel</b>       | const char [ ] | The channel that acquires the temperature of the thermocouple cold-junction if you set <b>cjcSource</b> to DAQmx_Val_Chan. |   |

## Return Value

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

|               |       |   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |
|---------------|-------|---|

## DAQmxBaseCreateAIVoltageChan

int32 DAQmxBaseCreateAIVoltageChan (TaskHandle taskHandle, const char physicalChannel[ ], const char nameToAssignToChannel[ ], int32 terminalConfig, float64 minVal, float64 maxVal, int32 units, const char customScaleName[ ]);

### Purpose

Creates channel(s) for voltage measurement and adds the channel(s) to the task you specify with **taskHandle**.

### Parameters

*Input*

| Name                         | Type           | Description   |
|------------------------------|----------------|---|
| <b>taskHandle</b>            | TaskHandle     | The task to which to add the channels that this function creates.   |
| <b>physicalChannel</b>       | const char [ ] | The names of the physical channels to use to create virtual channels. You can specify a list or range of physical channels such as the following: Dev1/ai0:3 or Dev1/ai0,Dev1/ai2 |
| <b>nameToAssignToChannel</b> | const char [ ] | Pass NULL for this parameter. NI-DAQmx Base currently ignores this parameter.   |
| <b>terminalConfig</b>        | int32          | The input terminal configuration for the channel.   |
|                              |                | <b>Value</b>  |
|                              |                | DAQmx_Val_Cfg_Default   |
|                              |                | <b>Description</b>  |
|                              |                | At run time, NI-DAQmx Base chooses the  |

|                        |                |  |   |
|------------------------|----------------|--|---|
|                        |                | (-1)   | default input terminal configuration for the channel. On E Series devices, if the channel supports differential mode, NI-DAQmx Base chooses DAQmx_Val_Diff. Otherwise, NI-DAQmx Base chooses DAQmx_Val_RSE. |
|                        |                | DAQmx_Val_RSE  | Referenced single-ended mode  |
|                        |                | DAQmx_Val_NRSE   | Non-referenced single-ended mode  |
|                        |                | DAQmx_Val_Diff   | Differential mode   |
| <b>minVal</b>          | float64        | The minimum value, in <b>volts</b> , that you expect to measure. |   |
| <b>maxVal</b>          | float64        | The maximum value, in <b>volts</b> , that you expect to measure. |   |
| <b>units</b>           | int32          | Always pass DAQmx_Val_Volts                                      |   |
|                        |                | <b>Name</b>  | <b>Description</b>  |
|                        |                | DAQmx_Val_Volts.   | Volts   |
| <b>customScaleName</b> | const char [ ] | Pass NULL for this parameter.                                    |   |

## Return Value

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

|               |       |   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |
|---------------|-------|---|

## DAQmxBaseCreateAOVoltageChan

int32 DAQmxBaseCreateAOVoltageChan (TaskHandle taskHandle, const char physicalChannel[ ], const char nameToAssignToChannel[ ], float64 minVal, float64 maxVal, int32 units, const char customScaleName[ ]);

### Purpose

Creates channel(s) to generate voltage and adds the channel(s) to the task you specify with **taskHandle**.

### Parameters

*Input*

| Name                         | Type           | Description   |       |
|------------------------------|----------------|---|-------|
| <b>taskHandle</b>            | TaskHandle     | The task to which to add the channels that this function creates.   |       |
| <b>physicalChannel</b>       | const char [ ] | The names of the physical channels to use to create virtual channels. You can specify a list or range of physical channels such as the following: Dev1/ao0:1 or Dev1/ao0,Dev1/ao2 |       |
| <b>nameToAssignToChannel</b> | const char [ ] | Pass NULL for this parameter. NI-DAQmx Base currently ignores this parameter.   |       |
| <b>minVal</b>                | float64        | The minimum value, in <b>volts</b> , that you expect to generate.   |       |
| <b>maxVal</b>                | float64        | The maximum value, in <b>volts</b> , that you expect to generate.   |       |
| <b>units</b>                 | int32          | Always pass DAQmx_Val_Volts.  |       |
|                              |                | <b>Name</b>   |       |
|                              |                | <b>Description</b>  |       |
|                              |                | DAQmx_Val_Volts   | Volts |
| <b>customScaleName</b>       | const char [ ] | Pass NULL for this parameter.   |       |

## Return Value

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

|               |       |   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |
|---------------|-------|---|

## DAQmxBaseCreateDIChan



```
int32 DAQmxBaseCreateDIChan (TaskHandle taskHandle, const char lines[ ], const char nameToAssignToLines[ ], int32 lineGrouping);
```

## Purpose

Creates channel(s) to measure digital signals and adds the channel(s) to the task you specify with **taskHandle**. For some devices (such as E Series), NI-DAQmx Base supports grouping the digital lines of a port as single channel, not multiple channels.

## Parameters

*Input*

| Name                       | Type                      | Description  |
|----------------------------|---------------------------|--|
| <b>taskHandle</b>          | TaskHandle                | The task to which to add the channels that this function creates.  |
| <b>lines</b>               | const char [ ]            | The names of the digital lines used to create a virtual channel. You can specify a list or range of lines such as the following: Dev1/port0:1 or Dev1/port0,Dev1/port2 or Dev1/port0/line0:4 |
| <b>nameToAssignToLines</b> | const char [ ]            | Pass NULL for this parameter. NI-DAQmx Base currently ignores this parameter.  |
| <b>lineGrouping</b>        | int32                     | Always pass DAQmx_Val_ChanForAllLines.   |
|                            | <b>Value</b>              | <b>Description</b>   |
|                            | DAQmx_Val_ChanForAllLines | One channel for all lines  |

## Return Value

| Name          | Type  | Description   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |

## DAQmxBaseCreateDOChan

```
int32 DAQmxBaseCreateDOChan (TaskHandle taskHandle, const char lines[ ], const char nameToAssignToLines[ ], int32 lineGrouping);
```

## Purpose

Creates channel(s) to generate digital signals and adds the channel(s) to the task you specify with **taskHandle**. For some devices (such as E Series), NI-DAQmx Base supports grouping the digital lines of a port as a single channel, not multiple channels.

## Parameters

*Input*

| Name                       | Type                      | Description  |
|----------------------------|---------------------------|--|
| <b>taskHandle</b>          | TaskHandle                | The task to which to add the channels that this function creates.  |
| <b>lines</b>               | const char [ ]            | The names of the digital lines used to create a virtual channel. You can specify a list or range of lines such as the following: Dev1/port0:1 or Dev1/port0,Dev1/port2 or Dev1/port0/line0:4 |
| <b>nameToAssignToLines</b> | const char [ ]            | Pass NULL for this parameter. NI-DAQmx Base currently ignores this parameter.  |
| <b>lineGrouping</b>        | int32                     | Always pass DAQmx_Val_ChanForAllLines.   |
|                            | <b>Value</b>              | <b>Description</b>   |
|                            | DAQmx_Val_ChanForAllLines | One channel for all lines  |

## Return Value

| Name          | Type  | Description   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |

## DAQmxBaseCreateCIPeriodChan


```
int32 DAQmxBaseCreateCIPeriodChan (TaskHandle taskHandle, const char counter[ ], const char nameToAssignToChannel
[ ], float64 minVal, float64 maxVal, int32 units, int32 edge, int32 measMethod, float64 measTime, uInt32 divisor, const
char customScaleName[ ]);
```

## Purpose

Creates a channel to measure the period of a digital signal and adds the channel to the task you specify with **taskHandle**. You can create only one counter input channel at a time with this function because a task can include only one counter input channel. To read from multiple counters simultaneously, use a separate task for each counter. Connect the input signal to the default input terminal of the counter.

## Parameters

### Input

| Name                              | Type   | Description  |       |             |                                   |  |                                |                 |
|-----------------------------------|--|--|-------|-------------|-----------------------------------|--|--------------------------------|-----------------|
| <b>taskHandle</b>                 | TaskHandle   | The task to which to add the channels that this function creates.  |       |             |                                   |  |                                |                 |
| <b>counter</b>                    | const char [ ]   | The name of the counter to use to create virtual channels such as <code>Dev1/ctr0</code> .   |       |             |                                   |  |                                |                 |
| <b>nameToAssignToChannel</b>      | const char [ ]   | Pass NULL for this parameter. NI-DAQmx Base currently ignores this parameter.  |       |             |                                   |  |                                |                 |
| <b>minVal</b>                     | float64  | The minimum value, in <b>units</b> , that you expect to measure.   |       |             |                                   |  |                                |                 |
| <b>maxVal</b>                     | float64  | The maximum value, in <b>units</b> , that you expect to measure.   |       |             |                                   |  |                                |                 |
| <b>units</b>                      | int32  | The units to use to return the measurement. <table border="1" data-bbox="617 819 1104 966"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><code>DAQmx_Val_Seconds</code></td> <td>Seconds</td> </tr> <tr> <td><code>DAQmx_Val_Ticks</code></td> <td>Timebase ticks</td> </tr> </tbody> </table>   | Value | Description | <code>DAQmx_Val_Seconds</code>    | Seconds  | <code>DAQmx_Val_Ticks</code>   | Timebase ticks  |
| Value                             | Description  |  |       |             |                                   |  |                                |                 |
| <code>DAQmx_Val_Seconds</code>    | Seconds  |  |       |             |                                   |  |                                |                 |
| <code>DAQmx_Val_Ticks</code>      | Timebase ticks   |  |       |             |                                   |  |                                |                 |
| <b>edge</b>                       | int32  | Specifies between which edges to measure the frequency or period of the signal. <table border="1" data-bbox="617 1029 1104 1155"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><code>DAQmx_Val_Rising</code></td> <td>Rising edge(s)</td> </tr> <tr> <td><code>DAQmx_Val_Falling</code></td> <td>Falling edge(s)</td> </tr> </tbody> </table>      | Value | Description | <code>DAQmx_Val_Rising</code>     | Rising edge(s)   | <code>DAQmx_Val_Falling</code> | Falling edge(s) |
| Value                             | Description  |  |       |             |                                   |  |                                |                 |
| <code>DAQmx_Val_Rising</code>     | Rising edge(s)   |  |       |             |                                   |  |                                |                 |
| <code>DAQmx_Val_Falling</code>    | Falling edge(s)  |  |       |             |                                   |  |                                |                 |
| <b>measMethod</b>                 | int32  | Always pass <code>DAQmx_Val_LowFreqCtr</code> . <table border="1" data-bbox="617 1197 1104 1323"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><code>DAQmx_Val_LowFreqCtr</code></td> <td>Use one counter that uses a constant timebase to measure the input signal.</td> </tr> </tbody> </table>  | Value | Description | <code>DAQmx_Val_LowFreqCtr</code> | Use one counter that uses a constant timebase to measure the input signal. |                                |                 |
| Value                             | Description  |  |       |             |                                   |  |                                |                 |
| <code>DAQmx_Val_LowFreqCtr</code> | Use one counter that uses a constant timebase to measure the input signal. |  |       |             |                                   |  |                                |                 |
| <b>measTime</b>                   | float64  | Always pass 0 for this parameter. <div data-bbox="633 1365 1412 1470" style="border: 1px solid red; padding: 5px; margin-top: 10px;">  <b>Warning</b> If you measure a high-frequency signal for too long a time, the count register could roll over, resulting in an incorrect measurement. </div> |       |             |                                   |  |                                |                 |
| <b>divisor</b>                    | uInt32   | Always pass 1 for this parameter.  |       |             |                                   |  |                                |                 |
| <b>customScaleName</b>            | const char [ ]   | Pass NULL for this parameter.  |       |             |                                   |  |                                |                 |

## Return Value

| Name          | Type  | Description   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |

## DAQmxBaseCreateCICountEdgesChan

```
int32 DAQmxBaseCreateCICountEdgesChan (TaskHandle taskHandle, const char counter[ ], const char
nameToAssignToChannel[ ], int32 edge, uInt32 initialCount, int32 countDirection);
```

## Purpose

Creates a channel to count the number of rising or falling edges of a digital signal and adds the channel to the task you

specify with **taskHandle**. You can create only one counter input channel at a time with this function because a task can include only one counter input channel. To read from multiple counters simultaneously, use a separate task for each counter. Connect the input signal to the default input terminal of the counter.

## Parameters

### Input

| Name                         | Type                    | Description  |
|------------------------------|-------------------------|--|
| <b>taskHandle</b>            | TaskHandle              | The task to which to add the channels that this function creates.  |
| <b>counter</b>               | const char [ ]          | The name of the counter to use to create virtual channels such as <code>Dev1/ctr0</code> .   |
| <b>nameToAssignToChannel</b> | const char [ ]          | Pass NULL for this parameter. NI-DAQmx Base currently ignores this parameter.  |
| <b>edge</b>                  | int32                   | Specifies on which edges of the input signal to increment or decrement the count.  |
|                              | <b>Value</b>            | <b>Description</b>   |
|                              | DAQmx_Val_Rising        | Rising edge(s).  |
|                              | DAQmx_Val_Falling       | Falling edge(s).   |
| <b>initialCount</b>          | uInt32                  | The value from which to start counting.  |
| <b>countDirection</b>        | int32                   | Specifies whether to increment or decrement the counter on each edge.  |
|                              | <b>Value</b>            | <b>Description</b>   |
|                              | DAQmx_Val_CountUp       | Increment the count register on each edge.   |
|                              | DAQmx_Val_CountDown     | Decrement the count register on each edge.   |
|                              | DAQmx_Val_ExtControlled | The state of a digital line controls the count direction. Each counter has a default count direction terminal. The default for Counter 0 is PFI 10. The default for Counter 1 is PFI 11. |

## Return Value

| Name          | Type  | Description   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |

## DAQmxBaseCreateCIPulseWidthChan

```
int32 DAQmxBaseCreateCIPulseWidthChan (TaskHandle taskHandle, const char counter[ ], const char nameToAssignToChannel[ ], float64 minVal, float64 maxVal, int32 units, int32 startingEdge, const char customScaleName [ ]);
```

### Purpose

Creates a channel to measure the width of a digital pulse and adds the channel to the task you specify with **taskHandle**. **startingEdge** determines whether to measure a high pulse or low pulse. You can create only one counter input channel at a time with this function because a task can include only one counter input channel. To read from multiple counters simultaneously, use a separate task for each counter. Connect the input signal to the default input terminal of the counter.

## Parameters

### Input

| Name                         | Type           | Description  |
|------------------------------|----------------|--|
| <b>taskHandle</b>            | TaskHandle     | The task to which to add the channels that this function creates.                          |
| <b>counter</b>               | const char [ ] | The name of the counter to use to create virtual channels such as <code>Dev1/ctr0</code> . |
| <b>nameToAssignToChannel</b> | const char [ ] | Pass NULL for this parameter. NI-DAQmx Base currently ignores this parameter.              |

|               |         |  |
|---------------|---------|--|
| <b>minVal</b> | float64 | The minimum value, in <b>units</b> , that you expect to measure. |
| <b>maxVal</b> | float64 | The maximum value, in <b>units</b> , that you expect to measure. |
| <b>units</b>  | int32   | The units to use to return the measurement.                      |

| Value | Description |
|-------|-------------|
|-------|-------------|

|                   |         |
|-------------------|---------|
| DAQmx_Val_Seconds | Seconds |
|-------------------|---------|

|                 |                |
|-----------------|----------------|
| DAQmx_Val_Ticks | Timebase ticks |
|-----------------|----------------|

|                     |       |   |
|---------------------|-------|---|
| <b>startingEdge</b> | int32 | Specifies on which edge to begin measuring pulse width. |
|---------------------|-------|---|

| Value | Description |
|-------|-------------|
|-------|-------------|

|                  |                |
|------------------|----------------|
| DAQmx_Val_Rising | Rising edge(s) |
|------------------|----------------|

|                   |                 |
|-------------------|-----------------|
| DAQmx_Val_Falling | Falling edge(s) |
|-------------------|-----------------|

|                        |                |                               |
|------------------------|----------------|-------------------------------|
| <b>customScaleName</b> | const char [ ] | Pass NULL for this parameter. |
|------------------------|----------------|-------------------------------|

## Return Value

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

|               |       |   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |
|---------------|-------|---|

## DAQmxBaseCreateCILinEncoderChan

```
int32 DAQmxBaseCreateCILinEncoderChan (TaskHandle taskHandle, const char counter[], const char
nameToAssignToChannel[], int32 decodingType, bool32 ZidxEnable, float64 ZidxVal, int32 ZidxPhase, int32 units, float64
distPerPulse, float64 initialPos, const char customScaleName[]);
```

## Purpose

Creates a channel that uses a linear encoder to measure linear position. You can create only one counter input channel at a time with this function because a task can include only one counter input channel. To read from multiple counters simultaneously, use a separate task for each counter. Connect the input signals to the default input terminals of the counter unless you select different input terminals.

## Parameters

*Input*

| Name                         | Type           | Description  |
|------------------------------|----------------|--|
| <b>taskHandle</b>            | TaskHandle     | The task to which to add the channels that this function creates.  |
| <b>counter</b>               | const char [ ] | The name of the counter to use to create virtual channels such as Dev1/ctr0.   |
| <b>nameToAssignToChannel</b> | const char [ ] | The name to assign to the created virtual channel. If you specify your own names for <b>nameToAssignToChannel</b> , you must use the names when you refer to these channels in other NI-DAQmxBase functions. |

If you create multiple virtual channels with one call to this function, you can specify a list of names separated by commas. If you provide fewer names than the number of virtual channels you create, NI-DAQmxBase automatically assigns names to the virtual channels.

|                     |       |  |
|---------------------|-------|--|
| <b>decodingType</b> | int32 | Specifies how to count and interpret the pulses that the encoder generates on signal A and signal B. DAQmx_Val_X1, DAQmx_Val_X2, and DAQmx_Val_X4 are valid for quadrature encoders only. DAQmx_Val_TwoPulseCounting is valid only for two-pulse encoders. |
|---------------------|-------|--|

DAQmx\_Val\_X2 and DAQmx\_Val\_X4 decoding are more sensitive to smaller changes in position than DAQmx\_Val\_X1 encoding, with DAQmx\_Val\_X4 being the most sensitive. However, more sensitive decoding is more likely to produce erroneous measurements if there is vibration in the encoder or other noise in the signals.

| Value | Description |
|-------|-------------|
|-------|-------------|

|              |   |
|--------------|---|
| DAQmx_Val_X1 | If signal A leads signal B, count the rising edges of signal A. If signal B |
|--------------|---|

|                        |                            |   |
|------------------------|----------------------------|---|
|                        |                            | leads signal A, count the falling edges of signal A.  |
|                        | DAQmx_Val_X2               | Count the rising and falling edges of signal A.   |
|                        | DAQmx_Val_X4               | Count the rising and falling edges of both signal A and signal B.   |
|                        | DAQmx_Val_TwoPulseCounting | Increment the count on rising edges of signal A. Decrement the count on rising pulses of signal B.  |
| <b>ZidxEnable</b>      | const char []              | Specifies whether to enable z indexing for the measurement.   |
| <b>ZidxVal</b>         | float64                    | The value, in <b>units</b> , to which to reset the measurement when signal Z is high and signal A and signal B are at the states you specify with <b>ZidxPhase</b> .  |
| <b>ZidxPhase</b>       | int32                      | The states at which signal A and signal B must be while signal Z is high for NI-DAQmxBase to reset the measurement. If signal Z is never high while the signal A and signal B are high, for example, you must choose a phase other than DAQmx_Val_AHighBHigh.<br><br>When signal Z goes high and how long it stays high varies from encoder to encoder. Refer to the documentation for the encoder to determine the timing of signal Z with respect to signal A and signal B. |
|                        | <b>Value</b>               | <b>Description</b>  |
|                        | DAQmx_Val_AHighBHigh       | Reset the measurement when both signal A and signal B are at high logic.  |
|                        | DAQmx_Val_AHighBLow        | Reset the measurement when signal A is at high logic and signal B is at low logic.  |
|                        | DAQmx_Val_ALowBHigh        | Reset the measurement when signal A is at low logic and signal B is at high logic.  |
|                        | DAQmx_Val_ALowBLow         | Reset the measurement when both signal A and signal B are at low logic.   |
| <b>units</b>           | int32                      | The units to use to return linear position measurements from the channel.   |
|                        | <b>Name</b>                | <b>Description</b>  |
|                        | DAQmx_Val_Meters           | Meters.   |
|                        | DAQmx_Val_Inches           | Inches.   |
|                        | DAQmx_Val_Ticks            | Timebase Ticks.   |
|                        | DAQmx_Val_FromCustomScale  | Units a custom scale specifies. Use <b>customScaleName</b> to specify a custom scale.   |
| <b>distPerPulse</b>    | float64                    | The distance measured for each pulse the encoder generates. Specify this value in <b>units</b> .  |
| <b>initialPos</b>      | float64                    | The position of the encoder when the measurement begins. This value is in <b>units</b> .  |
| <b>customScaleName</b> | const char [ ]             | Pass NULL for this parameter.   |

## Return Value

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

|               |       |   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |
|---------------|-------|---|

## DAQmxBaseCreateCIAngEncoderChan

```
int32 DAQmxBaseCreateCIAngEncoderChan (TaskHandle taskHandle, const char counter[], const char nameToAssignToChannel[], int32 decodingType, bool32 ZidxEnable, float64 ZidxVal, int32 ZidxPhase, int32 units, uInt32 pulsesPerRev, float64 initialAngle, const char customScaleName[]);
```

## Purpose

Creates a channel that uses an angular encoder to measure angular position. You can create only one counter input channel at a time with this function because a task can include only one counter input channel. To read from multiple counters simultaneously, use a separate task for each counter. Connect the input signals to the default input terminals of the counter unless you select different input terminals.

## Parameters

*Input*

| Name                         | Type           | Description  |
|------------------------------|----------------|--|
| <b>taskHandle</b>            | TaskHandle     | The task to which to add the channels that this function creates.  |
| <b>counter</b>               | const char [ ] | The name of the counter to use to create virtual channels such as <code>Dev1/ctr0</code> .   |
| <b>nameToAssignToChannel</b> | const char [ ] | The name to assign to the created virtual channel. If you specify your own names for <b>nameToAssignToChannel</b> , you must use the names when you refer to these channels in other NI-DAQmxBase functions. |

If you create multiple virtual channels with one call to this function, you can specify a list of names separated by commas. If you provide fewer names than the number of virtual channels you create, NI-DAQmxBase automatically assigns names to the virtual channels.

|                     |       |  |
|---------------------|-------|--|
| <b>decodingType</b> | int32 | Specifies how to count and interpret the pulses that the encoder generates on signal A and signal B. <code>DAQmx_Val_X1</code> , <code>DAQmx_Val_X2</code> , and <code>DAQmx_Val_X4</code> are valid for quadrature encoders only. <code>DAQmx_Val_TwoPulseCounting</code> is valid only for two-pulse encoders. |
|---------------------|-------|--|

`DAQmx_Val_X2` and `DAQmx_Val_X4` decoding are more sensitive to smaller changes in position than `DAQmx_Val_X1` encoding, with `DAQmx_Val_X4` being the most sensitive. However, more sensitive decoding is more likely to produce erroneous measurements if there is vibration in the encoder or other noise in the signals.

| Value                                   | Description  |
|---|--|
| <code>DAQmx_Val_X1</code>               | If signal A leads signal B, count the rising edges of signal A. If signal B leads signal A, count the falling edges of signal A. |
| <code>DAQmx_Val_X2</code>               | Count the rising and falling edges of signal A.  |
| <code>DAQmx_Val_X4</code>               | Count the rising and falling edges of both signal A and signal B.  |
| <code>DAQmx_Val_TwoPulseCounting</code> | Increment the count on rising edges of signal A. Decrement the count on rising pulses of signal B.                               |

|                   |         |   |
|-------------------|---------|---|
| <b>ZidxEnable</b> | bool32  | Specifies whether to enable z indexing for the measurement.   |
| <b>ZidxVal</b>    | float64 | The value, in <b>units</b> , to which to reset the measurement when signal Z is high and signal A and signal B are at the states you specify with <b>ZidxPhase</b> .  |
| <b>ZidxPhase</b>  | int32   | The states at which signal A and signal B must be while signal Z is high for NI-DAQmxBase to reset the measurement. If signal Z is never high while the signal A and signal B are high, for example, you must choose a phase other than <code>DAQmx_Val_AHighBHigh</code> . |

When signal Z goes high and how long it stays high varies from encoder to encoder. Refer to the documentation for the encoder to determine the timing of signal Z with respect to signal A and signal B.

| Value                             | Description  |
|-----------------------------------|--|
| <code>DAQmx_Val_AHighBHigh</code> | Reset the measurement when both signal A and signal B are at high logic.           |
| <code>DAQmx_Val_AHighBLow</code>  | Reset the measurement when signal A is at high logic and signal B is at low logic. |

|                        |                |  |   |
|------------------------|----------------|--|---|
|                        |                | DAQmx_Val_ALowBHigh  | Reset the measurement when signal A is at low logic and signal B is at high logic.    |
|                        |                | DAQmx_Val_ALowBLow   | Reset the measurement when both signal A and signal B are at low logic.               |
| <b>units</b>           | int32          | The units to use to return angular position measurements from the channel.   |   |
|                        |                | <b>Value</b>   | <b>Description</b>  |
|                        |                | DAQmx_Val_Degrees  | Degrees   |
|                        |                | DAQmx_Val_Radians  | Radians   |
|                        |                | DAQmx_Val_Ticks  | Timebase ticks  |
|                        |                | DAQmx_Val_FromCustomScale  | Units a custom scale specifies. Use <b>customScaleName</b> to specify a custom scale. |
| <b>pulsesPerRev</b>    | uInt32         | The number of pulses the encoder generates per revolution. This value is the number of pulses on one of either A signal or B signal, not the total number of pulses on both signal A and signal B. |   |
| <b>initialAngle</b>    | float64        | The starting angle of the encoder when the measurement begins. Specify this value in <b>units</b> .  |   |
| <b>customScaleName</b> | const char [ ] | Pass NULL for this parameter.  |   |

## Return Value

**Name** **Type** **Description**

**status** int32 The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error.

## DAQmxBaseCreateCOPulseChanFreq

int32 DAQmxBaseCreateCOPulseChanFreq (TaskHandle taskHandle, const char counter[ ], const char nameToAssignToChannel[ ], int32 units, int32 idleState, float64 initialDelay, float64 freq, float64 dutyCycle);

### Purpose

Creates a channel to generate digital pulses defined by **freq** and **dutyCycle** and adds the channel to the task you specify with **taskHandle**. The pulses appear on the default output terminal of the counter.

You can create only one counter output channel at a time with this function because a task can include only one counter output channel. To use multiple counters simultaneously, use a separate task for each counter.

### Parameters

*Input*

| <b>Name</b>                  | <b>Type</b>    | <b>Description</b>  |
|------------------------------|----------------|---|
| <b>taskHandle</b>            | TaskHandle     | The task to which to add the channels that this function creates.             |
| <b>counter</b>               | const char [ ] | The name of the counter to use to create virtual channels such as Dev1/ctr0.  |
| <b>nameToAssignToChannel</b> | const char [ ] | Pass NULL for this parameter. NI-DAQmx Base currently ignores this parameter. |
| <b>units</b>                 | int32          | The units in which to specify <b>freq</b> .                                   |
|                              |                | <b>Name</b> <b>Description</b>  |
|                              |                | DAQmx_Val_Hz Hertz  |
| <b>idleState</b>             | int32          | The resting state of the output terminal.                                     |
|                              |                | <b>Value</b> <b>Description</b>   |
|                              |                | DAQmx_Val_High High state. M Series only.                                     |
|                              |                | DAQmx_Val_Low Low state   |

|                     |         |  |
|---------------------|---------|--|
| <b>initialDelay</b> | float64 | The amount of time in seconds to wait before generating the first pulse.   |
| <b>freq</b>         | float64 | The frequency of the pulses to generate.   |
| <b>dutyCycle</b>    | float64 | The width of the pulse divided by the pulse period. NI-DAQmx Base uses this ratio, combined with frequency, to determine both pulse width and pulse delay. |

## Return Value

**Name** **Type** **Description**

**status** int32 The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error.

## DAQmxBaseCfgSampClkTiming

int32 DAQmxBaseCfgSampClkTiming (TaskHandle taskHandle, const char source[ ], float64 rate, int32 activeEdge, int32 sampleMode, uInt64 sampsPerChanToAcquire);

### Purpose

Sets the source of the Sample Clock, the rate of the Sample Clock, and the number of samples to acquire or generate.

### Parameters

*Input*

| <b>Name</b>                  | <b>Type</b>  | <b>Description</b>   |              |                    |                       |  |                     |  |
|------------------------------|--|--|--------------|--------------------|-----------------------|--|---------------------|--|
| <b>taskHandle</b>            | TaskHandle   | The task used in this function.  |              |                    |                       |  |                     |  |
| <b>source</b>                | const char [ ]   | The source terminal of the Sample Clock.   |              |                    |                       |  |                     |  |
| <b>rate</b>                  | float64  | The sampling rate in samples per second. If you use an external source for the Sample Clock, set this value to the maximum expected rate of that clock.  |              |                    |                       |  |                     |  |
| <b>activeEdge</b>            | int32  | Specifies on which edge of the clock to acquire samples. <table> <thead> <tr> <th><b>Value</b></th> <th><b>Description</b></th> </tr> </thead> <tbody> <tr> <td>DAQmx_Val_Rising</td> <td>Acquire samples on the rising edges of the Sample Clock.</td> </tr> <tr> <td>DAQmx_Val_Falling</td> <td>Acquire samples on the falling edges of the Sample Clock.</td> </tr> </tbody> </table>   | <b>Value</b> | <b>Description</b> | DAQmx_Val_Rising      | Acquire samples on the rising edges of the Sample Clock. | DAQmx_Val_Falling   | Acquire samples on the falling edges of the Sample Clock.                      |
| <b>Value</b>                 | <b>Description</b>   |  |              |                    |                       |  |                     |  |
| DAQmx_Val_Rising             | Acquire samples on the rising edges of the Sample Clock.                       |  |              |                    |                       |  |                     |  |
| DAQmx_Val_Falling            | Acquire samples on the falling edges of the Sample Clock.                      |  |              |                    |                       |  |                     |  |
| <b>sampleMode</b>            | int32  | Specifies whether the task acquires or generates samples continuously or for a finite number of samples. <table> <thead> <tr> <th><b>Name</b></th> <th><b>Description</b></th> </tr> </thead> <tbody> <tr> <td>DAQmx_Val_FiniteSamps</td> <td>Acquire a finite number of samples.</td> </tr> <tr> <td>DAQmx_Val_ContSamps</td> <td>Acquire samples until you call the <a href="#">DAQmxBaseStopTask</a> function.</td> </tr> </tbody> </table> | <b>Name</b>  | <b>Description</b> | DAQmx_Val_FiniteSamps | Acquire a finite number of samples.                      | DAQmx_Val_ContSamps | Acquire samples until you call the <a href="#">DAQmxBaseStopTask</a> function. |
| <b>Name</b>                  | <b>Description</b>   |  |              |                    |                       |  |                     |  |
| DAQmx_Val_FiniteSamps        | Acquire a finite number of samples.  |  |              |                    |                       |  |                     |  |
| DAQmx_Val_ContSamps          | Acquire samples until you call the <a href="#">DAQmxBaseStopTask</a> function. |  |              |                    |                       |  |                     |  |
| <b>sampsPerChanToAcquire</b> | uInt64   | The number of samples to acquire for each channel in the task if <b>sampleMode</b> is DAQmx_Val_FiniteSamps. NI-DAQmx Base currently supports the lower 32-bits of this parameter.   |              |                    |                       |  |                     |  |

## Return Value

**Name** **Type** **Description**

**status** int32 The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error.

## DAQmxBaseCfgImplicitTiming

int32 DAQmxBaseCfgImplicitTiming (TaskHandle taskHandle, int32 sampleMode, uInt64 sampsPerChanToAcquire);

### Purpose

Sets only the number of samples to acquire or generate without specifying timing. Typically, you should use this function when the task does not require sample timing, such as tasks that use counters for buffered frequency measurement,



buffered period measurement, or pulse train generation.

## Parameters

*Input*

| Name                         | Type   | Description  |      |             |                       |                                     |                     |  |
|------------------------------|--|--|------|-------------|-----------------------|-------------------------------------|---------------------|--|
| <b>taskHandle</b>            | TaskHandle   | The task used in this function.  |      |             |                       |                                     |                     |  |
| <b>sampleMode</b>            | int32  | Specifies whether the task acquires or generates samples continuously or for a finite number of samples.   |      |             |                       |                                     |                     |  |
|                              |  | <table border="1"> <thead> <tr> <th>Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>DAQmx_Val_FiniteSamps</td> <td>Acquire a finite number of samples.</td> </tr> <tr> <td>DAQmx_Val_ContSamps</td> <td>Acquire samples until you call the <a href="#">DAQmxBaseStopTask</a> function.</td> </tr> </tbody> </table> | Name | Description | DAQmx_Val_FiniteSamps | Acquire a finite number of samples. | DAQmx_Val_ContSamps | Acquire samples until you call the <a href="#">DAQmxBaseStopTask</a> function. |
| Name                         | Description  |  |      |             |                       |                                     |                     |  |
| DAQmx_Val_FiniteSamps        | Acquire a finite number of samples.  |  |      |             |                       |                                     |                     |  |
| DAQmx_Val_ContSamps          | Acquire samples until you call the <a href="#">DAQmxBaseStopTask</a> function. |  |      |             |                       |                                     |                     |  |
| <b>sampsPerChanToAcquire</b> | uInt64   | The number of samples to acquire for each channel in the task if <b>sampleMode</b> is <code>DAQmx_Val_FiniteSamps</code> . NI-DAQmx Base currently supports the lower 32-bits of this parameter.   |      |             |                       |                                     |                     |  |

## Return Value

| Name          | Type  | Description   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |

## DAQmxBaseDisableStartTrig

int32 DAQmxBaseDisableStartTrig (TaskHandle taskHandle);

### Purpose

Configures the task to start acquiring or generating samples immediately upon starting the task.

## Parameters

*Input*

| Name              | Type       | Description                     |
|-------------------|------------|---------------------------------|
| <b>taskHandle</b> | TaskHandle | The task used in this function. |

## Return Value

| Name          | Type  | Description   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |

## DAQmxBaseCfgDigEdgeStartTrig

int32 DAQmxBaseCfgDigEdgeStartTrig (TaskHandle taskHandle, const char triggerSource[ ], int32 triggerEdge);

### Purpose

Configures the task to start acquiring or generating samples on a rising or falling edge of a digital signal.

## Parameters

*Input*

| Name                 | Type           | Description   |
|----------------------|----------------|---|
| <b>taskHandle</b>    | TaskHandle     | The task used in this function.   |
| <b>triggerSource</b> | const char [ ] | The name of a terminal where there is a digital signal to use as the source of the trigger such as the following: /Dev1/PFI0. |
| <b>triggerEdge</b>   | int32          | Specifies on which edge of a digital signal to start acquiring or generating samples.   |
|                      | <b>Value</b>   | <b>Description</b>  |

|                   |                 |
|-------------------|-----------------|
| DAQmx_Val_Rising  | Rising edge(s)  |
| DAQmx_Val_Falling | Falling edge(s) |

## Return Value

**Name** **Type** **Description**

**status** int32 The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error.

## DAQmxBaseCfgAnlgEdgeStartTrig

int32 DAQmxBaseCfgAnlgEdgeStartTrig (TaskHandle taskHandle, const char triggerSource[ ], int32 triggerSlope, float64 triggerLevel);

### Purpose

Configures the task to start acquiring samples when an analog signal crosses the level you specify.

### Parameters

*Input*

**Name** **Type** **Description**

**taskHandle** TaskHandle The task used in this function.

**triggerSource** const char [ ] The name of a terminal where there is an analog signal to use as the source of the trigger such as the following: /Dev1/PFI0. The only terminal you can use for E Series devices is PFI0.

**triggerSlope** int32 Specifies on which slope of the signal to start acquiring samples when the signal crosses **triggerLevel**.

**Value**

**Description**

DAQmx\_Val\_RisingSlope

Trigger on the rising slope of the signal.

DAQmx\_Val\_FallingSlope

Trigger on the falling slope of the signal.

**triggerLevel** float64 The threshold at which to start acquiring samples. Specify this value in the units of the measurement. Use **triggerSlope** to specify on which slope to trigger at this threshold.

## Return Value

**Name** **Type** **Description**

**status** int32 The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error.

## DAQmxBaseCfgAnlgEdgeRefTrig

int32 DAQmxBaseCfgAnlgEdgeRefTrig (TaskHandle taskHandle, const char triggerSource[ ], int32 triggerSlope, float64 triggerLevel, uInt32 pretriggerSamples);

### Purpose

Configures the task to stop the acquisition when the device acquires all pretrigger samples, an analog signal reaches the level you specify, and the device acquires all post-trigger samples.

### Parameters

*Input*

**Name** **Type** **Description**

**taskHandle** TaskHandle The task used in this function.

**triggerSource** const char [ ] The name of a terminal where there is an analog signal to use as the source of the trigger, such as the following: /Dev1/PFI0. The only terminal you can use for E Series devices is PFI0.

**triggerSlope** int32 Specifies on which slope of the signal the Reference Trigger occurs.

|                          |         | <b>Value</b>           | <b>Description</b>   |
|--------------------------|---------|------------------------|--|
|                          |         | DAQmx_Val_RisingSlope  | Trigger on the rising slope of the signal.   |
|                          |         | DAQmx_Val_FallingSlope | Trigger on the falling slope of the signal.  |
| <b>triggerLevel</b>      | float64 |                        | Specifies at what threshold to trigger. Specify this value in the units of the measurement. Use <b>triggerSlope</b> to specify on which slope to trigger at this threshold.  |
| <b>pretriggerSamples</b> | uInt32  |                        | The minimum number of samples per channel to acquire before recognizing the Reference Trigger. The number of post-trigger samples per channel is equal to number of samples per channel in the NI-DAQmx Base Timing functions minus <b>pretriggerSamples</b> . |

## Return Value

**Name** **Type** **Description**

**status** int32 The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error.

## DAQmxBaseCfgDigEdgeRefTrig

```
int32 DAQmxBaseCfgDigEdgeRefTrig (TaskHandle taskHandle, const char triggerSource[ ], int32 triggerEdge, uInt32 pretriggerSamples);
```

## Purpose

Configures the task to stop the acquisition when the device acquires all pretrigger samples, detects a rising or falling edge of a digital signal, and acquires all posttrigger samples.

## Parameters

*Input*

| <b>Name</b>              | <b>Type</b>        | <b>Description</b>   |              |                    |                  |                |                   |                 |
|--------------------------|--------------------|--|--------------|--------------------|------------------|----------------|-------------------|-----------------|
| <b>taskHandle</b>        | TaskHandle         | The task used in this function.  |              |                    |                  |                |                   |                 |
| <b>triggerSource</b>     | const char [ ]     | The name of a terminal where there is an analog signal to use as the source of the trigger such as the following: /Dev1/PFI0. The only terminal you can use for E Series devices is PFI0.  |              |                    |                  |                |                   |                 |
| <b>triggerEdge</b>       | int32              | Specifies on which edge of the digital signal the Reference Trigger occurs. <table border="1"> <thead> <tr> <th><b>Value</b></th> <th><b>Description</b></th> </tr> </thead> <tbody> <tr> <td>DAQmx_Val_Rising</td> <td>Rising edge(s)</td> </tr> <tr> <td>DAQmx_Val_Falling</td> <td>Falling edge(s)</td> </tr> </tbody> </table> | <b>Value</b> | <b>Description</b> | DAQmx_Val_Rising | Rising edge(s) | DAQmx_Val_Falling | Falling edge(s) |
| <b>Value</b>             | <b>Description</b> |  |              |                    |                  |                |                   |                 |
| DAQmx_Val_Rising         | Rising edge(s)     |  |              |                    |                  |                |                   |                 |
| DAQmx_Val_Falling        | Falling edge(s)    |  |              |                    |                  |                |                   |                 |
| <b>pretriggerSamples</b> | uInt32             | The minimum number of samples per channel to acquire before recognizing the Reference Trigger. The number of post-trigger samples per channel is equal to number of samples per channel in the NI-DAQmx Base Timing functions minus <b>pretriggerSamples</b> .   |              |                    |                  |                |                   |                 |

## Return Value

**Name** **Type** **Description**

**status** int32 The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error.

## DAQmxBaseDisableRefTrig

```
int32 DAQmxBaseDisableRefTrig (TaskHandle taskHandle);
```

## Purpose

Disables reference triggering for the measurement or generation.

## Parameters

*Input*

| Name              | Type       | Description                     |
|-------------------|------------|---------------------------------|
| <b>taskHandle</b> | TaskHandle | The task used in this function. |

**Return Value**

| Name          | Type  | Description   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |

**DAQmxBaseReadAnalogF64**

int32 DAQmxBaseReadAnalogF64 (TaskHandle taskHandle, int32 numSampsPerChan, float64 timeout, bool32 fillMode, float64 readArray[ ], uInt32 arraySizeInSamps, int32 \*sampsPerChanRead, bool32 \*reserved);

**Purpose**

Reads multiple floating-point samples from a task that contains one or more analog input channels.

**Parameters***Input*

| Name                        | Type                               | Description  |       |             |                          |                                    |                             |                                    |
|-----------------------------|------------------------------------|--|-------|-------------|--------------------------|------------------------------------|-----------------------------|------------------------------------|
| <b>taskHandle</b>           | TaskHandle                         | The task to read samples from.   |       |             |                          |                                    |                             |                                    |
| <b>numSampsPerChan</b>      | int32                              | The number of samples, per channel, to read. If the task acquires a finite number of samples and you set this parameter to -1, the function waits for the task to acquire all requested samples, and then reads those samples.   |       |             |                          |                                    |                             |                                    |
| <b>timeout</b>              | float64                            | The amount of time, in seconds, to wait for the function to read the sample(s). This function returns an error if the timeout elapses.   |       |             |                          |                                    |                             |                                    |
| <b>fillMode</b>             | bool32                             | Specifies whether or not the samples are interleaved. <table> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>DAQmx_Val_GroupByChannel</td> <td>Group by channel (non-interleaved)</td> </tr> <tr> <td>DAQmx_Val_GroupByScanNumber</td> <td>Group by scan number (interleaved)</td> </tr> </tbody> </table> | Value | Description | DAQmx_Val_GroupByChannel | Group by channel (non-interleaved) | DAQmx_Val_GroupByScanNumber | Group by scan number (interleaved) |
| Value                       | Description                        |  |       |             |                          |                                    |                             |                                    |
| DAQmx_Val_GroupByChannel    | Group by channel (non-interleaved) |  |       |             |                          |                                    |                             |                                    |
| DAQmx_Val_GroupByScanNumber | Group by scan number (interleaved) |  |       |             |                          |                                    |                             |                                    |
| <b>arraySizeInSamps</b>     | uInt32                             | The size of the array, in samples, into which data is read.  |       |             |                          |                                    |                             |                                    |
| <b>reserved</b>             | bool32 *                           | Reserved for future use. Pass NULL to this parameter.  |       |             |                          |                                    |                             |                                    |

*Output*

| Name                    | Type        | Description   |
|-------------------------|-------------|---|
| <b>readArray</b>        | float64 [ ] | The array to read data into, organized according to <b>fillMode</b> . |
| <b>sampsPerChanRead</b> | int32 *     | The actual number of samples read from each channel.                  |

**Return Value**

| Name          | Type  | Description   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |

**DAQmxBaseReadBinaryI16**

int32 DAQmxBaseReadBinaryI16 (TaskHandle taskHandle, int32 numSampsPerChan, float64 timeout, bool32 fillMode, int16 readArray[], uInt32 arraySizeInSamps, int32 \*sampsPerChanRead, bool32 \*reserved);

**Purpose**

Reads multiple unscaled, signed 16-bit integer samples from a task that contains one or more analog input channels.

**Parameters***Input*

| Name                        | Type                               | Description   |       |             |                          |                                    |                             |                                    |
|-----------------------------|------------------------------------|---|-------|-------------|--------------------------|------------------------------------|-----------------------------|------------------------------------|
| <b>taskHandle</b>           | TaskHandle                         | The task to read samples from.  |       |             |                          |                                    |                             |                                    |
| <b>numSampsPerChan</b>      | int32                              | The number of samples, per channel, to read. The default value of -1 (DAQmx_Val_Auto) reads all available data. If <b>readArray</b> does not contain enough space, this function returns as much data as fits in <b>readArray</b> .<br><br>NI-DAQmx Base determines how many samples to read based on whether the task acquires samples continuously or acquires a finite number of samples.<br><br>If the task acquires samples continuously and you set this parameter to -1, this function reads all the samples currently available in the buffer.<br><br>If the task acquires a finite number of samples and you set this parameter to -1, the function waits for the task to acquire all requested samples, and then reads those samples. |       |             |                          |                                    |                             |                                    |
| <b>timeout</b>              | float64                            | The amount of time, in seconds, to wait for the function to read the sample(s). This function returns an error if the timeout elapses.  |       |             |                          |                                    |                             |                                    |
| <b>fillMode</b>             | bool32                             | Specifies whether or not the samples are interleaved.<br><br><table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>DAQmx_Val_GroupByChannel</td> <td>Group by channel (non-interleaved)</td> </tr> <tr> <td>DAQmx_Val_GroupByScanNumber</td> <td>Group by scan number (interleaved)</td> </tr> </tbody> </table>  | Value | Description | DAQmx_Val_GroupByChannel | Group by channel (non-interleaved) | DAQmx_Val_GroupByScanNumber | Group by scan number (interleaved) |
| Value                       | Description                        |   |       |             |                          |                                    |                             |                                    |
| DAQmx_Val_GroupByChannel    | Group by channel (non-interleaved) |   |       |             |                          |                                    |                             |                                    |
| DAQmx_Val_GroupByScanNumber | Group by scan number (interleaved) |   |       |             |                          |                                    |                             |                                    |
| <b>arraySizeInSamps</b>     | uInt32                             | The size of the array, in samples, into which data is read.   |       |             |                          |                                    |                             |                                    |
| <b>reserved</b>             | bool32 *                           | Reserved for future use. Pass NULL to this parameter.   |       |             |                          |                                    |                             |                                    |
| <i>Output</i>               |                                    |   |       |             |                          |                                    |                             |                                    |
| Name                        | Type                               | Description   |       |             |                          |                                    |                             |                                    |
| <b>readArray</b>            | int16 [ ]                          | The array to read data into, organized according to <b>fillMode</b> .   |       |             |                          |                                    |                             |                                    |
| <b>sampsPerChanRead</b>     | int32 *                            | The actual number of samples read from each channel.  |       |             |                          |                                    |                             |                                    |

## Return Value

| Name          | Type  | Description   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |

## DAQmxBaseReadCounterF64

int32 DAQmxBaseReadCounterF64 (TaskHandle taskHandle, int32 numSampsPerChan, float64 timeout, float64 readArray [ ], uInt32 arraySizeInSamps, int32 \*sampsPerChanRead, bool32 \*reserved);

## Purpose

Reads multiple floating-point samples from a counter task. Use this function when counter samples are scaled to a floating-point value, such as for frequency and period measurements.

## Parameters

*Input*

| Name                   | Type       | Description  |
|------------------------|------------|--|
| <b>taskHandle</b>      | TaskHandle | The task to read samples from.   |
| <b>numSampsPerChan</b> | int32      | The number of samples, per channel, to read. The default value of -1 (DAQmx_Val_Auto) reads all available data. If <b>readArray</b> does not contain enough space, this function returns as much data as fits in <b>readArray</b> .<br><br>NI-DAQmx Base determines how many samples to read based on whether the task acquires samples continuously or acquires a finite number of samples.<br><br>If the task acquires samples continuously and you set this parameter to -1, this function reads all the samples currently available in the buffer.<br><br>If the task acquires a finite number of samples and you set this parameter to -1, the function waits for the task to acquire all requested samples, and then reads those |

|                         |          |  |
|-------------------------|----------|--|
|                         |          | samples.   |
| <b>timeout</b>          | float64  | The amount of time, in seconds, to wait for the function to read the sample(s). This function returns an error if the timeout elapses. |
| <b>arraySizeInSamps</b> | uInt32   | The size of the array, in samples, into which data is read.  |
| <b>reserved</b>         | bool32 * | Reserved for future use. Pass NULL to this parameter.  |

*Output*

| Name                    | Type        | Description  |
|-------------------------|-------------|--|
| <b>readArray</b>        | float64 [ ] | The array to read data into.                         |
| <b>sampsPerChanRead</b> | int32 *     | The actual number of samples read from each channel. |

**Return Value**

| Name          | Type  | Description   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |

**DAQmxBaseReadCounterScalarF64**

int32 DAQmxBaseReadCounterScalarF64 (TaskHandle taskHandle, float64 timeout, float64 \*value, bool32 \*reserved);

**Purpose**

Reads a single floating-point sample from a counter task. Use this function when the counter sample is scaled to a floating-point value, such as for frequency and period measurement.

**Parameters***Input*

| Name              | Type       | Description  |
|-------------------|------------|--|
| <b>taskHandle</b> | TaskHandle | The task to read the sample from.  |
| <b>timeout</b>    | float64    | The amount of time, in seconds, to wait for the function to read the sample(s). This function returns an error if the timeout elapses. |
| <b>reserved</b>   | bool32 *   | Reserved for future use. Pass NULL to this parameter.  |

*Output*

| Name         | Type      | Description                    |
|--------------|-----------|--------------------------------|
| <b>value</b> | float64 * | The sample read from the task. |

**Return Value**

| Name          | Type  | Description   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |

**DAQmxBaseReadCounterScalarU32**

int32 DAQmxBaseReadCounterScalarU32 (TaskHandle taskHandle, float64 timeout, uInt32 \*value, bool32 \*reserved);

**Purpose**

Reads a 32-bit integer sample from a counter task. Use this function when the counter sample is returned unscaled, such as for edge counting.

**Parameters***Input*

| Name              | Type       | Description                       |
|-------------------|------------|-----------------------------------|
| <b>taskHandle</b> | TaskHandle | The task to read the sample from. |

|                 |          |  |
|-----------------|----------|--|
| <b>timeout</b>  | float64  | The amount of time, in seconds, to wait for the function to read the sample(s). This function returns an error if the timeout elapses. |
| <b>reserved</b> | bool32 * | Reserved for future use. Pass NULL to this parameter.  |

*Output*

| <b>Name</b>  | <b>Type</b> | <b>Description</b>             |
|--------------|-------------|--------------------------------|
| <b>value</b> | uInt32 *    | The sample read from the task. |

**Return Value****Name Type Description**

|               |       |   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |
|---------------|-------|---|

**DAQmxBaseReadCounterU32**

int32 DAQmxBaseReadCounterU32 (TaskHandle taskHandle, int32 numSampsPerChan, float64 timeout, uInt32 readArray [ ], uInt32 arraySizeInSamps, int32 \*sampsPerChanRead, bool32 \*reserved);

**Purpose**

Reads multiple 32-bit integer samples from a counter task. Use this function when counter samples are returned unscaled, such as for edge counting.

**Parameters***Input*

| <b>Name</b>             | <b>Type</b> | <b>Description</b>  |
|-------------------------|-------------|---|
| <b>taskHandle</b>       | TaskHandle  | The task to read samples from.  |
| <b>numSampsPerChan</b>  | int32       | The number of samples, per channel, to read. The default value of -1 (DAQmx_Val_Auto) reads all available data. If <b>readArray</b> does not contain enough space, this function returns as much data as fits in <b>readArray</b> .<br><br>NI-DAQmx Base determines how many samples to read based on whether the task acquires samples continuously or acquires a finite number of samples.<br><br>If the task acquires samples continuously and you set this parameter to -1, this function reads all the samples currently available in the buffer.<br><br>If the task acquires a finite number of samples and you set this parameter to -1, the function waits for the task to acquire all requested samples, and then reads those samples. |
| <b>timeout</b>          | float64     | The amount of time, in seconds, to wait for the function to read the sample(s). This function returns an error if the timeout elapses.  |
| <b>arraySizeInSamps</b> | uInt32      | The size of the array, in samples, into which data is read.   |
| <b>reserved</b>         | bool32 *    | Reserved for future use. Pass NULL to this parameter.   |

*Output*

| <b>Name</b>             | <b>Type</b> | <b>Description</b>                                   |
|-------------------------|-------------|--|
| <b>readArray</b>        | uInt32 [ ]  | The array to read data into.                         |
| <b>sampsPerChanRead</b> | int32 *     | The actual number of samples read from each channel. |

**Return Value****Name Type Description**

|               |       |   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |
|---------------|-------|---|

**DAQmxBaseReadDigitalScalarU32**

int32 DAQmxBaseReadDigitalScalarU32 (TaskHandle taskHandle, float64 timeout, uInt32 \*value, bool32 \*reserved);

## Purpose

Reads a single 32-bit integer sample from a task that contains a single digital input channel. Use this return type for devices with up to 32 lines per port. The data is returned in unsigned integer format.

## Parameters

### Input

| Name              | Type       | Description  |
|-------------------|------------|--|
| <b>taskHandle</b> | TaskHandle | The task to read the sample from.  |
| <b>timeout</b>    | float64    | The amount of time, in seconds, to wait for the function to read the sample(s). This function returns an error if the timeout elapses. |
| <b>reserved</b>   | bool32 *   | Reserved for future use. Pass NULL to this parameter.  |

### Output

| Name         | Type     | Description                    |
|--------------|----------|--------------------------------|
| <b>value</b> | uInt32 * | The sample read from the task. |

## Return Value

| Name          | Type  | Description   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |

## DAQmxBaseReadDigitalU32

int32 DAQmxBaseReadDigitalU32 (TaskHandle taskHandle, int32 numSampsPerChan, float64 timeout, bool32 fillMode, uInt32 readArray[ ], uInt32 arraySizeInSamps, int32 \*sampsPerChanRead, bool32 \*reserved);

## Purpose

Reads multiple 32-bit integer samples from a task that contains one or more digital input channels. Use this return type for devices with up to 32 lines per port. The data is returned in unsigned integer format.

## Parameters

### Input

| Name                        | Type                               | Description   |       |             |                          |                                    |                             |                                    |
|-----------------------------|------------------------------------|---|-------|-------------|--------------------------|------------------------------------|-----------------------------|------------------------------------|
| <b>taskHandle</b>           | TaskHandle                         | The task to read samples from.  |       |             |                          |                                    |                             |                                    |
| <b>numSampsPerChan</b>      | int32                              | The number of samples, per channel, to read. The default value of -1 (DAQmx_Val_Auto) reads all available data. If <b>readArray</b> does not contain enough space, this function returns as much data as fits in <b>readArray</b> .<br>NI-DAQmx Base determines how many samples to read based on whether the task acquires samples continuously or acquires a finite number of samples.<br>If the task acquires samples continuously and you set this parameter to -1, this function reads all the samples currently available in the buffer.<br>If the task acquires a finite number of samples and you set this parameter to -1, the function waits for the task to acquire all requested samples, and then reads those samples. |       |             |                          |                                    |                             |                                    |
| <b>timeout</b>              | float64                            | The amount of time, in seconds, to wait for the function to read the sample(s). This function returns an error if the timeout elapses.  |       |             |                          |                                    |                             |                                    |
| <b>fillMode</b>             | bool32                             | Specifies whether or not the samples are interleaved.<br><table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>DAQmx_Val_GroupByChannel</td> <td>Group by channel (non-interleaved)</td> </tr> <tr> <td>DAQmx_Val_GroupByScanNumber</td> <td>Group by scan number (interleaved)</td> </tr> </tbody> </table>  | Value | Description | DAQmx_Val_GroupByChannel | Group by channel (non-interleaved) | DAQmx_Val_GroupByScanNumber | Group by scan number (interleaved) |
| Value                       | Description                        |   |       |             |                          |                                    |                             |                                    |
| DAQmx_Val_GroupByChannel    | Group by channel (non-interleaved) |   |       |             |                          |                                    |                             |                                    |
| DAQmx_Val_GroupByScanNumber | Group by scan number (interleaved) |   |       |             |                          |                                    |                             |                                    |
| <b>arraySizeInSamps</b>     | uInt32                             | The size of the array, in samples, into which data is read.   |       |             |                          |                                    |                             |                                    |



**reserved** bool32 \* Reserved for future use. Pass NULL to this parameter.

#### Output

| Name                    | Type       | Description   |
|-------------------------|------------|---|
| <b>readArray</b>        | uInt32 [ ] | The array to read data into, organized according to <b>fillMode</b> . |
| <b>sampsPerChanRead</b> | int32 *    | The actual number of samples read from each channel.                  |

## Return Value

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

|               |       |   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |
|---------------|-------|---|

## DAQmxBaseReadDigitalU8

int32 DAQmxBaseReadDigitalU8 (TaskHandle taskHandle, int32 numSampsPerChan, float64 timeout, bool32 fillMode, uInt8 readArray[ ], uInt32 arraySizeInSamps, int32 \*sampsPerChanRead, bool32 \*reserved);

## Purpose

Reads multiple 8-bit integer samples from a task that one or more multiple digital input channels. Use this function for devices with up to 8 lines per port. The data is returned in unsigned byte format.

## Parameters

#### Input

| Name                        | Type                               | Description   |       |             |                          |                                    |                             |                                    |
|-----------------------------|------------------------------------|---|-------|-------------|--------------------------|------------------------------------|-----------------------------|------------------------------------|
| <b>taskHandle</b>           | TaskHandle                         | The task to write samples to.   |       |             |                          |                                    |                             |                                    |
| <b>numSampsPerChan</b>      | int32                              | The number of samples, per channel, to read. The default value of -1 (DAQmx_Val_Auto) reads all available data. If <b>readArray</b> does not contain enough space, this function returns as much data as fits in <b>readArray</b> .<br><br>NI-DAQmx Base determines how many samples to read based on whether the task acquires samples continuously or acquires a finite number of samples.<br><br>If the task acquires samples continuously and you set this parameter to -1, this function reads all the samples currently available in the buffer.<br><br>If the task acquires a finite number of samples and you set this parameter to -1, the function waits for the task to acquire all requested samples, and then reads those samples. |       |             |                          |                                    |                             |                                    |
| <b>timeout</b>              | float64                            | The amount of time, in seconds, to wait for the function to read the sample(s). This function returns an error if the timeout elapses.  |       |             |                          |                                    |                             |                                    |
| <b>fillMode</b>             | bool32                             | Specifies whether or not the samples are interleaved.<br><br><table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>DAQmx_Val_GroupByChannel</td> <td>Group by channel (non-interleaved)</td> </tr> <tr> <td>DAQmx_Val_GroupByScanNumber</td> <td>Group by scan number (interleaved)</td> </tr> </tbody> </table>  | Value | Description | DAQmx_Val_GroupByChannel | Group by channel (non-interleaved) | DAQmx_Val_GroupByScanNumber | Group by scan number (interleaved) |
| Value                       | Description                        |   |       |             |                          |                                    |                             |                                    |
| DAQmx_Val_GroupByChannel    | Group by channel (non-interleaved) |   |       |             |                          |                                    |                             |                                    |
| DAQmx_Val_GroupByScanNumber | Group by scan number (interleaved) |   |       |             |                          |                                    |                             |                                    |
| <b>arraySizeInSamps</b>     | uInt32                             | The size of the array, in samples, into which data is read.   |       |             |                          |                                    |                             |                                    |
| <b>reserved</b>             | bool32 *                           | Reserved for future use. Pass NULL to this parameter.   |       |             |                          |                                    |                             |                                    |

#### Output

| Name                    | Type      | Description   |
|-------------------------|-----------|---|
| <b>readArray</b>        | uInt8 [ ] | The array into which data is read, organized according to <b>fillMode</b> . |
| <b>sampsPerChanRead</b> | int32 *   | The actual number of samples read from each channel.                        |

## Return Value

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

|               |       |  |
|---------------|-------|--|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. |
|---------------|-------|--|

A positive value indicates a warning. A negative value indicates an error.

## DAQmxBaseWriteAnalogF64

int32 DAQmxBaseWriteAnalogF64 (TaskHandle taskHandle, int32 numSampsPerChan, bool32 autoStart, float64 timeout, bool32 dataLayout, float64 writeArray[ ], int32 \*sampsPerChanWritten, bool32 \*reserved);

### Purpose

Writes multiple floating-point samples to a task that contains one or more analog output channels.



**Note** Buffered writes require a minimum buffer size of two samples.

### Parameters

#### Input

| Name                        | Type                               | Description   |       |             |                          |                                    |                             |                               |
|-----------------------------|------------------------------------|---|-------|-------------|--------------------------|------------------------------------|-----------------------------|-------------------------------|
| <b>taskHandle</b>           | TaskHandle                         | The task to write samples to.   |       |             |                          |                                    |                             |                               |
| <b>numSampsPerChan</b>      | int32                              | The number of samples, per channel, to write. You must pass in a value of 0 or more in order for the sample to write. If you pass a negative number, this function returns an error.  |       |             |                          |                                    |                             |                               |
| <b>autoStart</b>            | bool32                             | Always set to FALSE.  |       |             |                          |                                    |                             |                               |
| <b>timeout</b>              | float64                            | The amount of time, in seconds, to wait for this function to write all the samples. This function returns an error if the timeout elapses.  |       |             |                          |                                    |                             |                               |
| <b>dataLayout</b>           | bool32                             | Specifies how the samples are arranged, either interleaved or non-interleaved. <table border="1" data-bbox="568 934 1380 1060"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>DAQmx_Val_GroupByChannel</td> <td>Group by channel (non-interleaved)</td> </tr> <tr> <td>DAQmx_Val_GroupByScanNumber</td> <td>Group by sample (interleaved)</td> </tr> </tbody> </table> | Value | Description | DAQmx_Val_GroupByChannel | Group by channel (non-interleaved) | DAQmx_Val_GroupByScanNumber | Group by sample (interleaved) |
| Value                       | Description                        |   |       |             |                          |                                    |                             |                               |
| DAQmx_Val_GroupByChannel    | Group by channel (non-interleaved) |   |       |             |                          |                                    |                             |                               |
| DAQmx_Val_GroupByScanNumber | Group by sample (interleaved)      |   |       |             |                          |                                    |                             |                               |
| <b>writeArray</b>           | float64[ ]                         | The array of 64-bit samples to write to the task.   |       |             |                          |                                    |                             |                               |
| <b>reserved</b>             | bool32 *                           | Reserved for future use. Pass NULL to this parameter.   |       |             |                          |                                    |                             |                               |

#### Output

| Name                       | Type    | Description  |
|----------------------------|---------|--|
| <b>sampsPerChanWritten</b> | int32 * | The actual number of samples per channel successfully written to the buffer. |

### Return Value

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

|               |       |   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |
|---------------|-------|---|

## DAQmxBaseWriteDigitalU8

int32 DAQmxBaseWriteDigitalU8 (TaskHandle taskHandle, int32 numSampsPerChan, bool32 autoStart, float64 timeout, bool32 dataLayout, uInt8 writeArray[ ], int32 \*sampsPerChanWritten, bool32 \*reserved);

### Purpose

Writes multiple eight-bit unsigned integer samples to a task that contains one or more digital output channels. Use this format for devices with up to 8 lines per port.



**Note** Buffered writes require a minimum buffer size of two samples.

### Parameters

#### Input

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

|                        |                             |  |
|------------------------|-----------------------------|--|
| <b>taskHandle</b>      | TaskHandle                  | The task to write samples to.  |
| <b>numSampsPerChan</b> | int32                       | The number of samples, per channel, to write. You must pass in a value of 0 or more in order for the sample to write. If you pass a negative number, this function returns an error. |
| <b>autoStart</b>       | bool32                      | Specifies whether or not this function automatically starts the task if you do not start it. This function is used only for static digital tasks; otherwise set autoStart to FALSE.  |
| <b>timeout</b>         | float64                     | The amount of time, in seconds, to wait for this function to write all the samples. This function returns an error if the timeout elapses.   |
| <b>dataLayout</b>      | bool32                      | Specifies how the samples are arranged, either interleaved or non-interleaved.   |
|                        | <b>Value</b>                | <b>Description</b>   |
|                        | DAQmx_Val_GroupByChannel    | Group by channel (non-interleaved)   |
|                        | DAQmx_Val_GroupByScanNumber | Group by sample (interleaved)  |
| <b>writeArray</b>      | uInt8 [ ]                   | The array of 8-bit integer samples to write to the task.   |
| <b>reserved</b>        | bool32 *                    | Reserved for future use. Pass NULL to this parameter.  |

*Output*

| <b>Name</b>                | <b>Type</b> | <b>Description</b>   |
|----------------------------|-------------|--|
| <b>sampsPerChanWritten</b> | int32 *     | The actual number of samples per channel successfully written to the buffer. |

**Return Value**

| <b>Name</b>   | <b>Type</b> | <b>Description</b>  |
|---------------|-------------|---|
| <b>status</b> | int32       | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |

**DAQmxBaseWriteDigitalU32**

int32 DAQmxBaseWriteDigitalU32 (TaskHandle taskHandle, int32 numSampsPerChan, bool32 autoStart, float64 timeout, bool32 dataLayout, uInt32 writeArray[ ], int32 \*sampsPerChanWritten, bool32 \*reserved);

**Purpose**

Writes multiple 32-bit unsigned integer samples to a task that contains one or more digital output channels. Use this format for devices with up to 32 lines per port.



**Note** Buffered writes require a minimum buffer size of 2 samples.

**Parameters***Input*

| <b>Name</b>            | <b>Type</b>                 | <b>Description</b>   |
|------------------------|-----------------------------|--|
| <b>taskHandle</b>      | TaskHandle                  | The task to write samples to.  |
| <b>numSampsPerChan</b> | int32                       | The number of samples, per channel, to write. You must pass in a value of 0 or more in order for the sample to write. If you pass a negative number, this function returns an error. |
| <b>autoStart</b>       | bool32                      | Specifies whether or not this function automatically starts the task if you do not start it. This is used only for static digital tasks, otherwise set autoStart to False.           |
| <b>timeout</b>         | float64                     | The amount of time, in seconds, to wait for this function to write all the samples. This function returns an error if the timeout elapses.   |
| <b>dataLayout</b>      | bool32                      | Specifies how the samples are arranged, either interleaved or non-interleaved.   |
|                        | <b>Value</b>                | <b>Description</b>   |
|                        | DAQmx_Val_GroupByChannel    | Group by channel (non-interleaved)   |
|                        | DAQmx_Val_GroupByScanNumber | Group by sample (interleaved)  |

**writeArray**                    uInt32 [ ]    The array of 32-bit integer samples to write to the task.  
**reserved**                    bool32 \*      Reserved for future use. Pass NULL to this parameter.

*Output*

| Name                       | Type    | Description  |
|----------------------------|---------|--|
| <b>sampsPerChanWritten</b> | int32 * | The actual number of samples per channel successfully written to the buffer. |

**Return Value**

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

|               |       |   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |
|---------------|-------|---|

**DAQmxBaseWriteDigitalScalarU32**

```
int32 DAQmxBaseWriteDigitalScalarU32 (TaskHandle taskHandle, bool32 autoStart, float64 timeout, uInt32 value, bool32 *reserved);
```

**Purpose**

Writes a single 32-bit unsigned integer sample to a task that contains a single digital output channel. Use this format for devices with up to 32 lines per port. Useful for static digital tasks only.

**Parameters***Input*

| Name              | Type       | Description  |
|-------------------|------------|--|
| <b>taskHandle</b> | TaskHandle | The task to write the sample to.   |
| <b>autoStart</b>  | bool32     | Specifies whether or not this function automatically starts the task if you do not start it. This is used only for static digital tasks; otherwise set autoStart to FALSE. |
| <b>timeout</b>    | float64    | The amount of time, in seconds, to wait for this function to write the <b>value</b> . This function returns an error if the timeout elapses.                               |
| <b>value</b>      | uInt32     | A 32-bit integer sample to write to the task.  |
| <b>reserved</b>   | bool32 *   | Reserved for future use. Pass NULL to this parameter.  |

**Return Value**

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

|               |       |   |
|---------------|-------|---|
| <b>status</b> | int32 | The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error. |
|---------------|-------|---|

**DAQmxBaseExportSignal**

```
int32 DAQmxBaseExportSignal (TaskHandle taskHandle, int32 signalID, const char outputTerminal[]);
```

**Purpose**

Routes a control signal to the specified terminal. The output terminal can reside on the device that generates the control signal or on a different device. Use this function to share clocks and triggers between multiple tasks and devices. The routes created by this function are task-based routes.

**Parameters***Input*

| Name              | Type                     | Description   |
|-------------------|--------------------------|---|
| <b>taskHandle</b> | TaskHandle               | The task used in this function.                     |
| <b>signalID</b>   | int32                    | The name of the trigger, clock, or event to export. |
|                   | <b>Value</b>             | <b>Description</b>                                  |
|                   | DAQmx_Val_AIConvertClock | Clock that causes an analog-to-digital              |

|                                |  |
|--------------------------------|--|
|                                | conversion on an E Series device. One conversion corresponds to a single sample from one channel.  |
| DAQmx_Val_20MHzTimebaseClock   | Output of an oscillator that is the onboard source of the Master Timebase. Other timebases are derived from this clock.  |
| DAQmx_Val_SampleClock          | Clock the device uses to time each sample.   |
| DAQmx_Val_AdvanceTrigger       | Trigger that moves a switch to the next entry in a scan list.  |
| DAQmx_Val_ReferenceTrigger     | Trigger that establishes the reference point between pretrigger and posttrigger samples.   |
| DAQmx_Val_StartTrigger         | Trigger that begins a measurement or generation.   |
| DAQmx_Val_AdvCmpltEvent        | Signal that a switch product generates after it both executes the command(s) in a scan list entry and waits for the settling time to elapse.   |
| DAQmx_Val_AIHoldCmpltEvent     | Signal that an E Series device generates when the device latches analog input data (the ADC enters "hold" mode) and it is safe for any external switching hardware to remove the signal and replace it with the next signal. This event does not indicate the completion of the actual analog-to-digital conversion. |
| DAQmx_Val_CounterOutputEvent   | Signal that a counter generates. Each time the counter reaches terminal count, this signal toggles or pulses.  |
| DAQmx_Val_ChangeDetectionEvent | Signal that a static DIO device generates when the device detects a rising or falling edge on any of the lines or ports you selected when you configured change detection timing.  |
| DAQmx_Val_WDTEexpiredEvent     | Signal that a static DIO device generates when the watchdog timer expires.   |

**outputTerminal** const char [ ] The destination terminal of the exported signal.

## Return Value

**Name** **Type** **Description**

**status** int32 The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error.

## DAQmxBaseCfgInputBuffer

int32 DAQmxBaseCfgInputBuffer (TaskHandle taskHandle, uInt32 numSampsPerChan);

## Purpose

Overrides the automatic input buffer allocation that NI-DAQmx Base performs.

## Parameters

*Input*

| Name                   | Type       | Description  |
|------------------------|------------|--|
| <b>taskHandle</b>      | TaskHandle | The task used in this function.  |
| <b>numSampsPerChan</b> | uInt32     | The number of samples the buffer can hold for each channel in the task. Zero indicates no buffer should be allocated. Use a buffer size of 0 to perform a hardware-timed operation without using a buffer. |

## Return Value

### Name Type Description

**status** int32 The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A positive value indicates a warning. A negative value indicates an error.

## DAQmxBaseGetExtendedErrorInfo

int32 DAQmxBaseGetExtendedErrorInfo (char errorString[ ], uInt32 bufferSize);

## Purpose

Returns dynamic, specific error information. This function is valid only for the last function that failed; additional NI-DAQmxBase calls may invalidate this information.

If you pass valid values for **errorString** and **bufferSize**, this function returns as much of the available data as possible.

If you pass NULL for **errorString** or 0 for **bufferSize**, this function returns the number of bytes you need to allocate.

## Parameters

### Input

#### Name Type Description

**bufferSize** uInt32 The size, in bytes, of **errorString**. If you pass 0, this function returns the number of bytes you need to allocate.

### Output

#### Name Type Description

**errorString** char [ ] Dynamic error information. If you pass NULL, this function returns the number of bytes you need to allocate.

## Return Value

### Name Type Description

**status** int32 The error code returned by the function in the event of an error or warning. A value of 0 indicates success. A negative value indicates an error.

If you pass in a valid value for **errorString** and its **bufferSize**, this function returns as much of the available data as possible.

If you pass NULL for **errorString** or 0 for **bufferSize**, this function returns the number of bytes you need to allocate.

## Glossary

|          |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|----------|-----------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Prefixes | Numbers/Symbols | A | B | C | D | E | F | G | H | I | L | M | N | O | P | R | S | T | U | V | W |
|----------|-----------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| Prefix | Meaning | Value            |
|--------|---------|------------------|
| n-     | nano-   | 10 <sup>-9</sup> |
| μ-     | micro-  | 10 <sup>-6</sup> |
| m-     | milli-  | 10 <sup>-3</sup> |
| k-     | kilo-   | 10 <sup>3</sup>  |
| M-     | mega-   | 10 <sup>6</sup>  |

| Symbols | Meaning               |
|---------|-----------------------|
| %       | percent               |
| +       | positive of, or plus  |
| -       | negative of, or minus |
| Ω       | ohm                   |
| °       | degree                |

**A**

|               |   |
|---------------|---|
| accelerometer | A sensor that represents acceleration as a voltage.   |
| ADC           | Analog-to-digital converter—an electronic device, often an integrated circuit, that converts an analog signal to a digital value.   |
| ADE           | Application development environment—some examples include LabVIEW and LabWindows/CVI.   |
| AI            | Analog input—acquisition of data.   |
| amplification | A type of signal conditioning that improves accuracy in the resulting digitized signal by increasing signal amplitude relative to noise.  |
| analog        | Data represented by continuously variable physical quantities.  |
| AO            | Analog output—generation of data.   |
| API           | Application programming interface—A library of functions, classes or VIs, attributes, and properties for creating applications for your device.   |
| asynchronous  | <ol style="list-style-type: none"> <li>1. Hardware—a signal that occurs or is acted upon at an arbitrary time, without synchronization to another signal, such as a reference clock.</li> <li>2. Software—a VI or function that begins an operation and returns prior to the completion or termination of the operation.</li> </ol> |
| attenuation   | The reduction of a voltage or acoustical pressure. Measured referenced to the original voltage.   |

**B**

|           |  |
|-----------|--|
| bandwidth | The range of frequencies present in a signal, or the range of frequencies to which a measuring device can respond. |
| bipolar   | A signal range that includes both positive and negative values (for example, 0 V to +5 V).                         |
| bit       | The smallest unit of data used in a digital operation. Bits are binary, so they can be either a 1 or a 0.          |
| buffer    | In software, temporary storage for acquired or to-be-generated samples.  |

**C**

|                            |   |
|----------------------------|---|
| CH                         | Channel.  |
| channel                    | <ol style="list-style-type: none"> <li>1. Physical—a terminal or pin at which you can measure or generate an analog or digital signal. A single physical channel can include more than one terminal, as in the case of a differential analog input channel or a digital port of eight lines. The name used for a counter physical channel is an exception because that physical channel name is not the name of the terminal where the counter measures or generates the digital signal.</li> <li>2. Virtual—a collection of property settings that can include a name, a physical channel, input terminal connections, the type of measurement or generation, and scaling information. You can define NI-DAQmx virtual channels outside a task (global) or inside a task (local). Configuring virtual channels is optional in Traditional NI-DAQ and earlier versions, but is integral to every measurement you take in NI-DAQmx. In Traditional NI-DAQ, you configure virtual channels in MAX. In NI-DAQmx, you can configure virtual channels either in MAX or in a program, and you can configure channels as part of a task or separately.</li> <li>3. Switch—a switch channel represents any connection point on a switch. It may be made up of one or more signal wires (commonly one, two, or four), depending on the switch topology. A virtual channel cannot be created with a switch channel. Switch channels may be used only in the NI-DAQmx Switch functions and VIs.</li> </ol> |
| clock                      | A periodic digital signal.  |
| CMRR                       | Common-mode rejection ratio—a measure of the ability of an instrument to reject interference from a common-mode signal, usually expressed in decibels. (dB)   |
| code width                 | The smallest detectable change in an input voltage of a DAQ device.   |
| cold-junction compensation | A method of compensating for inaccuracies in thermocouple circuits.   |
| counter/timer              | A circuit that counts digital edges. Counters and timers usually have from 16 bits to 48 bits (sometimes more) counting capability. The total number of counts possible equals $2^N$ , where N is   |

the number of bits in the counter. When the edges counted are produced by a clock, elapsed time can be computed from the number of edges counted if the clock frequency is known.

custom scale A method of instructing NI-DAQmx to apply additional scaling to your data. Refer to the Create Scale function/VI in your reference help.

## D

DAC Digital-to-analog converter—an electronic device, often an integrated circuit, that converts a digital number into a corresponding analog voltage or current.

DAQ Refer to *data acquisition*.

DAQ Assistant A graphical interface for configuring measurement tasks, channels, and scales.

DAQ device A device that acquires or generates data and can contain multiple channels and conversion devices. DAQ devices include plug-in devices, PCMCIA cards, and DAQPad devices, which connect to a computer USB or 1394 (FireWire) port. SCXI modules are considered DAQ devices.

data Samples.

data acquisition

1. Acquiring and measuring analog or digital electrical signals from sensors, acquisition transducers, and test probes or fixtures.
2. Generating analog or digital electrical signals.

dB Decibel—the unit for expressing a logarithmic measure of the ratio of two signal levels:  $\text{dB} = 20 \log_{10} V_1/V_2$ , for signals in volts.

DC Direct current.

delay from sample The amount of time to wait after receiving a sample clock edge before beginning the acquisition of a sample.

delay from start The amount of time to wait after receiving a start trigger before beginning the operation.

device

1. An instrument or controller you can access as a single entity that controls or monitors real-world I/O points. A device often is connected to a host computer through some type of communication network.
2. See also [DAQ device](#) and [measurement device](#).

digital A TTL signal. Refer to [edge](#).

DIO digital input/output.

DMA Direct Memory Access—A method of transferring data between a buffer and a device that is used most often for high-speed operations.

driver Software unique to the device or type of device, and includes the set of commands the device accepts.

## E

E Series A standard architecture for instrumentation-class, multichannel data acquisition devices.

edge A digital edge is a single rising or falling TTL transition. An analog edge is defined by the slope, level, and hysteresis settings.

event A digital signal produced from a device or circuit.

excitation Supplying a voltage or current source to energize an active sensor or circuit.

## F

fall time The time for a signal to transition from 90% to 10% of the maximum signal amplitude.

FIFO A type of memory that implements a First In First Out strategy in which samples are removed in the order they were written. FIFOs are typically used as intermediate buffers between an ADC or DAC and the memory buffer.

filtering A type of signal conditioning that you can use to remove unwanted frequency components from the signal you are measuring.

floating signal sources Signal sources with voltage signals that are not connected to an absolute reference or system ground.



**G**

|                         |  |
|-------------------------|--|
| gain                    | The factor by which a signal is amplified, often expressed in dB. Gain as a function of frequency is commonly referred to as the magnitude of the frequency response function.         |
| grounded signal sources | Signal sources with voltage signals that are referenced to a system ground, such as the earth or a building ground. Grounded signal sources are also called referenced signal sources. |

**H**

|                     |   |
|---------------------|---|
| hardware triggering | A form of triggering in which the source of the trigger is an analog or digital signal. Refer to <a href="#">Software Trigger</a> . |
| hysteresis          | A window around a trigger level that is often used to reduce false triggering due to noise or jitter in the signal.                 |
| Hz                  | Hertz—cycles per second of a periodic signal.   |

**I**

|                   |  |
|-------------------|--|
| instrument driver | Refer to <a href="#">driver</a> .  |
| interrupts        | A method whereby a device notifies the computer of some condition on the device that requires the computer's attention. When this condition is a request for data or a notification of available data, interrupts are used as a data transfer mechanism. |
| I/O               | Input/output—the transfer of data to/from a computer system involving communications channels, operator interface devices, and/or data acquisition and control interfaces.   |
| isolation         | A type of signal conditioning in which you isolate the transducer signals from the computer. Isolation makes sure the measurements from the measurement device are not affected by differences in ground potentials.                                     |

**L**

|               |  |
|---------------|--|
| LED           | Light-emitting diode—a semiconductor light source.   |
| line          | An individual signal in a digital port. The difference between a bit and a line is that the bit refers to the actual data transferred, and the line refers to the hardware the bit is transferred on. However, the terms line and bit are fairly interchangeable. For example, an 8-bit port is the same as a port with eight lines. |
| linearization | A type of signal conditioning in which software linearizes the voltage levels from transducers, so the voltages can be scaled to measure physical phenomena.   |
| LSB           | Least significant bit—often used to refer to the smallest voltage change detectable by an A/D converter or the smallest voltage change that can be generated by a D/A converter.   |
| LVDT          | Linear variable differential transformer—a sensor that measures linear displacement.   |

**M**

|   |   |
|---|---|
| M Series                                | An architecture for instrumentation-class, multichannel data acquisition devices based on the earlier E Series architecture with added new features.  |
| Measurement & Automation Explorer (MAX) | A centralized configuration environment that allows you to configure all of your National Instruments devices.  |
| measurement device                      | DAQ devices such as the E Series multifunction I/O (MIO) devices, SCXI signal conditioning modules, and switch modules.   |
| memory buffer                           | Refer to <a href="#">buffer</a> .   |
| memory mapping                          | A technique for reading and writing to a device directly from your program, which avoids the overhead of delegating the reads and writes to kernel-level software. Delegation to the kernel is safer, but slower. Memory mapping is less safe because an entire 4 KB page of memory must be exposed to your program for this to work, but it is faster. |
| MIO                                     | Multifunction I/O—Designates a family of data acquisition devices that have multiple analog input channels, digital I/O channels, timing, and optionally, analog output channels. An MIO product can be considered a miniature mixed signal tester, due to its broad range of signal types and  |

flexibility. It is also known as multifunction DAQ. An E Series device is an example of an MIO device.

- module A board assembly and its associated mechanical parts, front panel, optional shields, and so on. A module contains everything required to occupy one or more slots in a mainframe. SCXI and PXI devices are modules.
- multiplexer A switching device with multiple terminals that sequentially connects each of its terminals to a single terminal, typically at high speeds. Often used to measure several signals with a single analog input channel.

## N

- NI-DAQ Driver software included with all NI measurement devices. NI-DAQ is an extensive library of VIs and functions you can call from an application development environment (ADE), such as LabVIEW, to program all the features of an NI measurement device, such as configuring, acquiring and generating data from, and sending data to the device.
- NI-DAQmx The latest NI-DAQ driver with new VIs, functions, and development tools for controlling measurement devices. The advantages of NI-DAQmx over earlier versions of NI-DAQ include the DAQ Assistant for configuring channels and measurement tasks for your device for use in LabVIEW, LabWindows/CVI, and Measurement Studio; increased performance such as faster single-point analog I/O; and a simpler API for creating DAQ applications using fewer functions and VIs than earlier versions of NI-DAQ.
- NI-DAQmx Base NI-DAQmx Base is a NI-DAQ driver with the following features: provides a high-level NI-DAQmx interface on LabVIEW PDA for Pocket PC 2003, Linux, Mac OS X, and certain USB devices on Windows; is a subset of the NI-DAQmx API: if you are familiar with NI-DAQmx, you should be able to comfortably use NI-DAQmx Base; is comprised of LabVIEW VIs, which allows you to customize the driver, if needed.
- nonlinearity A measure in percentage of full-scale range (FSR) of the worst-case deviation from the ideal transfer function—a straight line.
- This specification is included only for DAQ products, such as signal conditioning products, that do not have an ADC. Because a product with this specification can also be used with a DAQ product with an ADC, this nonlinearity specification must be added to the relative accuracy specification of the DAQ product with the ADC.
- NRSE Nonreferenced single-ended mode—all measurements are made with respect to a common (NRSE) measurement system reference, but the voltage at this reference can vary with respect to the measurement system ground.

## O

- onboard Provided by the data acquisition device.
- onboard channels Channels provided by the plug-in data acquisition device.
- onboard clock The default source for a particular clock. Usually, the device has dedicated a circuit for producing this signal and its only purpose is to act as the source for a certain clock.
- onboard memory Memory provided by a device for temporary storage of input or output data. Typically, onboard memory is a FIFO, which is distinct from computer memory.

## P

- parallel mode A type of SCXI operating mode in which the module sends each of its input channels directly to a separate analog input channel of the device connected to the module.
- pattern I/O Pattern input and output—a digital I/O operation on which a clock signal initiates a digital transfer. Because the clock signal is a constant frequency, you can generate and receive patterns at a constant rate.
- PCI Peripheral Component Interconnect—a high-performance expansion bus architecture originally developed by Intel to replace ISA and EISA. PCI has achieved widespread acceptance as a standard for PCs and work stations, and it offers a theoretical maximum transfer rate of 132 Mbytes/s.
- PFI Programmable Function Interface—general purpose input terminals, fixed purpose output terminals. The name of the fixed output signal is often placed on the I/O connector next to the terminal as a hint.

|                     |   |
|---------------------|---|
| physical channel    | Refer to <a href="#">channel</a> .  |
| pin                 | Refer to <a href="#">terminal</a> .   |
| Poisson's Ratio     | The negative ratio of the strain in the transverse direction (perpendicular to the force) to the strain in the axial direction (parallel to the force).   |
| port                | A collection of digital lines. Usually the lines are grouped into either a 8-bit or 32-bit port. Most E Series devices have one 8-bit port.   |
| port width          | The number of lines in a port. For example, most E Series devices have one port with eight lines; therefore, the port width is eight.   |
| posttrigger samples | If there is no reference trigger, posttrigger samples are the data acquired after the task is started. If there is a reference trigger, this is the data acquired after the reference trigger.  |
| pretrigger samples  | Data acquired before the occurrence of the reference trigger.   |
| pretriggering       | The technique used on a measurement device to keep a circular buffer filled with samples, so that when the reference trigger conditions are met, the buffer includes samples leading up to the trigger condition as well as samples acquired immediately after the trigger.                     |
| programmed I/O      | a data transfer mechanism in which a buffer is not used and instead, the computer reads and writes directly to the device.  |
| propagation delay   | The amount of time required for a signal to pass through a circuit.   |
| pulsed output       | A form of counter signal generation by which a pulse is generated when a counter reaches a certain value.   |
| PXI                 | PCI eXtensions for Instrumentation—a rugged, open system for modular instrumentation based on CompactPCI, with special mechanical, electrical, and software features. The PXI standard was originally developed by National Instruments in 1997 and is now managed by the PXI Systems Alliance. |
| PXI trigger bus     | The timing bus that connects PXI DAQ devices directly, by means of connectors built into the backplane of the PXI chassis, for precise synchronization of functions. This bus is functionally equivalent to the RTSI bus for PCI DAQ devices.   |

## R

|                          |  |
|--------------------------|--|
| range                    | The minimum and maximum analog signal levels that the ADC can digitize.  |
| raw                      | Data that has not been changed in any way. For input, data is returned exactly as received from the device. For output, data is written as is to the device. Refer to <a href="#">unscaled</a> and <a href="#">scaled</a> .                                    |
| referenced signal source | Signal sources with voltage signals that are referenced to a system ground, such as the earth or a building ground. Also called grounded signal sources.   |
| resolution               | The smallest amount of input signal change that a device or sensor can detect. The term <i>discrimination</i> is also used for resolution.   |
| rise time                | The time for a signal to transition from 10% to 90% of the maximum signal amplitude.   |
| route                    | A connection between a pair of terminals. Any time the source or destination terminal of a signal is specified, a route is created.  |
| RSE                      | Referenced single-ended mode—all measurements are made with respect to a common reference measurement system or a ground. Also called a grounded measurement system.   |
| RTD                      | Resistance temperature detector—a metallic probe that measures temperature based on its coefficient of resistivity.  |
| RTSI bus                 | Real-time system integration bus—the NI timing bus that connects DAQ devices directly, by means of connectors on top of the devices, for precise synchronization of functions. This bus is functionally equivalent to the PXI Trigger bus for PXI DAQ devices. |
| RVDT                     | Rotary variable differential transformer—a sensor whose output signal represents the rotation of the shaft.  |

## S

|     |  |
|-----|--|
| s   | Seconds.   |
| S   | Samples. Refer to Sample.  |
| S/s | Samples per second—used to express the rate at which a measurement device samples an |

|                       |  |
|-----------------------|--|
|                       | analog signal.   |
| sample                | A sample is a single measurement from a single channel or, for output, a single generation to a single channel. A device may produce more than one sample per channel upon receiving a single digital edge of a sample clock. An E Series device, for example, produces one sample from each analog input channel in its task for every sample clock edge. |
| sample clock          | The clock controlling the time interval between samples. Each time the Sample Clock ticks (produces a pulse) one sample per channel is acquired or generated.  |
| sample clock rate     | The number of samples per channel per second. For example, a sample clock rate of 10 S/s means sampling each channel 10 times per second.  |
| scaled                | Data that has been mathematically transformed into engineering units. Other manipulations also can be done such as reordering to match the channel order.  |
| scanning              | Method of sequentially connecting channels.  |
| SCXI                  | Signal Conditioning eXtensions for Instrumentation—the NI product line for conditioning low-level signals within an external chassis near sensors so that only high-level signals are sent to measurement devices in the noisy PC environment. SCXI is an open standard available for all vendors.   |
| sensor                | A device that responds to a physical stimulus (heat, light, sound, pressure, motion, flow, and so on) and produces a corresponding electrical signal.  |
| signal                | A means of conveying information. An analog waveform, a clock, and a single digital (TTL) edge are all examples of signals.  |
| signal conditioning   | The manipulation of signals to prepare them for digitizing.  |
| software trigger      | A VI or function that, when it executes, triggers an action such as starting an acquisition.   |
| source impedance      | A parameter of signal sources that reflects current-driving ability of voltage sources (lower is better) and the voltage-driving ability of current sources (higher is better).  |
| STC                   | System timing controller.  |
| synchronous           | <ol style="list-style-type: none"> <li>1. Hardware—a signal that occurs or is acted upon in synchrony with another signal, such as a reference clock.</li> <li>2. Software—a VI or function that begins an operation and returns only when the operation is complete.</li> </ol>   |
| <b>T</b>              |  |
| task                  | A collection of one or more channels, timing, and triggering and other properties that apply to the task itself. Conceptually, a task represents a measurement or generation you want to perform.  |
| task buffer           | Refer to <a href="#">buffer</a> .  |
| terminal              | A named location on a DAQ device where a signal is either generated (output or produced) or acquired (input or consumed).  |
| terminal count        | When counting up, an $N$ bit counter reaches its terminal count at $2^N - 1$ . An $N$ bit counter counting down reaches its terminal count at 0.   |
| thermistor            | A semiconductor sensor that produces a repeatable change in electrical resistance as a function of temperature. Most thermistors have a negative temperature coefficient.  |
| thermocouple          | A temperature sensor created by joining two dissimilar metals. The junction produces a small voltage as a function of the temperature.   |
| threshold             | The voltage level a signal must reach for a trigger to occur.  |
| tick                  | A digital edge of a clock.   |
| timebase              | A clock that is divided down to produce another clock or a clock provided to a counter for measuring elapsed time.   |
| Traditional NI-DAQ    | An upgrade to the earlier version of NI-DAQ. Traditional NI-DAQ has the same VIs and functions and works the same way as NI-DAQ 6.9.x. You can use both Traditional NI-DAQ and NI-DAQmx on the same computer, which is not possible with NI-DAQ 6.9.x.   |
| transducer            | Refer to <a href="#">sensor</a> .  |
| transducer excitation | A type of signal conditioning that uses external voltages and currents to excite the circuitry of a  |

|         |  |
|---------|--|
|         | signal conditioning system into measuring physical phenomena.                            |
| trigger | Any signal that causes a device to perform an action, such as starting an acquisition.   |
| TTL     | Transistor-transistor logic—a signal having two discrete levels, a high and a low level. |

## U

|          |   |
|----------|---|
| unipolar | A signal range that is always positive (for example, 0 to +10 V).   |
| unscaled | Samples in the integer form that the hardware produces or requires. Although no mathematical transformations are applied to unscaled data, other manipulations may be done such as reordering to match the channel order. |

## V

|                    |  |
|--------------------|--|
| V                  | Volts.   |
| VI                 | Virtual instrument. Refer to <i>virtual instrument</i> .                               |
| virtual channel    | Refer to <a href="#">channel</a> .   |
| virtual instrument | A program in LabVIEW that models the appearance and function of a physical instrument. |

## W

|                    |  |
|--------------------|--|
| waveform data type | A LabVIEW data type that bundles timing information along with the data. |
| WDT                | Refer to <a href="#">waveform data type</a> .                            |

## Important Information

[Warranty](#)

[Copyright](#)

[Trademarks](#)

[Patents](#)

[Warning Regarding Use of NI Products](#)

[Environmental Management](#)

## Warranty

The HARDWARE PRODUCT NAME is warranted against defects in materials and workmanship for a period of one year from the date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace equipment that proves to be defective during the warranty period. This warranty includes parts and labor.

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THEREOF PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover

damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

## Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

National Instruments respects the intellectual property of others, and we ask our users to do the same. NI software is protected by copyright and other intellectual property laws. Where NI software may be used to reproduce software or other materials belonging to others, you may use NI software only to reproduce materials that you may reproduce in accordance with the terms of any applicable license or other legal restriction.

## Trademarks

National Instruments, NI, ni.com, and LabVIEW are trademarks of National Instruments Corporation. Refer to the *Terms of Use* section on ni.com/legal for more information about [National Instruments trademarks](#).

FireWire® is the registered trademark of Apple Computer, Inc.

Handle Graphics®, MATLAB®, Real-Time Workshop®, Simulink®, Stateflow®, and xPC TargetBox® are registered trademarks, and TargetBox™ and Target Language Compiler™ are trademarks of The MathWorks, Inc.

Tektronix® and Tek are registered trademarks of Tektronix, Inc.

The Bluetooth® word mark is a registered trademark owned by the Bluetooth SIG, Inc.

Other product and company names mentioned herein are trademarks or trade names of their respective companies.

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

## Patents

For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the patents.txt file on your media, or [ni.com/patents](http://ni.com/patents).

## WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

## Environmental Management

National Instruments is committed to designing and manufacturing products in an environmentally responsible manner. NI recognizes that eliminating certain hazardous substances from our products is beneficial not only to the environment but also to NI customers.

For additional environmental information, refer to the [NI and the Environment](#) Web page at ni.com/environment. This page contains the environmental regulations and directives with which NI complies, as well as other environmental information not included in this document.

## Waste Electrical and Electronic Equipment (WEEE)



**EU Customers** At the end of their life cycle, all products *must* be sent to a WEEE recycling center. For more information about WEEE recycling centers and National Instruments WEEE initiatives, visit [ni.com/environment/weee.htm](http://ni.com/environment/weee.htm).

## 电子信息产品污染控制管理办法（中国 RoHS）



**中国客户** National Instruments符合中国电子信息产品中限制使用某些有害物质指令（RoHS）。关于National Instruments中国RoHS合规性信息，请登录 [ni.com/environment/rohs\\_china](http://ni.com/environment/rohs_china)。（For information about China RoHS compliance, go to [ni.com/environment/rohs\\_china](http://ni.com/environment/rohs_china)。）

## Technical Support and Professional Services

Visit the following sections of the award-winning National Instruments Web site at [ni.com](http://ni.com) for technical support and professional services:

- [Support](#)—Technical support resources at [ni.com/support](http://ni.com/support) include the following:
  - **Self-Help Resources**—For answers and solutions, visit [ni.com/support](http://ni.com/support) for software drivers and updates, a searchable [KnowledgeBase](#), [product manuals](#), step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on. Registered users also receive access to the [NI Discussion Forums](#) at [ni.com/forums](http://ni.com/forums). NI Applications Engineers make sure every question submitted online receives an answer.
  - **Standard Service Program Membership**—This program entitles members to direct access to NI Applications Engineers via phone and email for one-to-one technical support, as well as exclusive access to on demand training modules via the [Services Resource Center](#). NI offers complementary membership for a full year after purchase, after which you may renew to continue your benefits.  
  
For information about other [technical support options](#) in your area, visit [ni.com/services](http://ni.com/services) or [contact](#) your local office at [ni.com/contact](http://ni.com/contact).
- [Training and Certification](#)—Visit [ni.com/training](http://ni.com/training) for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- [System Integration](#)—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit [ni.com/alliance](http://ni.com/alliance).
- [Declaration of Conformity \(DoC\)](#)—A DoC is our claim of compliance with the Council of the European Communities using the manufacturer's declaration of conformity. This system affords the user protection for electromagnetic compatibility (EMC) and product safety. You can obtain the DoC for your product by visiting [ni.com/certification](http://ni.com/certification).
- [Calibration Certificate](#)—If your product supports calibration, you can obtain the calibration certificate for your product at [ni.com/calibration](http://ni.com/calibration).

If you searched [ni.com](http://ni.com) and could not find the answers you need, contact your [local office](#) or NI corporate headquarters. You also can visit the [Worldwide Offices](#) section of [ni.com/niglobal](http://ni.com/niglobal) to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.