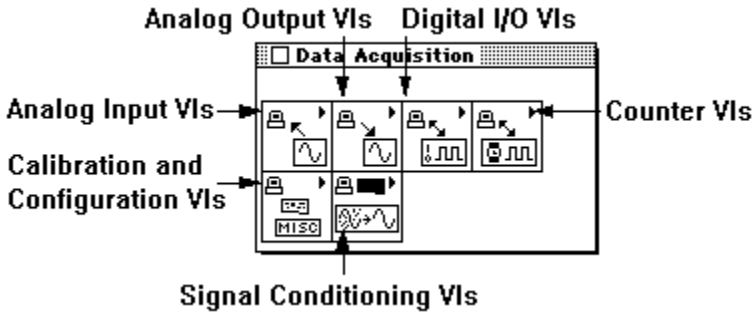
Data Acquisition VIs

This file provides online help for the data acquisition (DAQ) VIs in LabVIEW. The DAQ VIs are a collection of VIs that work with your DAQ hardware devices. With LabVIEWs DAQ VIs you can develop complete instrumentation, acquisition, and control applications.

The **Data Acquisition** palette contains six subpalette icons that take you to the different classes of DAQ VIs. By clicking on any of the subpalette icon labels in the following illustration, you can learn about the classes of DAQ VIs in the subpalette.



DAQ VI Subpalettes

Analog Input
Analog Output
Digital I/O
Counter
Signal Conditioning
Calibration and Configuration

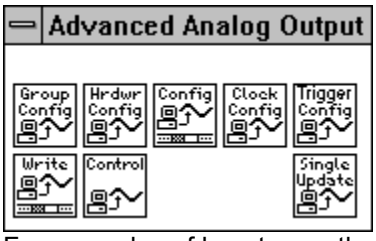
DAQ Error Codes

Click here to see a list of DAQ Error Codes.

Advanced Analog Output VIs

The Advanced Analog Output VIs are the interface to the NI-DAQ software and are the foundation of the Easy, Utility and Intermediate Analog Output VIs.

Access the **Advanced Analog Output** palette by choosing **Functions»Data Acquisition»Analog Output»Advanced Analog Output**. The icon that you must select to access the Advanced Analog Output VIs is on the bottom row of the **Analog Output** palette. Click on an icon in the following illustration for pop-up help.



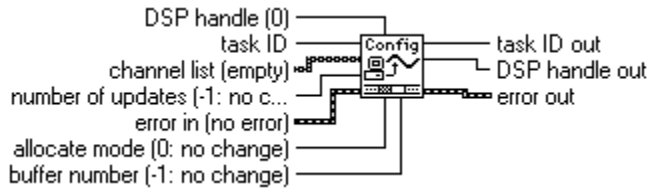
For examples of how to use the analog output VIs, see the examples in `examples\daq\anlogout.llb`.

Advanced VIs

AO Buffer Config
AO Buffer Write
AO Clock Config
AO Control
AO Group Config
AO Hardware Config
AO Single Update
AO Trigger and Gate Config (Windows and Sun)

AO Buffer Config

Allocates memory for an analog output buffer. If you are using interrupts, you can allocate a series of analog output buffers and assign them to a group by calling the AO Buffer Config VI multiple times. Each buffer can have its own size. If you are using DMA, you may allocate only one buffer.



Note: (*Macintosh*) If you are using the NB-A2100 with the NB-DMA2800, the AO Buffer Write VI restricts the amount of data that can be put into the VI to one-half of the buffer size specified in the AO Buffer Config VI.

Use the number you assign to the buffer with this VI when you need to refer to this buffer for other VIs.

U32 **taskID** identifies the group and the I/O operation.

abc **channel list**. If **channel list** has no elements, the VI assumes that the buffer holds data for all the channels in the group. In order for DMA to service a group, LabVIEW must interleave the data for all channels in the group in the same buffer. If you use DMA or interleave data for all the channels in the group in a single buffer, you do not need to create a **channel list**. The **channel list** parameter defaults to no elements.

If **channel list** contains one or more channels, the array defines a subset of the group. The VI

assumes that the buffer holds data for those channels in the group subset. All channels listed must belong to the same group.

 **buffer number** is a logical number that you can use to refer to the buffer at other points on the block diagram. When you assign a series of analog output buffers to a group, use different buffer numbers for each buffer.

The default input for **buffer number** is -1, which means LabVIEW leaves **buffer number** unchanged. The default setting for **buffer number** is 1.

 **number of updates** determines the size of the internal analog output buffer. If **channel list** is empty, the VI assumes the number of points in an update is equal to the number of channels in the group. In this case, the VI assumes the order of the channels in the buffer is the order of the channels in the group. If **channel list** is not empty, the number of points in an update is equal to the number of channels in **channel list**. The VI assumes the order of the channels in the buffer is the order of the channels in **channel list**.

(***Macintosh***) If you are using DMA, the buffer size cannot be greater than 224-1, which is 16,777,215 bytes. Remember that each update contains a point for each channel, and each point is 2 bytes.

The **number of updates** parameter defaults to -1, which means LabVIEW leaves **number of updates** unchanged. The default setting is 1000.

(***Macintosh***) If your device has an output FIFO and you have disallowed regeneration during buffer initialization, **number of updates** must be at least twice the size of your FIFO.

 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

 **allocate mode**.

0: Do not change allocate mode (default input).
1: Deallocate only the buffer named by **buffer number** or the current buffer if **buffer number** is -1.
2: Deallocate all the buffers assigned to the group.
3: Allocate memory (default setting).
4: Allocate DSP memory. The resulting handle is an output. *LabVIEW for Macintosh does not currently support this option.*
5: Use DSP memory that was previously allocated. **DSP handle** points to this memory. *LabVIEW for Macintosh does not currently support this option.*
6: (***Windows***) Use onboard FIFO memory. The entire waveform must fit in the FIFO. You cannot write data to the buffer once waveform generation has started. You can use this mode only with devices that have analog output FIFOs. Refer to the analog output capabilities tables in *Appendix B, Hardware Capabilities*, to determine if your device has an analog output FIFO. This option is only supported on LabVIEW for Windows.

 **DSP handle** is the DSP memory handle LabVIEW uses when **allocate mode** is 5. This option is only supported on LabVIEW for Windows.

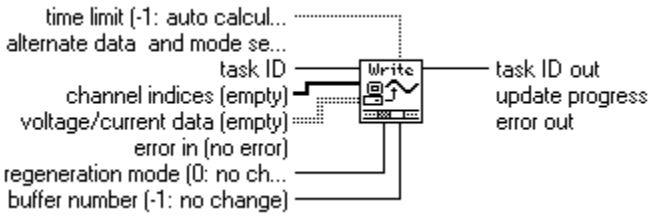
 **DSP handle out** is the DSP memory handle LabVIEW produces when **allocate mode** is 4. If **allocate mode** is 5, **DSP handle out** is equal to the **DSP handle** control.

 **taskID out** has the same value as **taskID in**.

 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

AO Buffer Write

Writes analog output data to buffers created by the AO Buffer Config VI.



The AO Buffer Write VI always begins writing at the update to which the write mark points. After a write, the write mark points to the update following the last update written.

taskID identifies the group and the I/O operation.

voltage/current data is a 2D array containing data in units of volts or milliamperes. The first dimension contains the data, and the second dimension contains the channel number.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

time limit is expressed in seconds. The resolution of the timeout clock is about 17 ms (***Macintosh***), 55 ms (***Windows***), 1 ms (***Sun***). The VI cannot write new data if LabVIEW is transferring data from the same region of the output buffer to which this VI is about to write. The VI waits until the analog output generation leaves this region before beginning the write. You can use **time limit** to limit this waiting period. This parameter defaults to -1, which means LabVIEW calculates a **time limit** setting. If LabVIEW cannot determine the update rate (for example, when the clock source is external), and **time limit** is -1, LabVIEW cannot calculate a **time limit**. In this case, LabVIEW uses a value of 1 s.

alternate data and mode set cluster contains the following parameters.

binary data is a 2D array that contains binary data that the VI writes to the DACs. The VI does not modify this data before writing it. The first dimension contains the data, and the second dimension contains the channel.

If you do not wire **voltage/current data**, LabVIEW uses **binary data**.

updates to write specifies how much data to write into the buffer when the analog output buffer resides in DSP memory. This option is only supported on LabVIEW for Windows.

DSP memory handle points to the analog output buffer when the buffer resides in DSP memory. This option is only supported on LabVIEW for Windows.

write offset. The VI adds the value of **write offset** to the write mark to determine the position at which the write begins. This parameter defaults to -1, which means LabVIEW leaves the **write offset** setting unchanged. The default setting is 0.

(***Macintosh***) If waveform generation has already started (the AO Control VI has executed with **control code** 0), **write offset** must be 0.

write mode.

0: Do not change the **write mode** setting (default input).
1: Relative to the write mark (default setting).
2: Relative to the beginning of the buffer.

Setting **write mode** to 2 moves the write mark to the beginning of the buffer before the VI adds **write offset** to the write mark.

(***Macintosh***) If waveform generation has already started (the AO Control VI has executed with **control code** 0), **write mode** must be either 1 or 0 (if the previous execution of the AO Buffer Config VI set **write mode** to 1).

channel indices determines which channels in the buffer named by **buffer number** receive new data. If you do not wire **channel indices**, the VI assumes the data array contains new data for all the

channels in the buffer. Use the Channel To Index VI to obtain a list of channel indices. Refer to the Calibration and Configuration VIs, for the Channel To Index VI description.

regeneration mode.

0: Do not change the regeneration mode setting (default input).
1: Allow regeneration of data (default setting).
2: Do not allow regeneration of data. When DMA is used, LabVIEW examines your waveform for stale data, only at the halfway point and at the end of the buffer. Data is fresh until it has been generated. Once the data has been generated, is it stale. So, LabVIEW cannot detect when stale data is about to be regenerated and stop immediately. LabVIEW can only stop halfway through or at the end of your waveform, so some old data is likely to be regenerated. When interrupts are used, LabVIEW examines every point for stale data and can stop immediately when it detects stale data and not regenerate any. If you want to prevent regeneration of data you must set regeneration mode to 2 when you call the AO Buffer Write VI before you start you waveform, not after.
3: This is the last write. Stop the waveform after this fresh data has been generated. Regardless of whether DMA or interrupts are used, LabVIEW will stop waveform generation immediately after generating the last point of this write.

(***Macintosh and Windows***) If **regeneration mode** is set to 2 or 3, your buffer must be at least twice the size of your analog output FIFO. Refer to your devices to see if your device has a FIFO and, if so, how big it is.

(***Macintosh***) The Lab and 1200 Series devices do not support **regeneration mode** values 2 and 3.

buffer number determines the buffer to which the VI writes. If the write mark already points to an update within the buffer that **buffer number** specifies, this input does nothing. Otherwise, the write mark moves to the buffer named by **buffer number**. The **buffer number** parameter defaults to -1, which means LabVIEW leaves the **buffer number** setting unchanged. The default setting is 1.

taskID out has the same value as **taskID in**.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

update progress contains the following parameters.

buffer number indicates the buffer to which the output parameters of this cluster (except for **channel state array**) apply.

output mark points to the next update LabVIEW is to generate.

write mark points to the next update to which LabVIEW is to write.

buffer iterations indicates the number of complete iterations of the buffer produced so far.

buffer state.

0: No such buffer (default).
1: Active.
2: Finished.
3: Waiting for data.
4: Ready.

The **buffer state** parameter is 3 when LabVIEW has allocated a buffer but has not yet written to it. **buffer state** is 4 when LabVIEW has written to a buffer but has not yet started it.

channel state.

0: Clear.
1: Running.
2: Finished.

3: Paused.

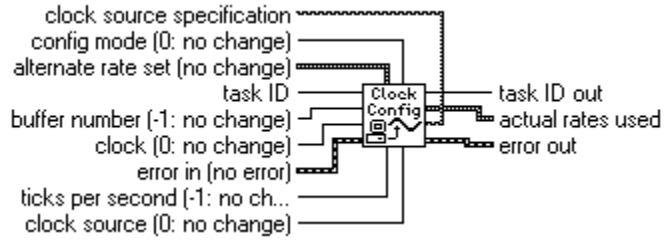
The VI always returns **channel state** information for all channels in the group. Use the Channel To Index VI to obtain an index for this array so that you can obtain the state of a particular channel.

You wire the new data to one of three inputs**voltage/current data**, **binary data**, or **DSP memory handle**. The VI searches these inputs in that order for the first array with a length greater than zero. The VI then writes the data from this array to the output buffer. The length of the **voltage/current data** or **binary data** arrays determines the number of updates the VI writes. If **DSP memory handle** points to the source of the data, **updates to write** must indicate how many updates the VI is to write. When no data is wired, this VI is still useful for reporting update progress information.

The total number of updates written to a buffer before you start it can be less than the number of updates you allocated the buffer to hold when you called the AO Buffer Config VI. LabVIEW generates only the updates written to the buffer.

AO Clock Config

Configures an update or interval clock for analog output.



Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the clocks available with your DAQ device.

clock source specification is a string control where you enter the PFI or RTSI pin number to use when clock source is 8, 9, 10 or 11. Simply enter the pin number without any other text. If **clock source** is 8 and **clock source specification** contains the string 3 then you are selecting the signal present at PFI 3 as your clock source. LabVIEW for Macintosh does not support this parameter.

taskID in identifies the group and the I/O operation.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

ticks per second defaults to a -1.0 input. This means LabVIEW does not use this parameter, and skips to the **seconds per tick** input for clock rate information. The default setting for **ticks per second** is undefined. To turn off the clock, set **ticks per second** to 0.0. If you set **ticks per second** to *n*>0.0 for an update clock, LabVIEW updates the channels in the group *n* times each second. When you use the **ticks per second** control to specify the update rate, you do not need to use the **alternate rate set** cluster.

config mode.

0: Do not change the config mode setting (default input).
1: Change rate immediately (default setting).
2: Change rate at the end of the current buffer iteration. This option is only supported on LabVIEW for Macintosh.
3: Translate only. Do not change the clock rate settings.

When **config mode** is 1, you do not need to wire an input to this control.On the Macintosh, only the NB-A2100 supports **config mode** 2.

alternate rate set contains the following parameters.

seconds per tick defaults to an input of -1.0. This means LabVIEW does not use this parameter, but skips to the three timebase parameters for clock rate information. The default setting for **seconds per tick** is undefined. To turn off the clock, set **seconds per tick** to 0.0.

timebase source.

0: Do not change the timebase source (default input).
1: Internal frequency in hertz (default setting).
2: Source n.
3: Gate n.
4: Reserved.
5: PFI pin, low to high (E Series only: low to high edges of the signal connected to PFI pin n serve as your timebase).
6: PFI pin, high to low (E Series only: high to low edges of the signal connected to PFI pin n. serve as your timebase).
7: RTSI pin, low to high (E Series only: low to high edges of the signal connected to RTSI pin n serve as your timebase).
8: RTSI pin, high to low (E Series only: high to low edges of the signal connected to RTSI pin n serve as your timebase).
9: ATCOut, low to high (E Series only: low to high edges of the analog trigger circuitry output serve as your timebase).
10: ATCOUT, high to low (E Series only: high to low edges of the analog trigger circuitry output serve as your timebase).

When **timebase source** is 1 use the **timebase signal** control to enter the frequency in hertz.

When **timebase source** is 2 through 8 use the **timebase signal** control to enter the value for the Source, Gate or pin number.

timebase signal works with **timebase source** to fully specify the timebase. If **timebase source** is 1, **timebase signal** is the frequency in hertz. If **timebase source** is 2 or 3, **timebase signal** is the number of the counter associated with the source or gate input, respectively. If **timebase source** is 5 or 6, **timebase signal** is the PFI pin number (range 0 through 9). If **timebase source** is 7 or 8, **timebase signal** is the RTSI pin number (range 0 through 6). If **timebase source** is 9 or 10, **timebase signal** is ignored. The default input for **timebase signal** is -1.0, which tells LabVIEW not to change **timebase signal**. The default setting for **timebase signal** is undefined.

Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, to determine the range of timebase frequencies available on your device.

timebase divisor is the divide-down value the VI uses to create the clock rate. This parameter defaults to an input of -1, which means LabVIEW leaves **timebase divisor** unchanged. The default setting is undefined.

The following example shows how to use the three timebase parameters to specify a clock rate. Assume that **timebase source** is 1, **timebase signal** is 1,000,000.0 Hz, and **timebase divisor** is 25. This produces an update rate of 1,000,000.0/25 or 4,000 ticks per second. This means the VI updates the analog output channels 4,000 times/s.

buffer number is the buffer to which the clock configuration applies.

-1: Do not change the **buffer number** setting.
0: Apply the clock configuration to *all* buffers assigned to the group.
1: Apply the clock configuration to buffer number 1 (default setting).
n: Apply the clock configuration to buffer *n*.

If you set **buffer number** to 0, you can call this VI to establish default clock settings for all buffers.

If you call the AO Clock Config VI with a value of 0 for **buffer number** and then call the VI with **buffer number** set to a specific buffer (for example, buffer *n*), buffer *n* has a unique clock configuration. This means all the other buffers have the clock configuration set by the first call. The **buffer number** parameter defaults to a setting of 1. If you use only one buffer, and you assigned it to a value of 1, you do not need to wire an input to this control.

clock does not need to be wired unless you want to use a clock other than update clock 1.

0: Do not change the **clock** setting (default input).
1: Update clock 1 (default setting).
2: Interval clock 1. Only supported on LabVIEW for Windows.
3: Update clock 2. Only supported on LabVIEW for Windows.

Update clocks are the primary analog output clocks. Each time an update clock produces a pulse, LabVIEW updates the DACs in the group with new voltages. You must configure an update clock to generate a waveform. All devices that support clocked analog output have an update clock 1. Only the AT-AO-6/10 has an update clock 2.

(***Windows***) Use the interval clock only when the entire waveform buffer can fit inside the analog output FIFO on the device. You can use the interval clock to set a time interval between cycles during which LabVIEW does not produce update pulses. For example, if the analog output buffer contains one cycle of a sine wave and you specify a buffer iteration count of 2 (when the control VI starts the waveform) and set up interval clock 1 to count for one second, the output voltage does not change for one second after every two cycles of the wave. The most basic way to specify this time interval is by using the **seconds per tick** input. The **seconds per tick** parameter is the length of the interval in seconds.

clock source.

0: Do not change the clock source setting (default input).
1: InternalLabVIEW picks the clock (default setting).
2: Counter 1. This option is only supported on Windows.
3: Counter 2. This option is only supported on Windows.
4: Counter 3. This option is only supported on Windows.
5: Counter 5.This option is only supported on Windows.
6: I/O connector.
7: RTSI connector. This option is not supported on LabView for Sun.
8: PFI pin, low to high (E Series only: low to high edges of the signal connected to PFI pin n update your analog output).
9: PFI pin, high to low (E Series only: high to low edges of the signal connected to PFI pin n update your analog output).
10: RTSI pin, low to high (E Series only: low to high edges of the signal connected to RTSI pin n update your analog output).
11: RTSI pin, high to low (E Series only: high to low edges of the signal connected to RTSI pin n update your analog output).
12: GPCTR 1 output, low to high (E Series only: low to high edges of the output signal of GPCTR1 update your analog output).
13: GPCTR 1 output, high to low (E Series only: high to low edges of the output signal of GPCTR1 update your analog output).
14: ATCOUT, low to high (E Series only: low to high edges of the analog trigger circuitry output update your analog output).
15: ATCOUT, high to low (E Series only: high to low edges of the analog trigger circuitry output update your analog output).

(***Windows***) If you use settings 2, 3, 4, and 5, you can choose which internal clock provides the update signal. These options are valid only when you set clock to 1, (select update clock 1), and use an AT-MIO-16F-5, AT-MIO-16X, or AT-MIO-64F-5 (counter 3 is available only on the latter two devices).

Settings 6 and 7 tell LabVIEW that the clock signals are produced externally and where they connect. If

you have an E Series device you should not use settings 6 or 7 but choose among settings 8 through 11 instead.

Settings 8 through 15 are for E Series devices only. Settings 8 through 11 select a PFI pin or RTSI pin. The actual pin number is entering in the clock source specification control. Settings 12 and 13 select the output of GPCTR1 so you must use the CTR VIs to set up this counter to produce a signal. Settings 14 and 15 select the ATCOUT signal so you must use the AI Trigger Config VI to configure the analog trigger circuitry to produce this signal.

You typically use the internal clock source and therefore do not need to wire an input to this control.

You can express clock rates three wayswith **ticks per second**, **seconds per tick**, or the three timebase parameters. The VI searches these parameters in that order and expresses clock rates on the first parameter with a wired valid input. When you configure an update clock, one tick equals one update. When you configure the interval clock, one tick equals one interval.

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

I32 **actual rates used**. Because LabVIEW uses discrete timebases to produce the update pulses, it may not be able to produce the exact update rate you request. The **actual rates used** cluster returns the update rate LabVIEW chooses. This cluster contains the following parameters.

SGL **ticks per second used**.

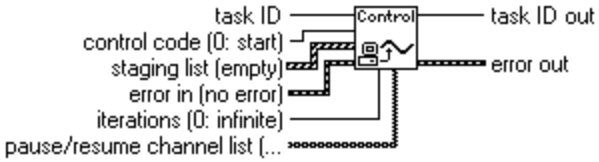
SGL **seconds per tick used**.

SGL **timebase signal used** is the **timebase signal** (expressed in hertz) that the VI used when you set **timebase source** to 1.

I32 **timebase divisor used** is the **timebase divisor** that the VI used.

AO Control

Starts, pauses, resumes, and clears analog output tasks.



I32 **taskID in** identifies the group and the I/O operation.

I32 **control code**.

0: Start (default input).
1: Pause.
2: Pause at the end of the current buffer iteration. LabVIEW does not currently support this option.
3: Resume.
4: Clear.
5: Switch buffer. LabVIEW does not currently support this option.
6: Switch buffer at the end of the current buffer iteration. LabVIEW does not currently *support this option*.

The start operation, **control code** 0, starts the clocks associated with the group.

The pause and resume operations, **control code** 1 and 3 pause and resume either an entire group, or if **pause/resume channel list** is not empty and you are not using DMA, only those

channels listed in **channel list**. The VI ignores the **staging list** cluster array when **control code** is 1 or 3.

The clear operation, **control code** 4, stops the clock associated with a group and deallocates all internal analog output buffers. The VI ignores **pause/resume channel list** and **staging list** when **control code** is 0.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

iterations determines the number of times the VI generates the buffer(s). If you use the **iterations** control, the VI applies the value you assign the control to all the buffers the VI starts.

1: Do not use this control. Use the buffer iterations in **staging list** instead.
0: Infinite iterations (default input and setting).
n: *n* iterations.

All the Macintosh devices except the NB-A2100 support *n* = 1 and *n* = 0 iterations only. The NB-A2100 supports *n* ³ 1.

staging list is an array of clusterseach containing four arrays with buffer numbers, iteration counts, interval iteration counts, and update rates. The VI starts the buffers listed in the first cluster of **staging list** when **control code** is 0. If the length of **staging list** is 0, the VI starts all buffers. Unless you have more than one buffer to which you want to assign different iteration counts, you do not need to create a **staging list**. LabVIEW currently uses only the first cluster in **staging list**.

buffer numbers lists the buffers the VI starts.

buffer iterations lists the iteration counts for the buffers in **buffer numbers**. An empty list or a value of 0 for **buffer iterations** tells LabVIEW to iterate indefinitely. If the length of **buffer iterations** is less than the length of **buffer numbers**, the last value in **buffer iterations** applies to the remaining buffers.

Note: The iterations control takes precedence over this control.

interval iterations. *LabVIEW does not currently support this option.*

update rate lists the update rate in ticks per second to apply to the buffers in the **buffer numbers** list. If the length of **update rate** is less than the length of **buffer numbers**, the VI applies the last value in **update rate** to the remaining buffers. A value of -1.0 in **update rate** tells LabVIEW to use the rate you specify with the AO Clock Config VI. Otherwise, **update rate** overrides all the AO Clock Config VI settings except for the choice of update clock. *LabVIEW does not currently support this option.*

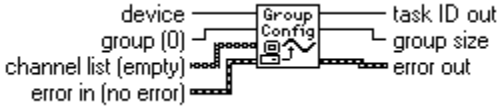
pause/resume channel list contains the channels to which the pause and resume control codes apply. Individual channels in a group may be paused and resumed only when DMA is not being used.

taskID out has the same value as **taskID in**.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

AO Group Config

Assigns a list of analog output channels to a group number and produces the taskID that all the other analog output VIs use.



Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the channels available with your DAQ device.

device is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

channel list contains the set of analog output channels to which the VI assigns a group number. You must not list a channel more than once in **channel list.** The elements of **channel list** follow the syntax outlined in the AI Group Config VI description. To specify a single channel, *x*, enter `x` or `OBx` as a **channel list** element. Use a comma to separate a list of channels and a colon to specify a range of channels. On the Macintosh or in Windows, you can remove the assignment on all the channels in the group by passing an empty string ( ) into the **channel list**.

(***Macintosh and Windows***) The architecture of the AT-AO-6/10, NB-MIO-16, and NB-AO-6 devices impose certain restrictions on **channel list**. Refer to the user manual for those particular devices for more information.

(***Macintosh and Windows***) If you are using one or more SCXI-1124 modules, use the `SCn!MDm!CHx:y` syntax where `n` is your chassis ID, `m` is your module slot number, and `x` and `y` are the start and stop channels of a consecutive range. You may combine channels from multiple SCXI-1124 modules in a single group.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

group is the number the VI assigns to the set of channels. The **group** parameter ranges from 0 to 15 and has a default input and setting of 0. If you do not call the AO Group Config VI before using one of the other VIs in this chapter, LabVIEW creates a default **group** with a group number of 0 for you. The default **group** 0 contains channels 0 through *n*, where *n* is the maximum number of channels minus 1. The maximum number of channels varies with device type.

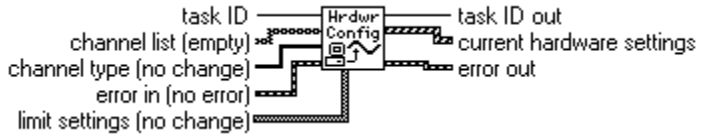
taskID out has the same value as **taskID in**.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

group size indicates the number of channels in the group. For example, if **channel list** is 0:3, **group size** is 4. The **group size** parameter is also known as the update width.

AO Hardware Config

Configures the reference voltage level, output polarity, and the unit of measure for the data of a given channel (volts or milliamperes). This VI always returns the current settings for all the channels in the group.



Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the channel ranges, and output limits available with your DAQ device.

taskID in identifies the group and the I/O operation.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

limit settings is an array of clusters. Each array element specifies the limits for a channel or channels. The **channel list** description explains how LabVIEW applies the limits to the elements of **channel list**.) If the **limit settings** array length is less than the group size (if you configured on a per

group basis) or less than the length of **channel list** (if you configured on a per channel basis), the values in the last element of the **limit settings** array apply to the remaining channels. Each cluster contains the following parameters.

I32 **upper limit** is equal to your reference voltage and is the maximum voltage the DAC can produce. (***Macintosh and Windows***) If you have an AT-AO-6/10, NB-AO-6, or SCXI-1124 device and your **channel type** is current, you calculate the maximum possible current with the following equation.

$$I_{max} = \frac{V_{ref} + 2.5}{0.625} mA$$

I32 **reference source**.

0: Do not change the **reference source** setting (default input).
1: Internal (default setting).
2: External.

I32 **channel list**. If **channel list** is empty, the VI configures hardware on a group basis. If **channel list** contains one or more channels, the VI configures the hardware on a channel basis.

The VI applies the following values contained in **channel type** and **limit settings** (the configuration arrays) to the channels in the group (group basis), or the channels listed in **channel list** (channel basis). The values listed first in the configuration arrays (at index 0) apply to the first channel in the group or the channel(s) listed in index 0 of **channel list**. The values listed second in the configuration arrays (at index 1) apply to the second channel in the group or the channel(s) listed in index 1 of **channel list**. The values in each configuration array apply to channels in the group or **channel list** in this fashion until the VI exhausts the configuration arrays. If channels in the group or **channel list** remain unconfigured, the final value listed in the configuration arrays applies to all the remaining unconfigured channels.

U16 **channel type** determines whether a channel is a voltage or a current channel. LabVIEW expresses voltage channel data in volts. When a channel you configure is a current channel, LabVIEW assumes data for that channel is expressed in milliamperes.

0: Do not change the type for the channel (default input).
1: Voltage channel (default setting).
2: Current channel. LabVIEW for Sun does not currently support this option.

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

[ ] **current hardware settings** is an array of clusters that returns the hardware settings for all the channels in the group. Each cluster contains the following parameters.

abc **channel** is the channel number to which the other elements of the cluster apply.

I32 **channel type** is the voltage or current setting of this channel.

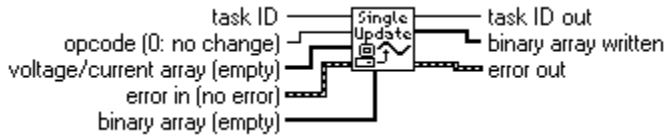
I32 **upper limits** is the upper limit setting of this channel.

I32 **lower limits** is the lower limit setting of this channel.

I32 **reference source** is the internal/external setting of the reference voltage for this channel.

AO Single Update

Performs an immediate update of the channels in the group.



I32 **taskID in** identifies the group and the I/O operation.

I32 **opcode**.

0: Do not change the **opcode** setting (default input).
1: Output data and update DACs (default setting).
2: Output data only, do not update DACs. *LabVIEW for Sun does not currently support this option.*
3: Update DACs.
4: Do not output data. Translate voltage or current to binary data only.

Each DAC is internally buffered, so using a DAC to produce a voltage is a two-step process. First, write (or output) new data to the DAC. Second, send an update pulse to the DAC to produce the D/A conversion and the new voltage. Typically, a clock in a waveform generation application produces these update pulses, but a software command can also produce an update pulse.

To performs both steps, set **opcode** to 1 and a software command produces the update pulse. To complete only the first step, set **opcode** to 2. Set **opcode** to 3 to perform only the second step.

You do not need to wire an input to this control, unless you want to use delayed update mode (by setting **opcode** to 2 or 3) or only translate data (by setting **opcode** to 4).

I32 **voltage/current array** contains the data in volts or milliamperes.

I32 **binary array** contains the data scaled to 12-bit or 16-bit DAC values.
The **voltage/current** and **binary** data arrays are 1D arrays. The VI writes the data in element *i* of these arrays to the *ith* channel in the **channel list** you used to configure the group. For example, if your group contains the channels 2 and 5 in that order, the VI writes data at index 0 of the data array to channel 2 and the data at index 1 to channel 5. In this example, if the data array contains just one value, the VI writes only to channel 2.

The VI decides which array to output depending on which of these two arrays you wire. The precedence order is **voltage/current** array and then **binary** array.

(***Macintosh and Windows***) The output channel type determines whether the VI interprets an entry in **voltage/current array** as a voltage or a current. The configuration utility or the AO Hardware Config VI sets this output channel type. If the VI interprets the data as current values, it calculates corresponding voltages and writes them to the channels. The following shows the formula for the conversion.

voltage = 0.625 * (current in milliamperes) 2.5

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

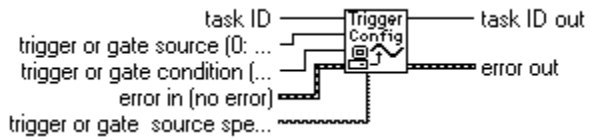
I16 **binary array written** contains the binary DAC values the VI calculated from the data in **voltage/current array**.

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

AO Trigger and Gate Config (Windows and Sun)

Configures the trigger and gate conditions for analog output operations on E Series devices.



I32 **taskID in** identifies the group and the I/O operation.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **trigger or gate source** selects the source of your trigger or gate signal.

0: No change (default input).
1: None (default setting).
2: PFI pin.
3: RTSI pin.
4: AI Start Trigger.
5: ATCOUT (the output of the analog trigger circuitry).

Setting **trigger or gate** source to 1 (none) turns off any trigger or gates you have configured.

If **trigger or gate source** is 2 or 3 you must use the **trigger or gate specification** control to enter your PFI or RTSI pin number.

If youve set up a triggered analog input and want to use the same trigger to start your analog output, set **trigger or gate source** to 4.

If you want to trigger off an analog input signal with or without a corresponding analog input operation, set **trigger or gate source** to 5.

I32 **trigger or gate condition** selects a rising or falling edge trigger or a high or low level gate.

0: No change (default input).
1: None (default setting).
2: Trigger on rising edge.
3: Trigger on falling edge.
4: Pause while gate is TTL high.
5: Pause while gate is TTL low.

Setting **trigger or gate condition** to 1 (none) turns off any trigger or gates you have configured.

I32 **trigger or gate source specification** identifies your PFI or RTSI pin number when you have set the trigger or gate source control to 2 or 3. Simply enter the number without any other text. The range for PFI pin is 0 through 9. The range for RTSI pin is 0 through 6.

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

AO Buffer Config VI

AO Buffer Config

AO Buffer Write VI

AO Buffer Write

AO Clock Config VI

AO Clock Config

AO Control VI

AO Control

AO Group Config VI

AO Group Config

AO Hardware Config VI

AO Hardware Config

AO Single Update VI

AO Single Update

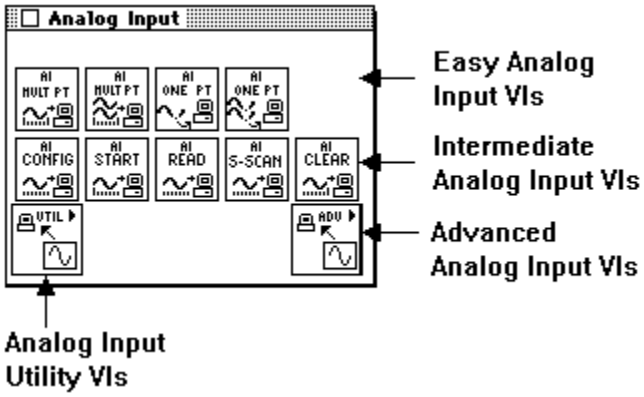
AO Trigger and Gate Config VI

AO Trigger and Gate Config (Windows and Sun)

Analog Input VIs

This chapter explains the Analog Input VIs and shows where you can find them in LabVIEW.

There are four classes of Analog Input VIs found in the **Analog Input** palette. The Easy Analog Input VIs, Intermediate Analog Input VIs, Analog Input Utility VIs, and Advanced Analog Input VIs. The following illustration shows these different VI classes. Click on the label next the VIs for a brief explanation of the class of VIs or click on an actual icon to go to that particular VI.



For examples of how to use the analog input VIs, see examples\daq\anlogin.llb or examples\daq\anlog_io.llb.

Note: **Use only the inputs that you need on each VI. LabVIEW sets all unwired inputs to their default values. Many of the DAQ function inputs are optional and do not appear in the Simple Diagram Help window. These inputs typically specify rarely-used options. If an input is required, your VI wiring remains broken until a value is wired to the input. Required inputs appear in bold in the Help window, recommended inputs appear in plain text, and optional inputs are in gray text. The default values for inputs appear in parentheses beside the input name in the Help window.**

Easy Analog Input - Accessing the VI Palette

The Easy Analog Input VIs are stand-alone executables, which means you only need one Easy Analog Input VI to perform each basic analog input operation. Unlike intermediate- and advanced-level VIs, Easy Analog Input VIs automatically alert you to errors with a dialog box that asks you to stop the execution of the VI or to ignore the error.

The Easy Analog Input VIs are actually composed of Intermediate Analog Input VIs, which are in turn composed of Advanced Analog Input VIs. The Easy Analog Input VIs provide a basic, convenient interface with only the most commonly used inputs and outputs. For more complex applications, you should use the intermediate- or advanced-level VIs for more functionality and performance.

Click here to go to the **Easy Analog Input VI** palette and descriptions

Intermediate Analog Input - Accessing the VI Palette

You can find intermediate-level Analog Input VIs in two different places in the **Analog Input** palette. You can find the Intermediate Analog Input VIs in the second row of the **Analog Input** palette. The other intermediate-level VIs are in the **Analog Input Utilities** palette, which will be discussed later. The Intermediate Analog Input VIs are built from the fundamental building block layer, called the Advanced Analog Input VIs. These VIs offer almost as much power as the advanced-level VIs, and they conveniently group the advanced-level VIs into a tidy, logical sequence.

Click here to go to the **Intermediate Analog Input** palette and descriptions.

Analog Input Utility - Accessing the VI Palette

You can access the **Analog Input Utilities** palette by choosing the Analog Input Utility icon from the **Analog Input** palette. The Analog Input Utility VIs are single-VI solutions to common analog input problems. These VIs are convenient, but they lack flexibility. These three VIs are built from the Intermediate Analog Input VIs in the **Analog Input** palette.

Refer to Analog Input Utility VIs for specific VI information.

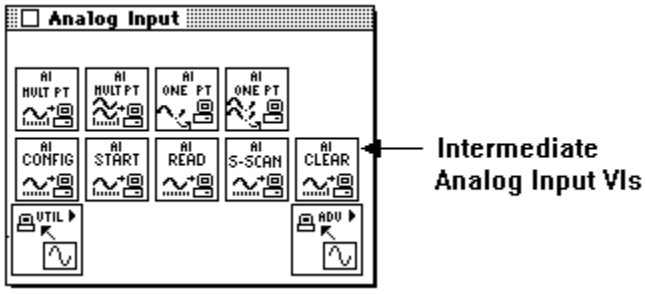
Advanced Analog Input - Accessing the VI Palette

You can access the **Advanced Analog Input** palette by choosing the Advanced Analog Input icon from the Analog Input palette. These VIs are the interface to the NI-DAQ data acquisition software and are the foundation of the Easy, Utility, and Intermediate Analog Input VIs. Because all these VIs rely on the advanced-level VIs, you can refer to the Advanced Analog Input VIs for additional information on the inputs and outputs and how they work.

Intermediate Analog Input VIs

This section describes the Intermediate Analog Input VIs. The Intermediate Analog Input VIs are built from the fundamental building block layer, called the Advanced Analog Input VIs. These VIs offer almost as much power as the advanced-level VIs, and they conveniently group the advanced-level VIs into a tidy, logical sequence.

You can access the Intermediate Analog Input VIs by choosing **Functions**»**Data Acquisition**»**Analog Input**. The Intermediate Analog Input VIs are the VIs on the second row of the **Analog Input** palette, as shown below. Click on a VI icon in this illustration to go to a particular Intermediate Analog Input VI.



For examples of how to use the Intermediate Analog Input VIs, open the example library by opening `examples\daq\anlogin.llb`.

Intermediate VIs

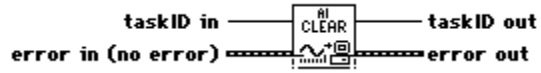
AI Clear
AI Config
AI Read
AI Single Scan
AI Start

Other Intermediate VIs

Handling Errors

AI Clear

Clears the analog input task associated with **taskID in**.



The AI Clear VI uses the AI Control VI to stop an acquisition associated with **taskID in** and release associated internal resources, including buffers. Before beginning a new acquisition, you must call the AI Config or AI Buffer Config VIs.

The AI Clear VI always clears the acquisition, even if **error in** indicates an error.

I32 **taskID in** identifies the group and the I/O operation.

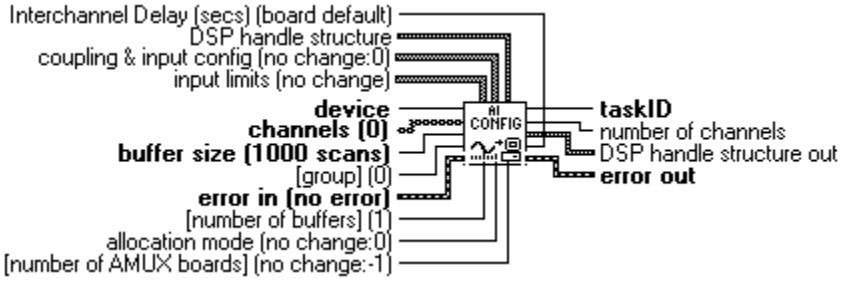
I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

AI Config

Configures an analog input operation for a specified set of channels. This VI configures the hardware and allocates a buffer for a buffered analog input operation.



This VI calls several advanced analog input VIs to set up a task for a buffered analog input acquisition. The AI Config VI calls the AI Group Config VI for the specified device, group, and channels to create a taskID. The AI Config VI then calls the AI Hardware Config VI to record information about the hardware configuration. The **input limits** input passes to the AI Hardware Config VI, which sets up the lower and upper voltage limits for each channel. If you leave the limits unspecified, the **input limits** stay the same. This VI configures the **number of AMUX boards**, the **coupling** used for each channel (that is, AC, DC, ground, or internal reference), and the **input configuration** (differential, referenced single-ended, or non-referenced single-ended). You can use the advanced AI Hardware Config VI instead if you prefer to specify the input signal range in terms of device range, polarity, and gain.

The AI Config VI configures the buffer for the acquisition by calling the AI Buffer Config VI. The AI Buffer Config VI allocates memory for the specified **number of buffers** (the default value is 1), where each buffer contains **buffer size** number of scans (the default value is 1000).

You can allocate more than one buffer only with the following devices.

- (Macintosh) NB-A2000, NB-A2100, and NB-A2150
- (Windows) EISA-A2000, AT-A2150, and AT-DSP2200
- (Sun) SB-A2200

Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual,* for the channel ranges, input limits, and scanning order you can use with your National Instruments DAQ device.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **channels** is a list of the analog channels you want to use. If `x`, `y`, and `z` refer to channels, you can specify a list of channels by separating the individual channels with commasfor example, `x,y,z`. If `x` refers to the first channel in a consecutive channel range and `y` refers to the last channel, you can specify the range by separating the first and last channels by a colonfor example, `x:y`. Refer to the *Channel Addressing with the AMUX-64T* section of Chapter 5, *Things You Should Know about Analog Input*, in the *LabVIEW Data Acquisition Basics Manual*, for the syntax you use for AMUX-64T channels or SCXI.

I32 **buffer size** is the number of scans you want each buffer to hold. This VI defaults to an input of 1000 scans.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **Interchannel Delay**. For devices with both scan and channel clocks, you can use Interchannel Delay to specify the waiting time between sampling channels within a scan. LabVIEW selects a default

Interchannel Delay automatically, giving the hardware time to settle between channels. The default value for **Interchannel Delay** is -1.0, which tells the AI Config VI to use the channel clock rate LabVIEW selects. Refer to Chapter 9, *Letting an Outside Source Control Your Acquisition Rate*, of the *LabVIEW Data Acquisition Basics Manual* for an explanation of how LabVIEW selects the default channel clock rate.

I32 **DSP handle structure** specifies a block of DSP memory that LabVIEW uses for the data acquisition when **allocation mode** is 4 (use the DSP memory that another VI previously allocated). Click here to see the **DSP handle structure** cluster contains the following parameters.

I32 **size** is not used.

I32 **DSP memory handle** specifies a block of DSP memory.

Note: **The DSP handle structure cluster is available only with LabVIEW for Windows.**

I32 **coupling & input config** is an array of clusters. Each array element contains the configuration for the channel or channels specified by the corresponding element of the **channels** array. If there are fewer elements in this array than in the **channels** array, the VI uses the last array element for the rest of the channels. The **coupling & input config** array defaults to an empty array, which means the parameters keep their default settings. Each cluster contains the following parameters.

I32 **coupling**.

0: Do not change the **coupling** setting.
1: DC.
2: AC.
3: Ground.
4: Internal reference.

I32 **input config**.

0: Do not change the **input config** setting.
1: Differential (default).
2: Referenced single-ended.
3: Nonreferenced single-ended.

I32 **input limits** is an array of clusters. Each array element contains the expected signal limits for the channels specified by the corresponding element of **channels**. If there are fewer elements in this array than in **channels**, the VI uses the last array element for the rest of the channels. The **input limits** array defaults to an empty array, which means the **input limits** keep their default settings. Click here to see the **input limits** cluster parameters.

I32 **group** is the number, from 0 to 15, that you assign to the specified set of channels. The default input and setting for **group** is 0. If you only have one acquisition for this device, leave this input unwired and use group 0.

I32 **number of buffers** is the number of buffers the VI allocates. This parameter defaults to 1, and you should change it only for multi-buffered acquisition.

I32 **allocation mode** determines how LabVIEW allocates the buffer.

0: Do not change the **allocation mode** setting (default input).
1: Do not allocate memory, ignore **buffer size** and **number of buffers**, and deallocate any existing memory.
2: Allocate memory on the host computer (default setting).
3: (Windows) Allocate DSP memory and return the handle in **DSP memory handle out**.
4: (Windows) Use the DSP memory that another VI previously allocated. **DSP memory handle** points to this memory.

I32 **number of AMUX boards** is the number of AMUX-64T boards attached.

0: No AMUX-64T boards (default).
1: (Macintosh and Windows) One AMUX-64T board.
2: (Macintosh and Windows) Two AMUX-64T boards.

4: (Macintosh and Windows) Four AMUX-64T boards.

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

I32 **number of channels** is the total number of channels in the group, including any AMUX or SCXI channels.

I32 **DSP handle structure out** specifies the block of DSP memory LabVIEW uses for the data acquisition. When **allocation mode** is 3, LabVIEW allocates DSP memory and when **allocation mode** is 4, LabVIEW uses the DSP memory that another VI previously allocated. The **DSP handle structure out** cluster contains the following parameters.

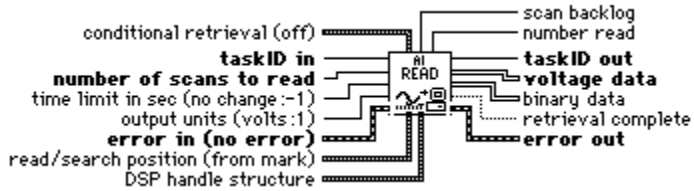
I32 **size** specifies the number of two-byte or four-byte elements in the **DSP memory handle** data buffer.

I32 **DSP memory handle** specifies a block of DSP memory.

Note: The DSP handle structure out cluster is not supported with LabVIEW for Macintosh.

AI Read

Reads data from a buffered data acquisition.



The AI Read VI calls the AI Buffer Read VI to read data from a buffered analog input acquisition.

I32 **taskID in** identifies the group and the I/O operation.

I32 **number of scans to read** is the number of scans the VI is to retrieve from the acquisition buffer. The default input is -1, which tells LabVIEW to set **number of scans to read** equal to the value of the **number of scans to acquire** control when the AI Start VI was called. If **number of scans to read** is -1 and **number of scans to acquire** was 0, LabVIEW sets **number of scans to read** to 100.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **conditional retrieval** is an artificial form of triggering. You can use this for devices that do not offer analog triggering. Conditional retrievals, unlike actual triggers, have no effect on the actual data acquisition.

With **conditional retrieval**, you specify trigger conditions for which the VI searches within the acquisition buffer. The search begins at the location specified by **read/search position**. After the VI finds the retrieval conditions in the buffer, the VI returns data starting at this location plus the offset. The search does not cross buffer boundaries for multiple buffer acquisitions.

I32 **mode** enables or disables conditional retrieval.

0: Do not change the **mode** setting (default input).
1: Off (default setting).
2: On.

I32 **channel index** specifies which channels data the VI searches to find the retrieval condition. You specify the channel in the scan list using an index. Use the Channel To Index VI to produce this index. The default input is -1, which tells LabVIEW not to change **channel index**. The default setting is 0.

slope specifies whether to search for a rising or falling voltage level.

0: Do not change the **slope** setting (default input).
1: Rising (default setting).
2: Falling.

level specifies the voltage level for which the VI searches. You express **level** in volts. The default input and setting are 0.0 V.

hysteresis specifies the voltage window that the signal must leave to meet the retrieval conditions. That is, **hysteresis** acts like a noise filter. For example, if you set **slope** to 1 (rising), set **level** to 1.0 V, and set **hysteresis** to 0.1 V, the signal must drop below 0.9 V before retrieving. If you change the **slope** to 2 (falling), the signal has to rise above 1.1 V before retrieving. You express **hysteresis** in volts. The default input and setting are 0.0 V.

skip count is the number of positive retrieval conditions (triggers) the VI skips before the final retrieval condition occurs. The default input is -1, which tells LabVIEW not to change the **skip count** setting. The default setting is 0, which tells the VI not to skip any triggers.

offset indicates the scan location from which the VI begins reading data relative to the retrieval condition. A negative number indicates data prior to the retrieval condition (pretrigger data), and a positive number indicates data after the retrieval condition (posttrigger data). The default setting is 0.

time limit in sec is the time limit for the read operation. The default input is -1.0, which means LabVIEW calculates a time limit based on the value of **number of scans to read** and the scan rate. If the scan rate is unknown, the VI uses the current time limit setting. The default setting is 1 s. The resolution of the timeout clock is about 17 ms (Macintosh) , 55 ms (Windows) , 1 ms (Sun) .

output units specifies whether the VI returns unscaled binary data or volts.

0: Do not change the **output units** setting (default input).
1: Return voltage data only (default setting). The **binary data** array appears empty.
2: Return binary data only. The **voltage data** array appears empty. The VI executes faster if you select this value, because the VI does not perform scaling.
3: Return voltage and binary data.

read/search position defines where you want to read from the acquisition buffer. For conditional retrievals, **read/search position** specifies where to begin the search in the acquisition buffer. **read/search position** contains the following parameters.

position indicates which input buffer reference mark is the read reference point. This reference can be the read mark, the beginning of the buffer, or the most recently acquired data (end of data). Initially, the read mark points to the beginning of the acquisition buffer. As you retrieve data from the buffer using this VI, LabVIEW increments the read mark to point to the next block of data to be read.

0: Do not change the **position** setting (default input).
1: Relative to the read mark (default setting).
2: Relative to the start of buffer.
3: Relative to the end of data.
4: Relative to the trigger point. This mode is valid only when the buffer completes.

read offset. The VI adds **read offset** scans to the mark specified by **position** to determine the starting point for the read. Refer to the description of the AI Buffer Read VI for more information.

DSP handle structure identifies a block of DSP memory into which the VI copies the data. If you do not wire this cluster, the VI copies the data into the **voltage data** or **binary data** arrays. **output units** must be 1 (volts) or 2 (binary) when you use DSP memory. The **DSP handle structure** cluster contains the following parameters.

size is not used.

DSP memory handle identifies a block of DSP memory.

taskID out has the same value as **taskID in**.

voltage data is a 2D array that contains analog data. (The AI Read One Scan, AI Single Scan,

AO Write One Update, and AO Single Update VIs return only a 1D array in the **voltage data** parameter.) This array is scan-ordered which means that each row contains the data of a single scan or value. If you want to index the array in a loop, you will get a scan on each iteration. Each column holds the data of a single channel. The upper array index selects the scan and the lower index selects the channel. Refer to the Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, in *the LabVIEW Data Acquisition Basics Manual* for more information about the array format.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

scan backlog is the amount of data remaining in the buffer after this VI completes. If **scan backlog** increases steadily, you are not reading data fast enough to keep up with the acquisition, and your newly-acquired data may overwrite unread data and give you an overwrite error. Decrease the **scan rate**, increase the **number of scans to read**, read the scans more often, or increase the **buffer size**. See the description of the AI Buffer Read VI for a precise definition of how to computer **scan backlog**.

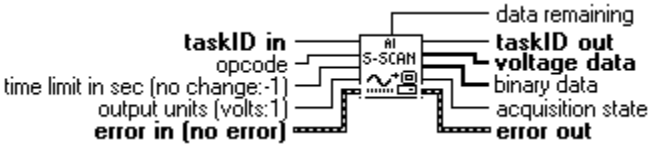
number read is the number of scans returned. This number is identical to **number of scans to read** unless an error or timeout appears or the VI reaches the end of the data.

binary data is a 2D array that contains analog input data. This array is scan-ordered which means that each row contains the data of a single scan. If you want to index the array in a loop, you will get a scan on each iteration. Each column holds the data of a single channel. The upper array index selects the scan and the lower index selects the channel.

retrieval complete is TRUE when the acquisition finishes and no backlog data remains.

AI Single Scan

Returns one scan of data from a previously configured group of channels.



If you have already started an acquisition with the AI Start VI, this VI reads one scan from the acquisition buffer data, or the onboard FIFO if the acquisition is not buffered. If you have not started an acquisition, this VI starts an acquisition, retrieves a scan of data, and then terminates the acquisition. The group configuration determines the channels the VI samples.

taskID in identifies the group and the I/O operation.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

opcode specifies the type of data retrieval the VI performs.

0: Do not change the **opcode** setting (default input).
1: Read oldest data (default setting).
2: Read newest data.
3: Set the value of the **data remaining** indicator only (no data is read).
4: (Macintosh and Windows) Empty the FIFO only (for timed, nonbuffered acquisition only). No data is returned.

If you do not call the AI Start VI, this VI initiates a single scan using the fastest safe channel clock rate. You can alter the channel clock rate with the AI Config VI. Refer to Chapter 9, *Letting an Outside Source Control Your Acquisition Rate*, of the *LabVIEW Data Acquisition Basics Manual* for an explanation of how LabVIEW selects the channel clock rate.

If you run the AI Start VI, a clock signal initiates the scans.

You must use the AI Start VI to set the clock source to external, for externally-clocked

conversions.

(Macintosh and Windows) If clock sources are internal and you do not allocate memory, a timed nonbuffered acquisition begins when you run the AI Start VI. You use this type of acquisition for synchronizing analog inputs and outputs in a point-to-point control application. The following devices do not support timed, nonbuffered acquisitions.

- (Macintosh) NB-A2000, NB-A2100, and NB-A2150
- (Windows) AT-DSP2200, EISA-A2000, and AT-A2150

Note: (Macintosh and Windows) LabVIEW restarts the device in the event of a FIFO overflow during a timed, nonbuffered acquisition.

(Macintosh and Windows) When you set **opcode** to 1 for a nonbuffered acquisition, the VI reads one scan from the FIFO and returns the data. If **opcode** is 2, the VI reads the FIFO until it is empty and returns the last scan read.

time limit in sec is the maximum length of time this VI waits for the requested data to be acquired. If the time expires, the VI returns a timeout error (-10800) in the **error out** cluster. LabVIEW expresses **time limit in sec** in seconds. This parameter defaults to -1, which means LabVIEW calculates the time limit based on the acquisition clocks or to uses 1 second for external timing. The resolution of the timeout clock is about 17 ms (Macintosh) , 55 ms (Windows) , 1 ms (Sun).

output units specifies whether the VI returns unscaled binary data or volts.

0: Do not change the **output units** setting (default input).
1: Return voltage data only (default setting).
2: Return binary data only.
3: Return voltage and binary data.

taskID out has the same value as **taskID in**.

voltage data is a 2D array that contains analog data. (The AI Read One Scan, AI Single Scan, AO Write One Update, and AO Single Update VIs return only a 1D array in the **voltage data** parameter.) This array is scan-ordered which means that each row contains the data of a single scan or value. If you want to index the array in a loop, you will get a scan on each iteration. Each column holds the data of a single channel. The upper array index selects the scan and the lower index selects the channel. Refer to the Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, in *the LabVIEW Data Acquisition Basics Manual* for more information about the array format.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

data remaining is 1 when data is still in the FIFO or buffer and the VI is about to return. The **data remaining** parameter is 0 when the FIFO is empty.

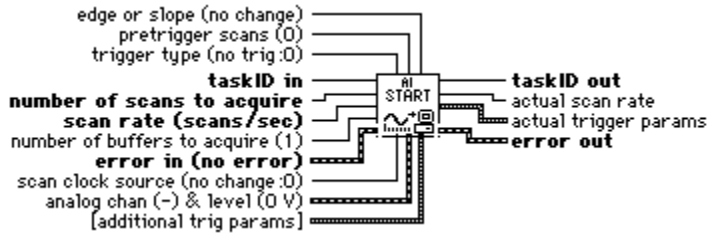
binary data contains the unscaled binary data the ADC produces.

acquisition state.

0: Running.
1: Finished with backlog.
2: Finished with no backlog.
3: Paused.
4: No acquisition.

AI Start

Starts a buffered analog input operation. This VI sets the scan rate, the number of scans to acquire, and the trigger conditions. The VI then starts an acquisition.



This VI calls several advanced analog input VIs to start the buffered analog input acquisition. The AI Start VI calls the AI Clock Config VI to configure the acquisition for the specified **scan clock source** (internal by default) and the specified **scan rate**.

The AI Start VI then calls the AI Trigger Config VI to set a trigger according to **trigger type**. If **trigger type** is 0, the AI Start VI then clears the trigger; otherwise it sets the specified type of trigger. The **analog chan and level** parameter determines the source and level conditions for an analog trigger, and **edge or slope** determines whether to look for a leading or trailing edge for digital triggers, or a leading or trailing slope for analog triggers.

Finally, the AI Start VI calls the AI Control VI to start the acquisition, using **total scans, pretrigger scans**, and **number of buffers to acquire** to determine how much data to acquire.

Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the channel ranges, input limits, scanning order, triggers, clocks and you can use with your National Instruments DAQ device.

I32 **taskID in** identifies the group and the I/O operation.

I32 **number of scans to acquire** is the total number of scans LabVIEW acquire before the acquisition completes. A scan is one point per channel. When the default input is -1, LabVIEW acquires exactly one buffer of data. The **buffer size** input to the AI Config VI determines the size of the buffer. The number of **total scans** includes any pretrigger scans requested. If you set **number of scans to acquire** to 0, LabVIEW acquires data indefinitely into the first (or only) buffer until you clear the acquisition with the AI Clear VI. In this case, the VI ignores the **pretrigger scans** input.

I32 **scan rate** is the number of scans/s to acquire. This is equivalent to the sampling rate per channel. This VI defaults to an input of 1000 scans/s. Refer to the description of the AI Clock Config VI for more options with alternate clock sources.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** Cluster Parameters.

I32 **edge or slope.**

0: Do not change the default setting (default input).
1: Leading edge for digital trigger; positive slope for analog trigger.
2: Trailing edge for digital trigger; negative slope for analog trigger.

I32 **pretrigger scans** is the number of scans you want to save in the buffer before the trigger. This parameter defaults to 0, which means LabVIEW does not save any data before the trigger.

I32 **trigger type**. Refer to the description of the AI Trigger Config VI for more details about trigger inputs.

0: Do not change the **trigger type** setting (default input).
1: Analog trigger (default setting).
2: Digital trigger A.
3: Digital trigger A and B.
4: Scan clock gating.

I32 **number of buffers to acquire** should be changed only when you allocate more than one buffer

with the AI Config or AI Buffer Config VIs for multibuffered acquisition. If **number of buffers to acquire** is 0, LabVIEW acquires data buffers indefinitely. Otherwise, LabVIEW acquires only the specified number of buffers. The default input is 1.

scan clock source identifies what clock controls the scan rate.

0: Do not change the **clock source** setting (default input).
1: An internal timebase is used (default setting).
2: You supply a signal through the I/O connector.
3: (Macintosh and Windows) Another device supplies the signal. Use the RTSI Control VI to make the connection.

analog chan and level contains the following parameters.

trigger channel is valid only when **trigger type** is 1. The **trigger channel** parameter is the analog channel that is the source of the trigger and it follows the syntax described in the *Channel, Port, and Counter Addressing* section of Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, in the *LabVIEW Data Acquisition Basics Manual*, except for the following cases.

- An empty string tells LabVIEW not to change the trigger source setting. An empty string is the default input.
- EXTx denotes an external analog input trigger, where x is the channel number. Currently, x must be 0. You can substitute the string EXTERNAL for EXT.

level is the voltage value the analog source must cross for a trigger to occur. You must also specify whether **level** must be crossed on a leading or trailing slope with the **edge or slope** input. The default input for **level** is 0.0 V.

additional trig params cluster contains the following parameters.

hysteresis is expressed in volts. The default input and setting for this parameter are 0.0 V.

coupling.

0: Do not change the **trigger coupling** setting (default input).
1: DC.
2: AC.

The default setting depends on the device you use.

delay is expressed in seconds. The default input and setting are 0.0 s (no delay).

skip count is the number of triggers the VI skips before triggering the acquisition. The default input is -1, which tells LabVIEW not to change the **skip count** setting. The default setting is 0, which tells the VI not to skip any triggers.

time limit restricts the amount of time LabVIEW waits for the trigger to occur. The default input is -1.0, which tells LabVIEW not to change the **time limit** setting. The default setting is 1.0 s.

taskID out has the same value as **taskID in**.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

actual scan rate may differ slightly from the requested scan rate, depending on the hardware capabilities.

actual trigger params cluster may differ slightly from the requested trigger inputs, depending on the hardware capabilities. It contains the following parameters.

actual level is the analog trigger level the VI used.

actual hysteresis is the hysteresis the VI used.

actual delay is the delay the VI used.

Error Handling

LabVIEW makes error handling easy with the Intermediate Analog Input VIs. Each Intermediate VI has an **error in** input cluster and an **error out** output cluster. The clusters contain a Boolean that indicates whether an error occurred, the error code for the error, and the name of the VI that returned the error. If **error in** indicates an error, the VI returns the error information in **error out** and does not continue to run.

Note: **The AI Clear VI always clears the acquisition regardless of whether error in indicates that an error occurred**.

When you use any of the Intermediate Analog Input VIs in a While Loop, you should stop the loop if the **status** in the **error out** cluster reads TRUE. If you wire the error cluster to the General Error Handler VI, the VI deciphers the error information and describes the error to you. The General Handler VI is in **Functions**»**Utilities** in LabVIEW.

Other Intermediate-Level VIs

There are three other intermediate-level VIs called the Analog Input Utility VIs. These VIs, which are located in the Analog Input Utility subpalette, have the same functionality as the Intermediate VIs discussed in this section.

error in Cluster Parameters

status is TRUE if an error occurred. If **status** is TRUE, the VI does not perform any operations.

code is the error code number identifying an error. A value of 0 means no error, a negative value means a fatal error, and a positive value is a warning. Refer to the Data Acquistion VI Error Codes for a code description.

source identifies where an error occurred. The **source** string is usually the name of the VI that produced the error.

input limits Cluster Parameters

high limit is the highest expected voltage level of the signals you want to measure. The default input is 10 V (5 V for Lab and 1200 Series devices). LabVIEW uses this value to compute the gain.

low limit is the lowest expected voltage level of the signals you want to measure. The default input is -10 V (-5 V for Lab and 1200 Series devices). LabVIEW uses this value to compute the gain.

AI Clear VI

AI Clear

AI Config VI

AI Config

AI Read VI

AI Read

AI Single Scan VI

AI Single Scan

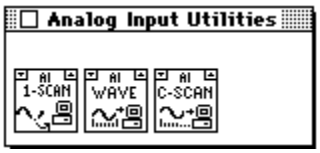
AI Start VI

AI Start

Analog Input Utility VIs

The Analog Input Utility VIs are single-VI solutions to common analog input problems. The Analog Input Utility VIs are intermediate-level VIs, so they rely on the advanced-level VIs. Refer to the Advanced Analog Input VIs section for additional information on the inputs and outputs and how they work.

You can access the **Analog Input Utilities** palette by choosing **Functions»Data Acquisition»Analog Input»Analog Input Utilities**. The icon that you must select to access the Analog Input Utility VIs is on the bottom row of the **Analog Input** palette, as shown below.



For examples of how to use the Analog Input Utility VIs, open the example library by opening `examples\daq \anlogin.llb`.

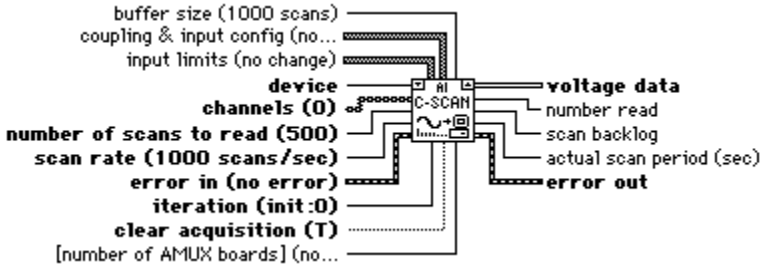
Utility VIs

AI Continuous Scan
AI Read One Scan
AI Waveform Scan

Error Handling

AI Continuous Scan

Makes continuous, time-sampled measurements of a group of channels, stores the data in a circular buffer, and returns a specified number of scan measurements on each call.



The AI Continuous Scan VI scans a group of channels indefinitely, as you might do in data logging applications. Place the VI in a While Loop and wire the loops iteration terminal to the VI **iteration** input.

Note: If your program iterates more than 231-1 times, do not wire the iteration input to the loop iteration terminal. Instead, set iteration to 0 on the first loop, then to any positive value on all other iterations. The VI reconfigures and restarts if iteration 0.

Also wire the condition that terminates the loop to the **clear acquisition** input, inverting the signal if necessary so that it reads TRUE on the last iteration. On iteration 0, the VI calls the AI Config VI to configure the channel group and hardware and allocates a data buffer; the VI calls the AI Start VI to set the scan rate and start the acquisition. On each iteration, the VI calls the AI Read VI to retrieve the number of measurements specified by **number of scans to read**, scales them, and returns the data as an array of voltages. On the last iteration (when **clear acquisition** is TRUE) or if an error occurs, the VI calls the AI Clear VI to clear any acquisition in progress. You should not need to call the AI Continuous Scan VI outside of a loop, but if you do, you can leave the **iteration** and **clear acquisition** inputs unwired.

When calling the AI Continuous Scan VI in a loop to read portions of the data from the ongoing acquisition, you must read the data fast enough so that newly acquired data does not overwrite it. The **scan backlog** output tells you how much data acquired by the VI, but remains unread. If the backlog increases steadily, your new data may eventually overwrite old data. Retrieve data more often, or adjust the **buffer size**, the **scan rate**, or the **number of scans to read** to fix this problem

Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual* for the channel ranges, input limits, and scanning order you can use with your National Instruments DAQ device.

Note: The AI Continuous Scan VI uses an uninitialized shift register as local memory to remember the taskID of the acquisition between VI calls. You normally use this VI only once on a diagram, but if you use it in multiple places, all the instances share the same taskID. All calls to this VI configure, read data from, or clear the same acquisition. Occasionally, you may want to use this VI in multiple places and have each instance refer to a different taskID (for instance, when you measure two devices simultaneously). Save a copy of this VI with a new name (for example, AI Continuous Scan R) and make your new VI reentrant. See the topic Reentrant Execution for information on creating a VI reentrant.


device is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.


channels is a list of the analog channels you want to use. If `x`, `y`, and `z` refer to channels, you can specify a list of channels by separating the individual channels with commasfor example, `x,y,z`. If `x` refers to the first channel in a consecutive channel range and `y` refers to the last channel, you can specify the range by separating the first and last channels by a colonfor example, `x:y`. Refer to the *Channel Addressing with the AMUX-64T* section of Chapter 5, *Things You Should Know about Analog Input*, in the *LabVIEW Data Acquisition Basics Manual*, for the syntax you use for AMUX-64T channels or SCXI.


number of scans to read is the number of scans the VI retrieves from the ongoing acquisitions buffer each time. This is equivalent to the number of points per channel. If the VI times out, it returns whatever number of scan is available.


scan rate is the number of scans per second acquired by the VI. This is equivalent to the sampling rate per channel. You cannot change the channel clock rate (or interchannel delay) for the VI. You must use the advanced AI Clock Config VI if you are dissatisfied with the interchannel delay LabVIEW chooses.


error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.


iteration can be used to optimize operation when you execute this VI in a loop. When **iteration** is 0 (default), LabVIEW calls the AI Config and AI Start VIs to configure the channel. If **iteration** is greater than 0, LabVIEW uses the existing configuration, which improves performance. You usually wire this input to an iteration terminal.


clear acquisition determines whether the VI clears the task after reading the specified number of scans. You should pass a value of `TRUE` for this parameter when reading the last set of scans for a given acquisition. The default is `TRUE`, which means that if you leave this input unwired, the VI reads data only once. You normally wire this input to the terminating condition of a loop, so that when the loop finishes, the VI clears the acquisition.


buffer size is the number of scans (with input operations) or updates (with output operations) you want the circular buffer to hold.


coupling & input config is an array of clusters. Each array element contains the configuration for the channel or channels specified by the corresponding element of the **channels** array. If there are fewer elements in this array than in the **channels** array, the VI uses the last array element for the rest of the channels. The **coupling & input config** array defaults to an empty array, which means the

parameters keep their default settings. Each cluster contains the following parameters.

I32 **coupling**.

0: Do not change the **coupling** setting.
1: DC.
2: AC.
3: Ground.
4: Internal reference.

I32 **input config**.

0: Do not change the **input config** setting.
1: Differential (default).
2: Referenced single-ended.
3: Nonreferenced single-ended.

I32 **input limits** is an array of clusters. Each array element contains the expected signal limits for the channels specified by the corresponding element of **channels**. If there are fewer elements in this array than in **channels**, the VI uses the last array element for the rest of the channels. The **input limits** array defaults to an empty array, which means the **input limits** keep their default settings. Click here to see the **input limits** cluster parameters.

I32 **number of AMUX boards** is the number of AMUX-64T boards attached (the default value is 0).

0: No AMUX-64T boards.
1: One AMUX-64T board.
2: Two AMUX-64T boards.
4: Four AMUX-64T boards.

I32 **voltage data** is a 2D array that contains analog data. (The AI Read One Scan, AI Single Scan, AO Write One Update, and AO Single Update VIs return only a 1D array in the **voltage data** parameter.) This array is scan-ordered which means that each row contains the data of a single scan or value. If you want to index the array in a loop, you will get a scan on each iteration. Each column holds the data of a single channel. The upper array index selects the scan and the lower index selects the channel. Refer to the Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, in *the LabVIEW Data Acquisition Basics Manual* for more information about the array format

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

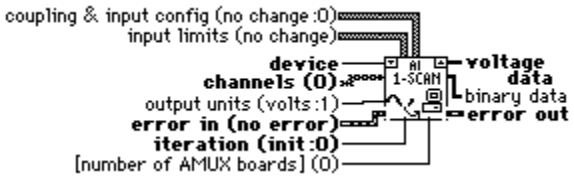
I32 **number read** is the number of scans returned. This parameter is the same as **number of scans to read** unless an error or timeout occurs or the VI reaches the end of the data.

I32 **scan backlog** is the amount of data remaining in the buffer after this VI completes. If **scan backlog** increases steadily, you are not reading data fast enough to keep up with the acquisition, and your newly-acquired data may overwrite unread data and give you an overwrite error. Decrease the **scan rate**, increase the **number of scans to read**, read the scans more often, or increase the **buffer size**. See the description of the AI Buffer Read VI for a precise definition of how to compute **scan backlog**.

I32 **actual scan period** is the actual period between scans, the inverse of the actual scan rate the VI used to acquire the data. This may differ slightly from the requested scan rate, depending on the hardware capabilities.

AI Read One Scan

Measures the signals on the specified channels and returns the measurements in an array of voltages or binary values.



The AI Read One Scan VI performs an immediate measurement of a group of one or more channels. If you place the VI in a loop to take multiple measurements from a group of channels, wire the loop iteration terminal to the VI **iteration** parameter.

Note: If your program iterates more than 231-1 times, do not wire the iteration parameter to the loop iteration terminal. Instead, set iteration to 0 on the first loop, then to any positive value on all other iterations. The VI reconfigures and restarts if iteration 0.

On iteration 0, this VI calls the AI Config VI to configure the channel group and hardware, then calls the AI Single Scan VI to measure and report the results. On subsequent iterations, the VI avoids unnecessary configuration and calls only the AI Single Scan VI. If you call the AI Read One Scan VI once to take a single measurement from the group of channels, the **iteration** parameter can remain unwired.

Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual,* for the channel ranges, input limits, and scanning order available with your DAQ device.

Note: The AI Read One Scan VI uses an uninitialized shift register as local memory to remember the taskID for the group of channels between VI calls. You normally use this VI in one place on your diagram, but if you use it more than once, the multiple instances of the VI share the same taskID. All the calls to this VI configure or read data from the same group. Occasionally, you may want to use this VI in multiple places on the diagram, but have each instance refer to a different taskID (for example, when you want to measure two devices simultaneously). Save a copy of this VI with a new name (for example, AI Read One Scan R) and make your new VI reentrant. See Chapter 19, VI Setup Options, in the LabVIEW User Manual for information on creating VI reentrants.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **channels** is a list of the analog channels you want to use. If `x`, `y`, and `z` refer to channels, you can specify a list of channels by separating the individual channels with commasfor example, `x,y,z`. If `x` refers to the first channel in a consecutive channel range and `y` refers to the last channel, you can specify the range by separating the first and last channels by a colonfor example, `x:y`. Refer to the *Channel Addressing with the AMUX-64T* section of Chapter 5, *Things You Should Know about Analog Input*, in the *LabVIEW Data Acquisition Basics Manual*, for the syntax you use for AMUX-64T channels or SCXI.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **iteration** can be used to optimize operation when you execute this VI in a loop. When **iteration** is 0 (default), LabVIEW calls the AI Config VI to configure the channel. If **iteration** is greater than 0, LabVIEW uses the existing configuration, which improves performance. You usually wire this input to an iteration terminal.

I32 **coupling & input config** is an array of clusters. Each array element specifies the configuration for the channels specified by the corresponding element of **channels**. If there are fewer elements in this array than in **channels**, the VI uses the last element of **coupling & input config** for the rest of the channels. This cluster defaults to an empty array, which means the parameters remain unchanged from the default settings. Each cluster contains the following parameters.

I32 **coupling**. Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for information about coupling on your specific type of DAQ device.

0: Do not change the **coupling** setting.
1: DC.
2: AC.
3: Ground.
4: Internal reference.

I32 **input config**.

0: Do not change the **input config** setting.
1: Differential (default).
2: Referenced single-ended.
3: Nonreferenced single-ended.

I32 **input limits** is an array of clusters. Each array element contains the expected signal limits for the channels specified by the corresponding element of **channels**. If there are fewer elements in this array than in **channels**, the VI uses the last array element for the rest of the channels. The **input limits** array defaults to an empty array, which means the **input limits** keep their default settings. Click here to see the **input limits** cluster parameters.

I32 **output units** specifies whether the VI returns unscaled binary data or volts.

0: Do not change the **output units** setting (default input).
1: Return voltage data only (default setting). The **binary data** output array is empty.
2: Return binary data only. The **voltage data** output array is empty. The VI executes slightly faster if you select this value, because the VI does not perform scaling.
3: Return voltage and binary data.

I32 **number of AMUX boards** is the number of AMUX-64T boards attached (the default value is 0).

0: No AMUX-64T boards.
1: (Macintosh and Windows) One AMUX-64T board.
2: (Macintosh and Windows) Two AMUX-64T boards.
4: (Macintosh and Windows) Four AMUX-64T boards.

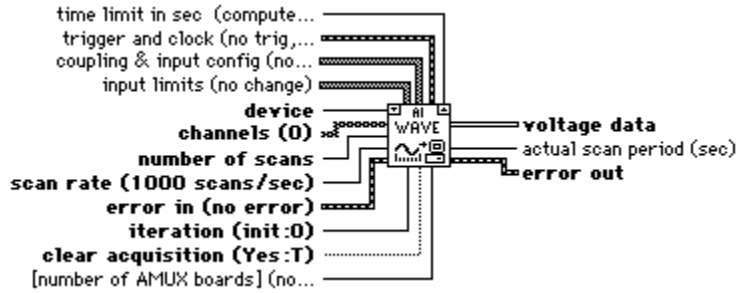
I32 **voltage data** is a 2D array that contains analog data. (The AI Read One Scan, AI Single Scan, AO Write One Update, and AO Single Update VIs return only a 1D array in the **voltage data** parameter.) This array is scan-ordered which means that each row contains the data of a single scan or value. If you want to index the array in a loop, you will get a scan on each iteration. Each column holds the data of a single channel. The upper array index selects the scan and the lower index selects the channel. Refer to the Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, in *the LabVIEW Data Acquisition Basics Manual* for more information about the array format.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

I32 **binary data** is a 2D array that contains analog input data. This array is scan-ordered which means that each row contains the data of a single scan. If you want to index the array in a loop, you will get a scan on each iteration. Each column holds the data of a single channel. The upper array index selects the scan and the lower index selects the channel.

AI Waveform Scan

Acquires the specified number of scans at the specified scan rate and returns all the data acquired. You can trigger the acquisition.



The AI Waveform Scan VI acquires a specified number of scans from a channel group at a specified scan

rate. If you place this VI in a loop to take multiple acquisitions from the same group of channels, wire the iteration terminal of the loop to the VI **iteration** input.

Note: If your program iterates more than 231-1 times, do not wire the iteration parameter to the loop iteration terminal. Instead, set iteration to 0 on the first loop, then to any positive value on all other iterations. The VI reconfigures and restarts if iteration 0.

Also wire the condition that terminates the loop to the VI **clear acquisition** input, inverting the signal if necessary so that it reads TRUE on the last iteration. On iteration zero, this VI calls the AI Config VI to configure the channel group and hardware and allocate a data buffer. On each iteration, this VI calls the AI Start and AI Read VIs. The AI Start VI sets the scan rate and trigger conditions and starts the acquisition. The VI stores the measurements in the buffer as they are acquired, and the AI Read VI retrieves them from the buffer, scales them, and returns all the data as an array of voltages. On the last iteration (when **clear acquisition** is TRUE) or if an error occurs, the VI also calls the AI Clear VI to clear the acquisition in progress. If you call the AI Waveform Scan VI only once, you can leave **iteration** and **clear acquisition** unwired.

Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the channel ranges, input limits, scanning order, triggers, and clocks you can use with your National Instruments DAQ device.

Note: The AI Waveform Scan VI uses an uninitialized shift register as local memory to remember the taskID for the group of channels between VI calls. You normally use this VI in one place on your diagram, but if you use it more than once, the multiple instances of the VI share the same taskID. All calls to this VI configure, read data from, or clear the same acquisition. Occasionally you may want to use this VI in multiple places and have each instance refer to a different taskID (for example, when you measure two devices simultaneously). Save a copy of this VI with a new name (for example, AI Waveform Scan R) and make your new VI reentrant. See the topic Reentrant Execution for information on how to make a VI reentrant.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **channels** is a list of the analog channels you want to use. If `x`, `y`, and `z` refer to channels, you can specify a list of channels by separating the individual channels with commasfor example, `x,y,z`. If `x` refers to the first channel in a consecutive channel range and `y` refers to the last channel, you can specify the range by separating the first and last channels by a colonfor example, `x:y`. Refer to the *Channel Addressing with the AMUX-64T* section of Chapter 5, *Things You Should Know about Analog Input*, in the *LabVIEW Data Acquisition Basics Manual*, for the syntax you use for AMUX-64T channels or SCXI.

I32 **number of scans** is the number of scans the VI acquires before the acquisition completes. This is equivalent to the number of points per channel.

I32 **scan rate** is the number of scans per second the VI acquires. This is equivalent to the sampling rate per channel. You cannot change the channel clock rate (or interchannel delay). You must use the advanced AI Clock Config VI if you are dissatisfied with the interchannel delay LabVIEW chooses.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **iteration** can be used to optimize operation when you execute this VI in a loop. When **iteration** is 0 (default), LabVIEW calls the AI Config VI to configure the channel. If **iteration** is greater than 0, LabVIEW uses the existing configuration, which improves performance. You usually wire this input to an iteration terminal.

I32 **clear acquisition** determines whether the VI clears the task after reading the specified number of

scans. You should pass a value of TRUE for this parameter when reading the last set of scans for a given acquisition. The default is TRUE, which means that if you leave this input unwired, the VI reads data only once. You normally wire this input to the terminating condition of a loop, so that when the loop finishes, the VI clears the acquisition.

time limit in sec is the maximum length of time this VI waits for the requested data to be acquired. If the time expires, the VI returns a timeout error (-10800) in the **error out** cluster. LabVIEW expresses **time limit in sec** in seconds. This parameter defaults to -1, which means LabVIEW calculates the time limit based on the acquisition clocks or to uses 1 s for external timing. The resolution of the timeout clock is about 17 ms (Macintosh) , 55 ms (Windows) , 1 ms (Sun) .

trigger and clock contains the following parameters. Refer to the description of the AI Trigger Config VI for more information about trigger inputs.

trigger type.

0: No trigger (default input).
1: Analog trigger.
2: Digital trigger A.
3: Digital trigger B.
4: Scan clock gating.
5: Software analog.

pretrigger scans is the number of scans you want the VI to save in the buffer before the trigger. The default input is 0, which tells LabVIEW to acquire all data after the trigger.

edge or slope.

0: Do not change the **edge or slope** setting (default input).
1: Leading edge for digital trigger or positive slope for analog trigger.
2: Trailing edge for digital trigger or negative slope for analog trigger.

analog chan and level contains the following parameters, which you use only when the trigger type is analog.

analog chan is the analog channel that is the source of the trigger. You should use the syntax described in *the Channel, Port, and Counter Addressing* section of Chapter 3*, Basic LabVIEW Data Acquisition Concepts*, in the *LabVIEW Data Acquisition Basics Manual*, with the following exceptions.

An empty string tells LabVIEW not to change the trigger source setting (default input).

EXTx denotes an external analog input trigger, where x is the channel number. Currently, x must be 0. You can substitute EXTERNAL for EXT.

level is the voltage value the analog source must cross for a trigger to occur. You must also specify whether the level must be crossed on a leading or trailing slope using the **edge or slope** input. The default input for **level** is 0.0 V.

scan clock source.

0: Do not change the **clock source** setting (default input).
1: Internal (default setting).
2: I/O connector.
3: (Macintosh and Windows) RTSI connection.

coupling & input config is an array of clusters. Each array element assigns the configuration for the channels specified by the corresponding element of **channels**. If there are fewer elements in this array than in **channels**, the VI uses the last element of **coupling & input config** for the rest of the channels. The default input for **coupling & input config** is an empty array, which means the parameters keep their default settings. Each cluster contains the following parameters.

coupling. Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual,* for information about coupling on your specific type of DAQ device.

0: Do not change the **coupling** setting.
1: DC.
2: AC.
3: Ground.
4: Internal reference.

I32 **input config**.

0: Do not change the **input config** setting.
1: Differential (default).
2: Referenced single-ended.
3: Nonreferenced single-ended.

I32 **input limits** is an array of clusters. Each array element contains the expected signal limits for the channels specified by the corresponding element of **channels**. If there are fewer elements in this array than in **channels**, the VI uses the last array element for the rest of the channels. The **input limits** array defaults to an empty array, which means the **input limits** keep their default settings. Click here to see the **input limits** cluster parameters.

I32 **number of AMUX boards** is the number of AMUX-64T boards attached (the default value is 0).

0: No AMUX-64T boards.
1: One AMUX-64T board.
2: Two AMUX-64T boards.
4: Four AMUX-64T boards.

I32 **voltage data** is a 2D array that contains analog data. (The AI Read One Scan, AI Single Scan, , AO Write One Update, and AO Single Update VIs VIs return only a 1D array in the **voltage data** parameter.) This array is scan-ordered which means that each row contains the data of a single scan or value. If you want to index the array in a loop, you will get a scan on each iteration. Each column holds the data of a single channel. The upper array index selects the scan and the lower index selects the channel. Refer to the Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, in *the LabVIEW Data Acquisition Basics Manual* for more information about the array format

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

I32 **actual scan period** is the interval between scans, the inverse of the actual scan rate the VI used to acquire the data. This may differ slightly from the requested scan rate, depending on the hardware capabilities.

AI Continuous Scan VI

AI Continuous Scan

AI Read One Scan VI

AI Read One Scan

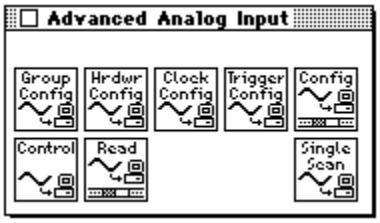
AI Waveform Scan VI

AI Waveform Scan

Advanced Analog Input VIs

This chapter contains reference descriptions of the Advanced Analog Input VIs.

You can access the Advanced Analog Input palette by choosing **Functions**»**Data Acquisition**»**Analog Input**»**Advanced Analog Input**. The icon that you must select to access the Advanced Analog Input VIs is on the bottom row of the **Analog Input** palette. The **Analog Input** palette is shown below. Click on an icon in this illustration to go to a particular Advanced Analog Input VI.



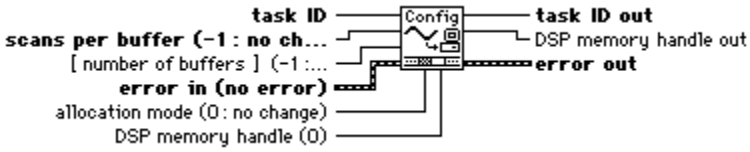
For examples of how to use the Advanced Analog Input VIs, open the example library by opening `examples\daq \anlogin.llb`.

Advanced VIs

AI Buffer Config
AI Buffer Read
AI Clock Config
AI Control
AI Group Config
AI Hardware Config
AI SingleScan
AI Trigger Config

AI Buffer Config

Allocates memory for LabVIEW to store analog input data until the AI Buffer Read VI can deliver it to you. LabVIEW refers to the buffer(s) allocated by the AI Buffer Config VI as internal buffers because you do not have direct access to them.



I32 **taskID in** identifies the group and the I/O operation.

I32 **scans per buffer** is the number of scans each buffer can hold. This parameter defaults to -1, which tells LabVIEW to leave the **scans per buffer** setting unchanged. The default setting is 100 scans.

(Macintosh) If you are using DMA, the total buffer size cannot be greater than 2^24 - 1, which is 16,777,215 bytes. Remember that each scan holds a sample from each channel in the group, and each sample is two bytes.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.error in Cluster Parameters

I32 **number of buffers** is the number of buffers the VI allocates. LabVIEW numbers individual buffers buffer 1, buffer 2, and so on in the AI Buffer Read VI. This parameter defaults to -1, which tells LabVIEW to leave the **number of buffers** setting unchanged. The default setting is 1 buffer.

I32 **allocation mode** determines how LabVIEW allocates the buffer.

0: Do not change the **allocation mode** setting (default input).
1: Do not allocate memory, ignore scans per buffer and number of buffers, and deallocate any existing memory.
2: Allocate memory on the host computer (default setting).
3: (Windows) Allocate DSP memory and return the handle in **DSP memory handle out**.
4: (Windows) Use the DSP memory that another VI previously allocated. **DSP memory handle** points to this memory.

Note: (Macintosh) If you are using an NB-A2000 with an NB-DMA2800, buffer size and total scans to acquire are both multiples of 32, and your computer has block-mode memory, the driver will automatically use block-mode DMA transfers.

I32 **DSP memory handle**. You use **DSP memory handle** when **allocation mode** is 4. *LabVIEW for Macintosh and Sun do not currently support this parameter.*

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

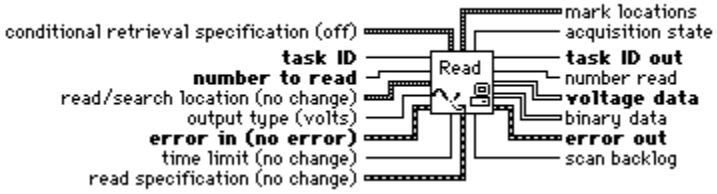
I32 **DSP memory handle out** is the handle the VI produces when **allocation mode** is 3, or the **DSP memory handle** you pass to the VI when **allocation mode** is 4. *LabVIEW for Macintosh and Sun do not currently support this parameter.*

Note: When you run the AI Control VI with control code set to 4 (clear), the VI performs the equivalent of running the AI Buffer Config VI with allocation mode set to 1. That is, both VIs deallocate the internal analog input data buffers. However, acquisitions that use DSP or expansion card memory are an exception. The AI Control VI does not deallocate DSP memory when clearing an acquisition. You must explicitly call the AI Buffer Config VI to deallocate DSP acquisition buffers.

AI Buffer Config VI Device-Specific Settings and Ranges is a table, listing the default settings and ranges for the AI Buffer Config VI. The first row gives the values for most devices, and the other rows give the values for devices that are exceptions to the rule.

AI Buffer Read

Returns analog input data from the internal data buffer(s).



I32 **taskID in** identifies the group and the I/O operation.

I32 **number to read**. When **read units** (a parameter of the **read specification** cluster) is 1, **number to read** specifies the number of scans. When **read units** is 2, **number to read** specifies the number of buffers to read. This parameter defaults to -1, which tells LabVIEW to leave the **number to read** setting unchanged. The default setting is equal to the **total scans to acquire** you pass to the AI Control VI. The default setting is 100 if **total scans to acquire** is 0. For more information, refer to the **read specification** parameter description.

Device-Specific Settings and Ranges for the number to read Parameter

Device	Default Setting	Range

All Devices	100	*n*>=0

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

conditional retrieval specification. Conditional retrieval is an artificial form of triggering. It is like a software trigger, but more versatile. Conditional retrieval is particularly useful for devices that do not offer analog triggering. Conditional retrievals, unlike actual triggers, have no effect on the actual data acquisition and you can search an acquired buffer repeatedly for the trigger condition.

Conditional retrieval allows you to specify trigger conditions for which the VI searches within the acquisition buffer. The search begins at the location specified by **read/search location**. After the VI finds the retrieval conditions in the buffer, it returns data starting at this location plus the offset. The search does not cross buffer boundaries for multiple buffer acquisitions.

To perform conditional retrieval, you must set **read units** (a parameter of the **read specification** cluster) to 1.

The **conditional retrieval specification** cluster contains the following parameters.

mode enables or disables conditional retrieval.

0: Do not change the mode setting (default input).
1: Off (default setting)
2: On.

channel index specifies which channels data the VI searches to find the retrieval conditions. You specify the channel using an index in the scan list. Use the Channel To Index VI to produce this index. This parameter defaults to -1, which tells LabVIEW to leave **channel index** unchanged. The default setting is 0.

slope tells the VI whether to search for a rising or falling voltage level.

0: Do not change the **slope** setting (default input).
1: Rising (default setting).
2: Falling.

level specifies the voltage level for which the VI searches. You express **level** in volts. The default input and setting are 0.0 V.

hysteresis specifies the voltage window from which the signal must leave to meet the retrieval conditions. That is, **hysteresis** works like a noise filter. For example, if you set **slope** to 1 (rising), **level** to 1.0 V, and **hysteresis** to 0.1 V, the signal must drop below 0.9 V before a retrieval can occur. If you change the **slope** to 2 (falling), the signal has to rise above 1.1 V before a retrieval can occur. You express **hysteresis** in volts. The default input and setting are 0.0 V.

skip count is the number of retrieval conditions (triggers) the VI skips before the final retrieval condition can occur. This parameter defaults to -1, which tells LabVIEW to leave the **skip count** setting unchanged. The parameter defaults to a setting of 0, which means the VI does not skip any triggers.

offset indicates the scan location from which the VI begins reading data relative to the retrieval condition. A negative number indicates data prior to the retrieval condition (pretrigger data), and a positive number indicates data after the retrieval condition (posttrigger data). The default setting is 0.

Device-Specific Settings and Ranges for the conditional retrieval specifications Cluster (Part 1)

	mode		**channel index**		**slope**		**level**	
Device	**DS***	**R***	**DS***	**R***	**DS***	**R***	**DS***	**R***

All Devices	1	1, 2	0	*n*>=0	1	1, 2	0.0	any	

**D S = Default Setting; R = Range*

Device-Specific Settings and Ranges for the conditional retrieval specifications Cluster (Part 2)

	hysteresis		**skip count**		**offset**	
Device	**DS***	**R***	**DS***	**R***	**DS***	**R***
All Devices	0.0	any	0.0	any	0.0	any

**D S = Default Setting; R = Range*

I32 read/search location contains the following parameters. When the **mode** parameter in the **conditional retrieval specification** cluster is off, conditional retrieval is disabled. This means **read/search location** specifies the acquisition buffer starting point from which to read data. When **mode** is on, LabVIEW enables conditional retrieval, and **read/search location** specifies where in the acquisition buffer to begin the search.

I32 **read/search mode** indicates which reference mark within an input buffer provides the starting point for the read or search. This reference can be the read mark, the start of the buffer, the most recently acquired data (end of data), or the trigger point.

0: Do not change the **read/search mode** setting (default input).
1: Relative to the read mark (default setting).
2: Relative to the start of the buffer.
3: Relative to the end of the data.
4: Relative to the trigger point. This mode is valid only when the buffer is completed.

The read mark is similar to a file I/O pointer in that it moves every time you read data from an input buffer. Reading always begins where the read mark points at the scan. After the read, the read mark points to the next unread scan.

The start of buffer points to scan number 1 of the buffer that LabVIEW is currently reading. That is, the value output as the read mark buffer position. If **buffer number** does not equal -1, the start of buffer points to scan number 1 of the buffer specified by **buffer number**.

The end of data mark points to the most recently acquired scan.

The trigger mark points to the first scan acquired after the stop trigger in a pretriggered acquisition.

Note: **When the VI reads from the trigger mark, it does not return data until the acquisition completes for the buffer containing the trigger.**

I32 **read/search offset**. The VI adds **read/search offset** to the mark specified by **read/search mode** to determine the starting point for the read. You express **read/search offset** in scans or buffers, depending on the value of **read units**. The **read/search offset** parameter can be negative. The default setting and input is 0.

Device-Specific Settings and Ranges for the read/search location Cluster

	read/search mode		read/search offset	
Device	**DS***	**R***	**DS***	**R***
PC-LPM-16 DAQCard-500 DAQCard-700	1	1<=*n*<=3	0	any
All Other Devices	1	1<=*n*<=4	0	any

* *D S = Default Setting; R = Range*

I32 **output type** specifies whether the VI returns unscaled binary data or volts.

0: Do not change the **output type** setting (default input).
1: Return voltage data only (default setting).
2: Return binary data only.
3: Return voltage and binary data.

Device-Specific Settings and Ranges for the output type Parameter

Device	Default Setting	Range
All Devices	1	1<=*n*<=3

I32 **time limit** is the maximum length of time this VI waits for the data. You express **time limit** in seconds. If the time you specify expires, the VI returns a timeout warning (10800) in **status**. This parameter defaults to -1.0, which tells LabVIEW to calculate the timeout based on the acquisition clocks or to use 1 s if you use external timing. The resolution of the timeout clock is about 17 ms (Macintosh) , 55 ms (Windows), 1 ms (Sun).

Device-Specific Settings and Ranges for the time limit Parameter

Device	Default Setting	Range
All Devices	variable	*n*>=0

I32 **read specification** contains the following parameters.

I32 **read units** specifies whether LabVIEW interprets the **number to read** and **read offset** input parameters as scans or buffers.

0: Do not change the **read units** setting (default input).
1: Scans (default setting).
2: Buffers.

Each buffer contains a certain number of scans. When **read units** is 2 (buffers), the number of scans read in each buffer equals the value you specified in the **total scans to acquire** parameter of the AI Control VI. However, for pretriggered acquisitions, the scans returned can be less than the value of the **total scans to acquire** parameter of the AI Control VI. The following devices are exceptionsthey always return a number of scans equal to total scans to acquire.

- (Macintosh) NB-A2000
- (Windows) EISA-A2000 and AT-DSP2200

- (Sun) SB-A2200

I32 **buffer number**. You can use **buffer number** to move the read mark to a specific buffer. This parameter defaults to -1, which tells LabVIEW to leave the **read mark buffer** unchanged.

I32 **channel indices** contains the channel indices for which you want to retrieve data. If **channel indices** is empty, the VI returns the entire scan. Otherwise, the VI returns only the channels specified. Use the Channel To Index VI to produce **channel indices**.

I32 **DSP memory handle**. If you wire a valid **DSP memory handle**, the VI places the retrieved data in the DSP device memory. When the VI puts retrieved data in DSP memory**, output type** must be 1 or 2. *LabVIEW for Macintosh and Sun do not currently support this parameter.*

Device-Specific Settings and Ranges for the read specifications Cluster

	read units		**buffer number**		**channel indices**		**DSP memory handle**	
Device	**DS***	**R***	**DS***	**R***	**DS***	**R***	**DS***	**R***
AT-DSP2200	1	1, 2	read mark buffer	*n* >=1	empty	*n*>= 0	0	any
All Other Devices	1	1, 2	read mark buffer	*n* >=1	empty	*n*>= 0	no support	

**D S = Default Setting; R = Range*

I32 **taskID out** has the same value as **taskID in**.

I32 **voltage data** is a 2D array that contains scaled analog input data if **output type** is 1 or 3.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

I32 **mark locations**. The scan and buffer positions of the read and end of data marks describe the current position of these marks. If the buffer position of a mark is 2 and the scan position is 57, the mark is within buffer 2 at scan 57. The first buffer is buffer 1, and the first scan within a buffer is scan 1. The **mark locations** cluster contains the following parameters.

I32 **read mark scan** indicates the next scan to be read.

I32 **end of data scan** indicates the most recently acquired scan.

I32 **read mark buffer** is the number of the buffer where the read mark currently resides.

I32 **end of data buffer** is the number of the buffer where the end-of-data mark currently resides.

I32 **acquisition state**.

0: Running.
1: Finished with backlog.
2: Finished with no backlog.
3: Paused.
4: No acquisition.

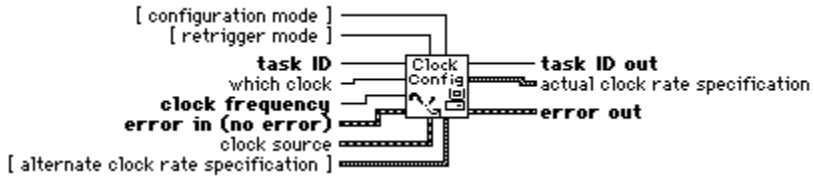
I32 **number read** is the number of scans or buffers returned in the data array. The value of **read units** determines the units for **number read**.

I32 **binary data** is a 2D array that contains unscaled analog input data if **output type** is 2 or 3.

I32 **scan backlog**. The amount of data acquired minus the amount of data read in the current buffer. If **scan backlog** continues to increase, LabVIEW is not reading data fast enough and eventually returns a buffer overwrite error and you lose data.

AI Clock Config

Sets the channel and scan clock rates.



Refer to Appendix B, *Hardware Capabilities*, in the *Data Acquisition VI Reference Manual*, for the clocks available with your DAQ device.

I32 **taskID** identifies the group and the I/O operation.

I32 **clock frequency**. You express **clock frequency** in scans per second (with the scan clock) or channels per second (with the channel clock), depending on the value of **which clock**. A value of 0 for **clock frequency** turns the clock off. The default input is -1, which tells LabVIEW to use the **alternate clock rate specification** instead of the **clock frequency** parameter. The default setting is clock- and device-specific.

Click here to see the default settings and ranges table for the scan clock.

Click here to see the settings and ranges for the channel clock.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **configuration mode** specifies when LabVIEW changes the clock rate.

0: Do not change the **configuration mode** setting (default input).
1: Change rate immediately (default setting).
2: Change rate when current buffer is finished. *LabVIEW does not currently support this option.*
3: Do not change rate. This is useful for rate translation only.

I32 **retrigger mode** dictates the initial programming of the scan and sample counters. *LabVIEW for Sun does not currently support this parameter.*

0: Do not change the **retrigger mode** setting (default input).
1: FALSE (default setting).
2: TRUE.

If **retrigger mode** is TRUE, one scan interval elapses before the VI initiates the first scan, and one channel interval elapses before the VI acquires the first sample. Because this is the state of the counters after the first buffer (while they wait for second and third triggers, and so on), this option supports retriggering.

If **retrigger mode** is FALSE, two **timebase source** edges elapse before the VI initiates the first scan and acquires the first sample.

I32 **which clock** determines the clock to which the remaining input parameters apply.

0: Do not change clocks (default input).
1: Scan clock 1 (default setting).
2: Channel clock 1.

For devices that have only a channel clock (Lab-LC, Lab-NB, NB-MIO-16, Lab-PC+, PCI-1200, PC-LPM-16, DAQCard-500, DAQCard-700, and DAQCard-1200), you cannot set independent channel and scan clock rates. Setting one resets the other because the channel rate equals scan rate/number of channels to scan.

For devices that have no channel clock (NB-A2000, NB-A2100, NB-A2150, EISA-A2000, AT-A2150, AT-DSP2200, and SB-A2200), setting the channel clock produces an error.

If you specify a value of 0 for the scan clock rate, interval scanning turns off, and channel scanning (or round-robin scanning) proceeds at the channel clock rate. This option is meaningful only for devices with independent channel and scan clocks.

 **clock source** indicates where the clock signal comes from. This cluster contains the following parameters.

 **clock source code**.

0: Do not change the clock source setting (default input).
1: Internal (default setting).
2: I/O connection.
3: RTSI connection (Non-E Series only).
4: PFI pin, low to high (E Series only).
5: PFI pin, high to low (E Series only).
6: RTSI pin, low to high (E Series only).
7: RTSI pin, high to low (E Series only).
8: GPCTR output, low to high (E Series only).
9: GPCTR output, high to low (E Series only).
10: ATCOut (analog trigger circuitry output), low to high (E Series only).
11: ATCOut (analog trigger circuitry output), high to low (E Series only).

The **clock frequency**, **clock period**, **timebase source**, **timebase divisor**, **retrigger mode**, and **configuration mode** parameters are valid only when the **clock source code** is internal.

When the **clock source code** is set to I/O connection, the VI assumes the external clock signal is connected to the OUT2 line for the scan clock (most devices) or the EXTCONV line for the channel clock (most devices). For MIO-E Series devices, the AI Clock Config VI assumes the external clock signal is connected to the PFI7 pin for the scan clock or the PFI2 pin for the channel clock. If you want to change the PFI pin, you should set the **clock source code** to 4 or 5 and specify the PFI pin number in the **clock source string**. If you select value 3, or RTSI connection, you must also use the RTSI Control VI to specify the RTSI connection for non-E Series devices. For externally clocked single channel sampling using AMUX devices, you must use the EXTCONV line.

 **clock source string**. When **clock source code** is 4, 5, 6, or 7 enter the PFI or RTSI pin number into this string control. When **clock source code** is 8 or 9, enter a 0 to indicate general purpose counter 0.

When you are scanning an SCXI-1140 module, you can use an external clock signal connected to the module front connector as the scan clock. If you want to use an external clock signal, you must specify in **clock source string** the SCXI-1140 module to which the external clock signal connects. Specify the SCXI chassis ID and module slot using the syntax of the channel strings described in the *Channel, Port, and Counter Addressing* section of Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, in the *LabVIEW Data Acquisition Basics Manual* (that is, SCx!MDy). In this case, **which clock** is scan clock 1 and **clock source code** is I/O connector.

The clock rate is the rate at which LabVIEW samples data or acquires scans. You can express the clock rate three wayswith **clock frequency**, with **clock period**, or with **timebase source**,

timebase signal, and **timebase divisor**. The VI searches these parameters in that order and sets the clock rate using the first one with a value not equal to -1.

Click here to see the default settings and ranges for the controls of the AI Clock Config VI.

I32 **alternate clock rate specification** contains the following parameters.

Note: The clock frequency parameter must be set to -1 in order to use alternate clock rate specification.

I32 **clock period**. You express **clock period** in seconds per scan or seconds per channel, depending on the value of **which clock**. A **clock period** value of 0 turns the clock off. The default input is -1 which tells LabVIEW to use **timebase source**, **timebase signal**, and **timebase divisor** instead of the **clock period** parameter. The default setting is clock- and device-specific.

I32 **timebase source**.

0: Do not change the timebase source (default input).
1: Internal frequency in Hz (default setting).
2: Source *n*.
3: PFI pin, low to high (E Series only).
4: PFI pin, high to low (E Series only).
5: RTSI pin, low to high (E Series only).
6: RTSI pin, high to low (E Series only).
7: ATCOUT, low to high (E Series only).
8: ATCOUT, high to low (E Series only).

I32 **timebase signal**, in conjunction with **timebase source**, fully specifies the timebase. If **timebase source** is 1, **timebase signal** is the frequency in Hz. If **timebase source** is 2, **timebase signal** is the number of the counter associated with the source, gate, or TC signal. If **timebase source** is 3, 4, 5, or 6, **timebase signal** is the pin number. If **timebase source** is 7 or 8, **timebase signal** is ignored. The default input for **timebase signal** is -1.0, which tells LabVIEW not to change the **timebase signal** setting. The default setting for **timebase signal** is clock and device-specific.

I32 **timebase divisor** is the value the VI uses to divide-down the **timebase signal** and create the clock rate. The default input for **timebase divisor** is -1, which tells LabVIEW not to change the **timebase divisor** setting. The default setting is clock and device-specific.

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

I32 **actual clock rate specification** contains the following output parameters. The actual clock rate the VI used may differ from the rate you specify because of your devices resolution or limitations on the maximum and minimum rates.

I32 **clock frequency** is the actual clock frequency in Hz.

I32 **clock period** is the actual clock period, or .

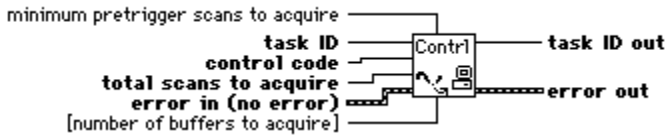
I32 **timebase signal** is the timebase signal frequency in Hz that the VI used when you set **timebase source** to internal frequency in Hz.

I32 **timebase divisor**.

Click here to see the possible scan and channel clock settings and ranges for the alternate clock rate specification cluster in the AI Clock Config VI. The parameters in this cluster are dependent upon the which clock parameter value. After you have chosen whether you want to use a scan or channel clock in the **which clock** parameter, refer to the appropriate table for your clock.

AI Control

Controls the analog input tasks and specifies the amount of data to acquire.



Note: {Macintosh and Windows) You cannot use this VI to start an acquisition when you use a Lab and 1200 Series device, PC-LPM-16, DAQCard-500, or a DAQCard-700 device to scan multiple SCXI channels in multiplexed mode. For this special case, you must use the AI SingleScan VI to acquire data. (For more information about the AI SingleScan VI, refer to its description.) However, you can use the AI Control VI for a Lab and 1200 Series device, PC-LPM-16, DAQCard-500, or DAQCard-700 device when you scan SCXI channels in parallel mode or sample a single SCXI channel in multiplexed mode. You can use this VI for an MIO device scanning SCXI channels in either mode.

I32 **taskID** identifies the group and the I/O operation.

I32 **control code**.

0: Start (default input and setting).
1: Pause immediately.
2: Pause at end of current buffer. *LabVIEW does not currently support this option.*
3: Resume. *LabVIEW does not currently support this option.*
4: Clear.

A **control code** of 0 starts the clock.When **control code** is 0 (Start), this VI tells LabVIEW how much data to acquire. You must start a timed acquisition this way even if the clock source is from an RTSI bus or external connection.

When **control code** is 1 or 2, the pause settings, the clock momentarily stops. When **control code** is 3 (Resume), the clock begins exactly where it left off when **control code** was 1 or 2. If you start after a pause, the acquisition starts at the beginning of the buffer and overwrites existing data.

When **control code** is 4 (Clear), the acquisition stops and internal resources are freed, including the buffers. Then, you must call the AI Buffer Config VI to begin a new acquisition. Setting **control code** to 4 does not deallocate DSP buffers. You must explicitly call the AI Buffer Config VI to deallocate a DSP buffer.

If you have not allocated any memory, LabVIEW ignores **number of buffers to acquire**, but **total scans to acquire** and **pretrigger scans to acquire** are still valid. If you do not call the buffer configuration VI before using the **control code** 0, a nonbuffered acquisition begins.

Note: **Nonbuffered acquisitions are not supported for the following devices.**

- (Macintosh) NB-A2000, NB-A2100, or NB-A2150
- (Sun) SB-A2200 or SB-MIO-16E-4
- (Windows) AT-DSP2200, EISA-A2000, or AT-A2150

I32 **total scans to acquire** defaults to a setting equal to the **scans per buffer** you pass to the AI Buffer Config VI.

-1: Do not change the **total scans to acquire** setting (default input).
0: Indefinite acquisition. LabVIEW ignores **number of buffers** and **minimum pretrigger scans to acquire**.
>0: The total number of scans to acquire per buffer. If the **minimum pretrigger scans to acquire** is 0, **total scans to acquire** can be greater than the number of scans you allocate to a buffer.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster

parameters.

I32 **minimum pretrigger scans to acquire**.

-1: Do not change the **minimum pretrigger scans to acquire** setting (default input).

0: The acquisition is strictly posttrigger (default setting).

>0: A pretrigger acquisition. LabVIEW subtracts **minimum pretrigger scans to acquire** from **total scans to acquire** to obtain the number of posttrigger scans. The **total scans to acquire** parameter must be less than or equal to the scans per buffer you specify with the AI Buffer Config VI (that is, the number of scans you allocate to a buffer).

The following example illustrates why **pretrigger scans to acquire** is a minimum quantity. Assume that with the AI Buffer Config VI you size the buffer to hold 1,000 scans. Then you set **total scans to acquire** to 100 and **minimum pretrigger scans to acquire** to 10. This means the actual number of pretrigger scans can range from 10 to 910. If the actual number of pretrigger scans is less than 910, the trigger occurred before the buffer filled completely. Notice that the number of posttrigger scans is always 90 (100 - 10).

The following are the only devices that have hold-off counters, which guarantee that the device acquires all pretrigger scans.

- All E Series devices
- (Macintosh) NB-A2000
- (Sun) SB-A2200
- (Windows) EISA-A2000 and AT-DSP2200

Other devices cannot prevent the trigger from happening early and may return a badReadOffsetErr error (-10018).

I32 **number of buffers to acquire** is the number of times LabVIEW fills the internal input buffers with data. If **number of buffers to acquire** is 0, LabVIEW acquires buffers indefinitely. If you enable triggering, the VI produces each buffer of data in response to a trigger. This parameter defaults to -1, which tells LabVIEW to leave the **number of buffers to acquire** setting unchanged. The default setting is equal to the **number of buffers** you pass to the AI Buffer Config VI.

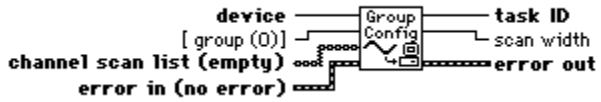
I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

The following table lists default settings and ranges for the AI Control VI.

AI Group Config

Defines what channels belong to a group and assigns them.



Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the channel ranges and scanning order available with your DAQ device.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **group** is the number of the group this VI configures. Values for **group** range from 0 through 15. If **group** is 0 (default) and the **channel scan list** is empty, the actual **channel scan list** is all the channels

in ascending order from channel 0. If the device is connected to an SCXI chassis, you must define the **channel scan list** explicitly.

If this **group** existed previously, LabVIEW stops its task, reconfigures the group, and returns a warning. LabVIEW does not restart the task.

I32 **channel scan list**. Each channel in **channel scan list** becomes a member of the group. The order of the channels in the scan list defines the order in which the channels are scanned during an acquisition. To erase a group, pass an empty channel scan list to the VI along with the group number.

The **channel scan list** parameter is an array of strings. You can use one channel entry per element, specify the entire scan list in a single element, or use a combination of these two methods. If x, y, and z refer to channels, you can specify a list of channels in a single element by separating the individual channels by commasfor example, x,y,z. If x refers to the first channel in a consecutive channel range and y refers to the last channel, you can specify the range by separating the first and last channels by a colonfor example, x:y. Refer to the *Channel, Port, and Counter Addressing* section of Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, in the *LabVIEW Data Acquisition Basics Manual*, for more detailed information about channels and channel string syntax.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **taskID** identifies the group and the I/O operation.

I32 **scan width** indicates the total number of channels in the scan list. For example, if you express **channel scan list** as 0:5,7, **scan width** is 7 channels.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.Click here to see a list of default settings and ranges for the AI Group Config VI. The first row of the table gives the values for most devices, and the other rows give the values for devices that are exceptions to the rule.

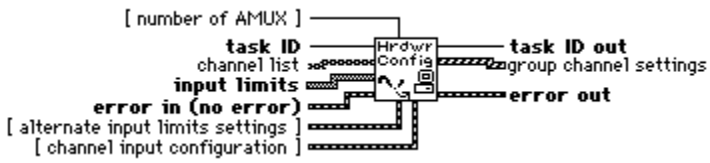
Note: (Macintosh and Windows) The Lab-LC , Lab-NB, Lab-PC+, PCI-1200, PC-LPM-16, DAQCard-500, DAQCard-700, and DAQCard-1200 must scan channel lists containing multiple channels from channel n (n³0) to channel 0 in sequential order, including all channels between n and 0. The NB-A2000, NB-A2150, EISA-A2000, and AT-A2150 allow only the following scan lists: (0), (1), (2), (3), (0, 1), (2, 3), and (0, 1, 2, 3). The NB-A2100 allows the following scan lists: (0), (1), (0, 1), and (1, 0).

The channel scan list range shown above is for single-ended mode. Please refer to Appendix B, *Hardware Capabilities*, in *the LabVIEW Data Acquisition VI Reference Manual*, to determine the valid range for channels in differential mode.

SCXI modules in multiplexed mode must scan channels in ascending consecutive order, starting from any channel on the module. The module order you specify can be arbitrary. SCXI modules in parallel mode must follow the DAQ device restrictions on the order of channel scan lists. Refer to the *Channel, Port, and Counter Addressing* section of Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, in the *LabVIEW Data Acquisition Basics Manual* for information about SCXI channel string syntax.

AI Hardware Config

Configures either the upper and lower input limits or the range, polarity, and gain. The AI Hardware Config VI also configures the coupling, input mode, and number of AMUX-64T devices. The configuration utility determines the default settings for the parameters of this VI.



You can use this VI to retrieve the current settings by wiring **taskID** only or by wiring both **taskID** and **channel list**. If **channel list** is empty, the VI configures channels on a *per group basis*. This means that the configuration applies to all the channels in the group. When you specify one or more channels in **channel list**, the VI configures channels on a *per channel basis*. This means that the configuration applies only to the channels you specify. This VI always returns the current settings for the entire group.

When the configuration is on a per channel basis, **channel list** can contain one or more channels. The channels in **channel list** must belong to the group named by **taskID**. You specify channels the same way you specify them for the AI Group Config VI. If you take multiple samples of a channel within a scan and you want to change the hardware configuration for that channel at each sample, you must supply the settings for each instance of the channel within the scan. If an element of **channel list** specifies more than one channel, the corresponding element of the other arrays applies to all those channels.

The VI applies the values contained in the configuration arrays (**upper input limits, lower input limits**, **coupling**, **range**, **polarity**, **gain**, and **mode**) to the channels in the group (if you configured on a per group basis) or the channels in **channel list** (if you configured on a per channel basis) in the following way. The VI applies the values listed first in the arrays (at index 0) to the first channel in the group or the channel(s) listed in index 0 of **channel list**. The VI applies the values listed second in the configuration arrays (at index 1) to the second channel in the group or channel(s) listed in index 1 of **channel list**. The VI continues to apply the values in this fashion until the arrays are exhausted. If channels in the group or **channel list** remain unconfigured, the VI applies the final values in the arrays to all the remaining unconfigured channels.

Click here to see the AI Hardware Config Channel Configuration on a per channel basis. The parameter **channel scan list**, specified in this table, is specified in the AI Group Config VI.

The **range**, **polarity**, and **gain** determine the lower and upper input limits. When you wire valid input limit arrays (that is, arrays of lengths greater than zero) the VI chooses suitable input ranges, polarities, and gains to achieve these input limits. The VI ignores the **range**, **polarity**, and **gain** arrays.

If you do not wire the input limit arrays, the VI checks **range**, **polarity**, and **gain**. Where the VI finds an array, it sets the corresponding input property to the values in the array. Where the VI does not find an array, it leaves the corresponding input property unchanged.

For some devices and SCXI modules, onboard jumpers set **range**, **polarity**, and/or **gain**. LabVIEW does not alter the settings of jumpered parameters when you specify input limits. If LabVIEW cannot achieve the desired input limits using the current jumpered settings, it returns a warning.

To override the current jumper values, you must call the AI Hardware Config VI and specify **range**, **polarity**, and/or **gain** explicitly. The configuration utility determines the initial setting for these parameters (the default value is the factory jumper setting).

If a pair of input limits values are both 0, the VI does not change the input limits.

SCXI channel hardware configurations are actually a combination of SCXI module and DAQ device settings and require special considerations. The way you specify channels indicates whether LabVIEW alters the SCXI module settings and/or the DAQ device settings. The **input limits** parameter always applies to the entire acquisition path.

When you configure on a per group basis, LabVIEW may alter both SCXI module and DAQ device settings. In this case, **gain** applies to the entire path and is the product of the SCXI channel gain and acquisition device channel gain. LabVIEW sets the highest gain needed on the SCXI module, then adds

DAQ device gain if necessary.

When configuration is on a per channel basis, you can specify the channels in one of three ways. The first way is to specify the entire path, as in the following example.

```
OB0!SC1!MD1!CH0:7
```

If you use this method, LabVIEW can alter both SCXI and DAQ device settings, and **gain** applies to the product of the SCXI channel gain and the DAQ device gain. LabVIEW sets the highest gain needed on the SCXI module, then adds DAQ device gain if necessary.

The second method is to specify the SCXI channel only, as in the following example.

```
SC1!MD1!CH0:7
```

This specification indicates that LabVIEW should alter SCXI settings only. Additionally, **gain** applies only to the SCXI channel.

The third way is to specify the acquisition device channel only, as in the following example.

```
OB0
```

In this case, LabVIEW alters only DAQ device settings. The **gain** parameter applies to the onboard channel only.

Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual* for the channel ranges, input limits, and scanning order available with your DAQ device.

number of AMUX is the number of AMUX-64T devices attached.

-1: Do not change the number of AMUX-64T devices (default input).
0: No AMUX-64T devices.
1: One AMUX-64T devices.
2: Two AMUX-64T devices.
4: Four AMUX-64T devices.

taskID identifies the group and the I/O operation.

channel list. If **channel list** is empty, the hardware configuration is on a per group basis. If **channel list** contains one or more channels, the hardware configuration is on a per channel basis. See *the Channel, Port, and Counter Addressing* section of Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, in the *LabVIEW Data Acquisition Basics Manual*, for the syntax you use to specify the channels.

input limits is an array of clusters. Each array element contains the expected signal limits for the channels specified by the corresponding element of **channels**. If there are fewer elements in this array than in **channels**, the VI uses the last array element for the rest of the channels. The **input limits** array defaults to an empty array, which means the **input limits** keep their default settings. Click here to see the **input limits** cluster parameters.

alternate input limits settings contains the following input parameters.

range values depend on the device. A value of 0 tells LabVIEW to leave the **range** setting unchanged. You express **range** in volts.

polarity.

0: Do not change the **polarity** setting.
1: Bipolar.
2: Unipolar.

I32 **gain** values depend on the device and apply only to the onboard gain. To specify SCXI gains, you must use the configuration utility. A value of 0 tells LabVIEW to leave the **gain** setting unchanged.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **channel input configuration** contains the following parameters.

I32 **coupling**.

0: Do not change the **coupling** setting.
1: DC.
2: AC.
3: Ground. *LabVIEW does not currently support this option*.
4: Internal reference. *LabVIEW does not currently support this option*.

I32 **input mode**.

0: Do not change the **input mode** setting.
1: Differential.
2: Referenced single-ended.
3: Non-referenced single-ended.

I32 **taskID out** has the same value as **taskID in**.

I32 **group channel settings** is an array of clusters that describe the current hardware settings. The number of clusters is equal to the scan width of the group. Element *i* of each cluster corresponds to channel *i* of the actual scan list. Each cluster in **group channel settings** contains the following parameters.

I32 **upper input limits**.

I32 **lower input limits**.

I32 **range**.

I32 **polarity**.

I32 **gain**.

I32 **coupling**.

I32 **input mode**.

I32 **scale multiplier**.

I32 **scale offset**.

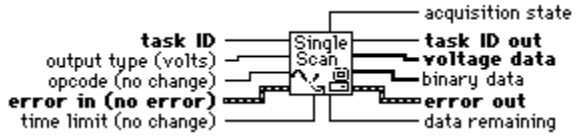
[abc] **channel**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Click here to see a list of default settings and ranges for the AI Hardware Config VI. A tilde (~) indicates that the parameter is configurable on a per group basis only.This means you cannot configure it by channel. The first row of these tables give the values for most devices, and the other rows give the values for devices that are exceptions to the rule. If you did not set the default settings with the configuration utility, use the default settings shown in these tables.

AI SingleScan

Returns one scan of data. If you started an acquisition with the AI Control VI, this VI reads one scan of the data from the internal buffer. On the Macintosh and in Windows, the VI reads from the onboard FIFO if the acquisition is nonbuffered. If you have not started an acquisition, this VI starts an acquisition, retrieves a scan of data, and then terminates the acquisition. The group configuration determines the channels the VI samples.



I32 **taskID** identifies the group and the I/O operation.
The easiest way to call this VI is to wire only the device number to **taskID**. The VI then uses the default group of 0 and returns scaled voltage data from each channel.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **output type**.
0: Do not change the **output type** setting (default input).
1: Scale to volts (default setting).
2: Binary.
3: Both.

I32 **opcode** specifies the type of data retrieval the VI performs.
0: Do not change the **opcode** setting (default input).
1: Read oldest data (default setting).
2: Read newest data.
3: Return data remaining only.
4: Empty the FIFO only (for timed, nonbuffered acquisition only).

(Macintosh and Windows) When you set **opcode** to 1 for a nonbuffered acquisition, the VI reads one scan from the FIFO and returns the data. If **opcode** is 2, the VI reads the FIFO until it is empty and returns the last scan read.

I32 **time limit** is the maximum length of time this VI waits for the requested data. If the time expires, the VI returns a timeout error (-10800) in the **status** parameter. The **time limit** parameter is expressed in seconds. This parameter defaults to -1, which tells LabVIEW to calculate the timeout based on the acquisition clocks or to use 1 s if you use external timing. The resolution of the timeout clock is about 55 ms (Windows), 17 ms (Macintosh), 1 ms (Sun).

I32 **taskID out** has the same value as **taskID in**.

I32 **voltage data** contains the data scaled to volts.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

I32 **acquisition state**.
0: Running.
1: Finished with backlog.
2: Finished with no backlog.
3: Paused.
4: No acquisition.

I32 **binary data** contains the unscaled binary data the ADC produces.

I32 **data remaining** is 1 when data is still in the FIFO or buffer when the VI is about to return. On the Macintosh and in Windows, the data remaining parameter is 0 when the FIFO is empty.
If you do not call the AI Control VI, this VI initiates a single scan using the fastest and most safe channel clock rate. You can, however, alter the channel clock rate with the AI Clock Config VI.

If you run the AI Control VI with **control code** set to 0 (Start), a clock signal initiates the scans.

If you want externally clocked conversions, you must use the AI Clock Config VI to set the clock source to

external.

(Macintosh and Windows) If clock sources are internal and you do not allocate memory, a timed, nonbuffered acquisition begins when you run the AI Control VI with **control code** set to 0. This type of acquisition is useful for synchronizing analog inputs and outputs in a point-to-point control application. The following devices do not support timed, nonbuffered acquisitions.

- (Macintosh) Lab-NB, Lab-LC, NB-A2000, NB-A2100, and NB-A2150
- (Windows) AT-DSP2200, EISA-A2000, and AT-A2150

Note: (Macintosh and Windows) In the event of a FIFO overflow during a timed, nonbuffered acquisition, LabVIEW restarts the device.

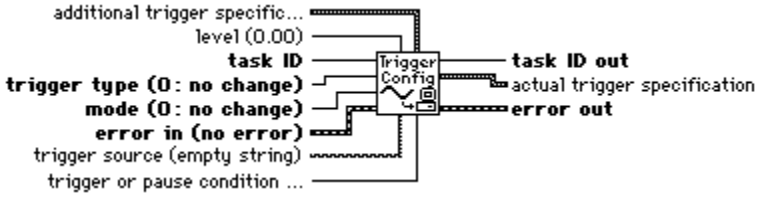
The following lists default settings and ranges for the AI SingleScan VI.

Device-Specific Settings and Ranges for the AI SingleScan VI

	output type		**opcode**		**time limit**	
Device	**D S***	**R***	**D S***	**R***	**D S***	**R***
AT-DSP2200 EISA-A2000 AT-A2150 NB-A2000 NB-A2100 NB-A2150	1	1<= *n*<=3	1	1	variable	*n*>=0
All Other Macintosh and Windows Devices		1<= *n*<=3	1	1<= *n*<=4	1<= *n*<=4	*n*>=0
All Sun Devices	1	1<= *n*<=3	1	1<= *n*<=3	variable	*n*>=0

AI Trigger Config

Configures the trigger conditions for starting the scan and channel clocks and the scan counter.



Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for information on the triggers available with your DAQ device. Refer to your E Series device user manual for a detailed description of the triggering capabilities of the device.

I32 **taskID** identifies the group and the I/O operation.

I32 **trigger type**.

0: Do not change the **trigger type** setting (default input).
1: Analog trigger (default setting).

2: Digital trigger A.
3: Digital trigger B.
4: Digital scan clock gating.
5: Analog scan clock gating.
6: General-purpose analog trigger ATCOut (E Series only). *LabVIEW for Macintosh does not currently support this option.*

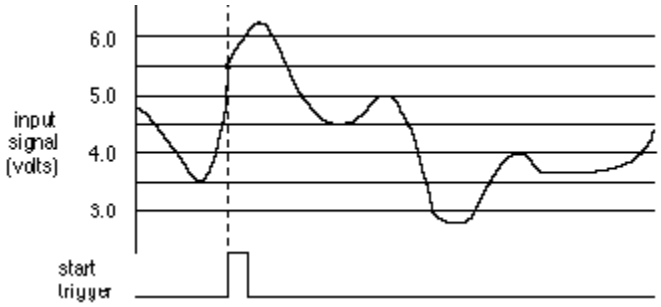
The trigger types 1, 2, and 3 interact with the **total scans to acquire** and **pretrigger scans to acquire** parameters of the AI Control VI in the following manner. Note that a trigger is not automatically enabled when the **pretrigger scans to acquire** value is not 0. You must explicitly enable a trigger by using this VI.

The following is a detailed description of trigger types 1, 2, and 3 as they apply to three types of applications: posttrigger, pretrigger with software start, and pretrigger with hardware start. The other trigger types are discussed at the end of this section.

Application Type 1: Posttriggered Acquisition (Start Trigger Only)

If **total scans to acquire** is ³ 0 and **pretrigger scans to acquire** is 0, you are performing a posttriggered acquisition. A **trigger type** of 1 or 2 (analog trigger or digital trigger A, respectively) starts the acquisition (digital trigger B is illegal). You provide a start trigger. To determine the default pin to which you connect your trigger signal, go to theAI Trigger Config VI Device-Specific Settings and Ranges (Part 1) On some devices you can specify an alternative source through the **trigger source** parameter.

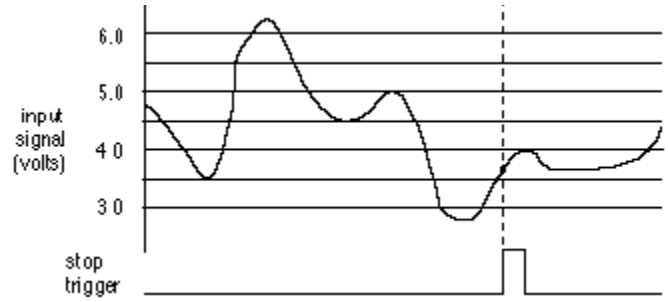
(Macintosh and Windows) With E Series devices, if you are using an analog trigger and the analog signal is connected to one of the analog input channels, that channel must be first in the scan list. This restriction does not apply if you connect the analog signal to PFI0.



In the above illustration, **total scans to acquire** is 1000 and **pretrigger scans to acquire** is 0. The start trigger can come from digital trigger A or an analog trigger (**trigger or pause condition** =1: Trigger on a rising edge or slope, **level** = 5.5 V, **window size** = 0.2 V).

Application Type 2: Pretriggered Acquisition (Stop Trigger Only)

If **total scans to acquire** and **pretrigger scans to acquire** are both > 0, a **trigger type** of 1 or 2 (analog trigger or digital trigger A, respectively) starts the acquisition of posttrigger data after the pretrigger data is acquired. The trigger is called a *stop trigger* because the acquisition does not stop until the trigger occurs. A software strobe starts the acquisition. This is a software start pretrigger acquisition. You provide the stop trigger. To determine the default pin to which you connect your trigger signal, go to the AI Trigger Config VI Device-Specific Settings and Ranges (Part 2) or AI Trigger Config VI Device-Specific Settings and Ranges (Part 3) tables.. On some devices, you can specify an alternative source through the **trigger source** parameter.



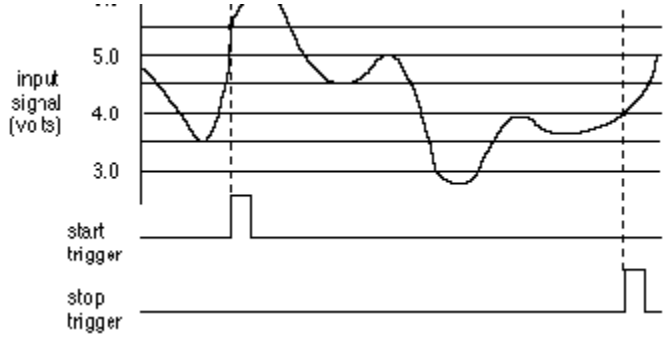
In the above illustration, **total scans to acquire** is 1000 and **pretrigger scans to acquire** is 900. The stop trigger can come from digital trigger A or an analog trigger (**trigger or pause condition** = 1: Trigger on rising edge or slope, **level** = 3.7 V, **window size** = 0.5 V).

(Macintosh and Windows) With E Series devices, if you are using an analog trigger and the analog signal is connected to an analog input channel, that channel must be the only channel in the scan list (no multiple channel scan allowed). This restriction does not apply if you connect the analog signal to PFI0.

Application Type 3: Pretriggered Acquisition (Start and Stop Trigger)

Application Type 3 is used infrequently. Unless you plan to provide both a start trigger and a stop trigger, skip this section.

On MIO devices, you can enable both the start trigger and the stop trigger. (You must call the AI Trigger Config VI twice to do this.) In this case, a digital or analog trigger signal starts the acquisition rather than a software strobe. This is a hardware start pretriggered acquisition. You provide both the start trigger (as described in Application Type 1) and the stop trigger (as described in Application Type 2). To determine the default pin to which you connect your trigger signal, go to the AI Trigger Config VI Device-Specific Settings and Ranges (Part 2) or AI Trigger Config VI Device-Specific Settings and Ranges (Part 3) tables. On some devices, you can specify an alternative source through the **trigger source** parameter.



In the above illustration, **total scans to acquire** is 1000 and **pretrigger scans to acquire** is 900. The start trigger can come from digital trigger B or an analog trigger (**trigger or pause condition** = 1: Trigger on rising edge or slope, **level** = 5.5 V, **window size** = 0.2 V). The stop trigger can come from digital trigger A or an analog trigger (**trigger or pause condition** = 1: Trigger on rising edge or slope, **level** = 4.0 V, **window size** = 0.2 V). Notice that some of the data after the start trigger has been discarded, because all 900 pretrigger scans have been collected and the stop trigger is more than 900 scans away from the start trigger.

(Macintosh and Windows) When using analog triggering on E Series devices, there are several restrictions that apply, as shown in the following table.

Restrictions for Analog Triggering on E Series Devices

Start Trigger	Stop Trigger	Restrictions

Digital A	Digital B	None
Digital B	Analog	Analog signal must be connected to PFI0, unless you are scanning only one channel, in which case the input to that channel can be used.
Analog	Digital A	Analog signal must be first in scan list if it is connected to an analog input channel.

A **trigger type** of 4 (digital scan clock gating) enables an external TTL signal to gate the scan clock on and off, effectively pausing and resuming an acquisition.

Channel clock and scan clock are the same on the NB-MIO-16. Therefore, if the scan clock gate becomes FALSE, the current scan does not complete and the scan clock ceases operation. When the scan clock gate becomes TRUE, the scan clock immediately begins operation again, where it left off previously. You wire your signal to the EXTGATE pin.

A **trigger type** of 5 (analog scan clock gating) enables an external analog signal to gate the scan clock on and off, effectively pausing and resuming an acquisition. A trigger type of 6 allows you to use the output of the analog trigger circuitry (ATCOUT) as a general purpose signal. For example, you can use ATCOut to start an analog output operation, or you can count the number of analog triggers appearing at ATCOut.

Note: **Trigger types 1, 5, and 6 on E Series devices use the same analog trigger circuitry. All three types can be enabled at the same time, but the last one enabled dictates how the analog trigger circuitry behaves. The E Series restrictions described in the trigger applications apply to all three trigger types.**

Trigger type 5 on E Series devices uses the digital scan clock gate and the analog trigger circuitry. Therefore, enabling trigger type 5 overwrites any settings made for trigger type 4.

I32 **mode** turns the trigger type on or off.

0: Do not change the **mode** setting (default input).
1: Off (default setting).
2: On.
3: Clear all triggers. This option turns off all triggers and returns all trigger parameters to their default settings.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **additional trigger specification** contains the following parameters.

I32 **window size** specifies a voltage window for analog edge/level triggering and gating, which the analog trigger signal must leave before a trigger occurs. That is, **window size** acts like a noise filter for an analog trigger. For example, if you set **trigger or pause condition** to 1 (trigger on rising edge or slope), set **level** to 1.0 V, and set **window size** to 0.1 V, the trigger signal must drop below 0.9 V before a trigger can occur. If you change **trigger or pause condition** to 2 (trigger on falling edge or slope), the voltage has to rise above 1.1 V before a trigger can occur.

For analog window triggering and gating, **window size** + **level** specifies the top of the window. You express **window size** in volts. The default input and setting are 0.0 V.

I32 **coupling**. *LabVIEW for Sun does not currently support this parameter.*

0: Do not change the **trigger coupling** setting (default input).
1: DC.
2: AC.

The default setting depends on the device you use.

I32 **delay** specifies how long the device waits after a trigger occurs before sampling posttrigger data. You express **delay** in seconds. The default input and setting are 0.0 s (no delay).

I32 **skip count** is the number of triggers the device skips before triggering the acquisition. The default input is -1, which tells LabVIEW to leave the trigger **skip count** setting unchanged. The default setting is 0, which tells the VI not to skip any triggers.

I32 **time limit** specifies how long after the acquisition starts the device waits for a trigger before stopping the acquisition. You use **time limit** when software performs the triggering. The default input is -1.0, which tells LabVIEW to leave the **time limit** setting unchanged. This parameter defaults to a setting of 1.0 s.

I32 **level** specifies the voltage at which the trigger is to occur. The **level** parameter is valid only when **trigger type** is 1 (analog trigger), 5 (analog scan clock gating), or 6 (general purpose analog trigger). The default input and setting are 0.0 V.

I32 **trigger source** specifies where the trigger comes from. In the following description, the quotation marks are used only to delimit the strings. Do not enter the quotation marks.
The default input is an empty string. This tells the VI to leave the **trigger source** setting unchanged. The default setting is dependent upon the **total scans to acquire** and **pretrigger scans to acquire** from the AI Control VI, as well as **trigger type** from this VI. See Table 5-3 to determine the default trigger sources for your device.

When **trigger type** is 1 (analog trigger), 5 (analog scan clock gating), **trigger source** defaults to 0 for analog input channel 0. On the Macintosh or in Windows, the trigger source also defaults to 0 when or trigger type is 6 (general purpose analog trigger ATCOut). The following are the valid **trigger source** values.

n	(where n is an analog input channel number)
PFI0	(E Series only)
EXT0	(A200 only)

When **trigger type** is 2 (digital trigger A), 3 (digital trigger B), or 4 (digital scan clock gating), and **trigger source** is an empty string. LabVIEW uses the default **trigger source** as shown in Table 5-3. On E Series devices, you can specify other trigger sources as shown below.

PFIn	(0<=n<=9)
RTSIn	(0<=n<=6) (Macintosh and Windows)
GPCTR0	(output of GPCTR0 can only be used as a start trigger)

When **trigger type** is 4 (scan clock gating), LabVIEW ignores **trigger source**.

I32 **trigger or pause condition** specifies when the analog input operation triggers or pauses. Values 3 through 8 are only applicable with E Series devices.

0: Do not change the **trigger or pause condition** setting (default input).
1: Trigger on rising edge or slope (default setting).
2: Trigger on falling edge or slope.
3: Trigger upon entering window (analog only).
4: Trigger upon leaving window (analog only).
5: Pause scanning while inside window (analog only).
6: Pause scanning while outside window (analog only).
7: Pause scanning while above level or while TTL is high.
8: Pause scanning while below level or while TTL is low.

When scanning is paused, the current scan finishes before the pause takes effect.

1 and 2 are valid for **trigger types** 1 (analog trigger), 2 (digital trigger A), 3 (digital trigger B), and 6 (general purpose analog trigger ATCOUT). 3 is valid for **trigger types** 1 (analog trigger) and 6 (general

purpose analog trigger ATCOUT). 4 is valid for **trigger type** 1 (analog trigger). 5 and 6 are valid for **trigger type** 5 (analog scan clock gating). 7 and 8 are valid for **trigger types** 4 (digital scan clock gating) and 5 (analog scan clock gating).

taskID out has the same value as **taskID in**.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

actual trigger specification contains the following output parameters.

actual level is the **level** the VI used.

actual hysteresis is the **window size** the VI used.

actual delay is the **delay** the VI used.

(Macintosh and Windows) For some devices, digital triggering is supported, but for these devices the source is predetermined. Therefore, the **trigger source** parameter is invalid. Click here to see the pin names on the I/O connector to which you should connect your digital trigger signal.

Click here to see a list of the default settings and ranges for the AI Trigger Config VI. The first row of each table gives the values for most devices, and the other rows give the values for devices that are exceptions to the rule.

AI Buffer Config VI Device Settings and Ranges

Device	scans per buffer DS*	scans per buffer R*	number of buffers DS*	number of buffers R*	allocation mode DS*	allocation mode R*
Most devices	100	0, *n*>=3	1	0, 1	2	1, 2
Lab-NB Lab-LC	100	*n*>=3	1	0, 1	2	1, 2
AT-DSP2200	100	*n*>=0	1	*n*>=0	2	1<=*n*<=4
NB-A2000 EISA-A2000 NB-A2100 NB-A2150 AT-A2150	100	*n*>=0	1	*n*>=0	2	1, 2
SB-A2200	100	*n*>=0	1	*n*>=1	2	1, 2
SB-MIO-16E-4	100	0, *n*>=3	1	1	2	1, 2

D S = Default Setting; R = Range

AI Clock Config VI Default Settings and Ranges for the Scan Clock

Device	clock freq in scans/s Default Setting	Range
AT-MIO-16X MIO-16L/H9	disabled	0.0015<=*n*<=111,111
MIO-16L/H15	disabled	0.0015<=*n*=<=66,667
MIO-16L/H25	disabled	0.0015<=*n*<=40,000
AT-MIO-16E-10 AT-MIO-16DE-10	disabled	0.00596<=*n*<=100,000
AT-MIO-16E-2	disabled	0.00596<=*n*<=500,000
AT-MIO-64E-3	disabled	0.00596<=*n*<=333,333
AT-MIO-16XE-50 PCI-MIO-16XE-50	disabled	0.00596<=*n*<=20,000
NEC-MIO-16E-4 SB-MIO-16E-4	disabled	0.00596<=*n*<=250,000
NB-A2000 EISA-A2000	1,000,000	0.0015<=*n*<=1,000,000
Lab and 1200 Series Devices	7,812.5	0.0015<=*n*<=62,500
AT-MIO-16F-5 AT-MIO-64F-5	disabled	0.0015<=*n*<=200,000
PC-LPM-16	3,125	0.0015<=*n*<=50,000
DAQCard-500 DAQCard-700	3,125	0.0015<=*n*<=100,000
NB-A2150S AT-A2150S	16,000	2,000<=*n*<=24,000
NB-A2150C	32,000	4,000<=*n*<=48,000
AT-A2150C AT-DSP2200 SB-A2200	32,000	4,000<=*n*<=51,200
NB-A2150F	32,000	3,840<=*n*<=51,200
NB-A2100	32,000	8,000<=*n*<=48,000

AI Clock Config VI Possible Channel Clock Settings and Ranges for the clock frequency Parameter

Device	clock freq in channels/s Default Setting	Range
MIO-16L/H9	*	0.0015<=*n*<=111,111
MIO-16L/H15	*	0.0015<=*n*<=66,667
MIO-16L/H25	*	0.0015<=*n*<=40,000
Lab and 1200 Series Devices	*	0.0015<=*n*<=62,500
AT-MIO-16F AT-MIO-64F-5	*	0.0015<=*n*<=200,000
AT-MIO-16X DAQCard-500 DAQCard-700	*	0.0015<=*n*<=100,000
AT-MIO-16E-10 AT-MIO-16DE-10	*	1.53<=*n*<=100,000
AT-MIO-16E-2	*	1.53<=*n*<=500,000
AT-MIO-64E-3	*	1.53<=*n*<=333,333
AT-MIO-16XE-50 PCI-MIO-16XE-50	*	1.53<=*n*<=20,000
NEC-MIO-16E-4 SB-MIO-16E-4	*	1.53<=*n*<=250,000
PC-LPM-16	*	0.0015<=*n*<=50,000
NB-MIO-16XL/H18	*	0.0015<=*n*<=55,600
NB-MIO-16XL/H42	*	0.0015<=*n*<=23,800
All Other Devices	no support	no support

Refer to Chapter 9, Letting an Outside Source Control Your Acquisition Rate*, in* the LabVIEW Data Acquisition Basics Manual *for an explanation of how LabVIEW selects the default channel clock frequency.*

AI Clock Config VI Controls Device-Specific Settings and Ranges

Device	configuration mode DS*	R*	retrigger mode DS*	which clock DS*	R*	clock source DS*	R*
AT-MIO-16E-2 AT-MIO-64E-3 NEC-MIO-16E-4 SB-MIO-16E-4	1	1, 3	no support	1	1, 2	1	1, 2, 4<= *n*<=11
AT-MIO-16E-10 AT-MIO-16DE-10 AT-MIO-16XE-50	1	1, 3	no support	1	1, 2	1	1, 2, 4<=*n*<=9
AT-A2150 NB-A2150 NB-A2100 NB-A2000 AT-DSP2200 EISA-A2000	1	1, 3	no support	1	1	1	1<=*n*<=3
PC-LPM-16 DAQCard-500 DAQCard-700 Lab-PC	1	1, 3	no support	1	1, 2	1	1, 2
Lab-LC Lab-NB NB-MIO-16	1	1, 3	no support	1	2	1	1, 2
SB-A2200	1	1, 3	no support	1	1	1	1
All Other Devices	1	1, 3	no support	1	1, 2	1	1<=*n*<=3

AI Control VI Device-Specific Settings and Ranges

Device	control code D S*	R*	total scans to acquire D S*	R*	min.pretrigger scans to acquire D S*	R*	number of buffers to acquire D S*	R*
AT-DSP2200 EISA-A2000 AT-A2150 NB-A2000 NB-A2150 SB-A2200	0	0, 1, 4	0	0, *n*>=0	*0, n*>=3	1	1	*n*>=0
PC-LPM-16 DAQCard-500 DAQCard-700	0	0, 1, 4	0	0, *n*>=3	0	no support	1	1
MIO-E Series	0	0,	0	0, *n*>=3	0	*0, n*>=3	1	1

		1, 4						
All Other Devices	0	0, 1, 4	0	0, *n*>=3	0	*n*>=0	1	1

D S = Default Setting; R = Range

alternate clock rate specification Cluster Scan Clock Settings and Ranges (Part 1)

Device	clock period in s/scan		timebase source	
	DS*	R*	DS*	R*
MIO-16L/H9	disabled	0.000009<= *n*<=655.35	1	1
MIO-16L/H15	disabled	0.000015<= *n*<=655.35	1	1
MIO-16L/H25	disabled	0.000025<= *n*<=655.35	1	1
AT-DSP2200	0.000031	0.0000195<= *n*<=0.00025	1	1
AT-MIO-16E-10 AT-MIO-16DE-10	disabled	0.0001<= *n*<=167.77	disabled	1, 3<=*n*<=6
AT-MIO-16E-2	disabled	0.000002<= *n*<=167.77	disabled	1, 3<=*n*<=8
AT-MIO-64E-3	disabled	0.000003<= *n*<=167.77	disabled	1, 3<=*n*<=8
AT-MIO-16XE-50 PCI-MIO-16XE-50	disabled	0.00005<= *n*<=167.77	disabled	1, 3<=*n*<=6
NEC-MIO-16E-4	disabled	0.000004<= *n*<=167.77	disabled	1, 3<=*n*<=8
SB-MIO-16E-4	disabled	0.000004<= *n*<=167.77	disabled	1, 5
NB-A2000	0.000001	0.000001<= *n*<=655.35	1	1
EISA-A2000	0.000001	0.000001<= *n*<=655.35	disabled	
Lab and 1200 Series Devices	0.000128	0.000016<= *n*<=655.35	1	1
AT-MIO-16F-5	disabled	0.000005<= *n*<=655.35	1	1
AT-MIO-64F-5	disabled	0.000005<= *n*<=655.35	disabled	1
AT-MIO-16X	disabled	0.000009<= *n*<=655.35	no support	1
PC-LPM-16	0.00032	0.00002<=	disabled	1

		n<=655.35		
DAQCard-500 DAQCard-700	0.00032	0.00001<= *n*<=655.35	disabled	1
NB-A2150S	0.0000625	0.000042<= *n*<=0.0005	1	1
AT-A2150S	0.0000625	0.000042<= *n*<=0.0005	disabled	1
NB-A2150C	0.000031	0.0000208<= *n*<=0.00025	1	1
AT-A2150C	0.000031	0.0000195<= *n*<=0.00025	no support	1
NB-A2150F	0.000031	0.0000195<= *n*<=0.00026	1	1
NB-A2100	0.000031	0.0000208<= *n*<=0.0001224	1	1
SB-A2200	0.000031	0.0000195<= *n*<=0.00025	disabled	
NB-MIO-16X			disabled	1
All Other Devices			disabled	1

* *D S = Default Setting; R = Range*

Click here for Part 2 of this table.

alternate clock rate specification Cluster Scan Clock Settings and Ranges (Part 2)

Device	timebase signal in Hz		timebase divisor	
	DS*	R*	DS*	R*
SB-A2200	32,000	32,000, 48,000, 51,200	disabled	1<=*n*<= 65,535
AT-DSP2200	32,000	32,000, 44,100, 48,000, 51,200	1	1, 2, 4, 8
EISA-A2000 NB-A2000	5,000,000	100, 1,000, 10,000, 100,000, 1,000,000, 5,000,000	5	1<=*n*<= 65,535
Lab-LC Lab-NB Lab-PC+ PCI-1200 PC-LPM-16 DAQCard-500 DAQCard-700 DAQCard-1200 NB-MIO-16	no support	no support	no support	no support
AT-MIO-16F-5 AT-MIO-64F-5 AT-MIO-16X	disabled	100, 1,000, 10,000, 100,000, 1,000,000, 5,000,000	disabled	1<=*n*<= 65,535
AT-A2150S		16,000, 20,000, 24,000	1	1, 2, 4, 8
AT-A2150C		32,000, 44,100, 48,000, 51,200	1	1, 2, 4, 8
NB-A2100	32,000	16,000, 32,000, 44,100, 48,000	1	1, 2
NB-A2150C	32,000	32,000, 44,100, 48,000	1	1, 2, 4, 8
NB-A2150F	32,000	30,750,	1	1, 2, 4, 8

		32,000, 48,000, 51,200		
NB-A2150S	16,000	16,000, 20,000, 24,000	1	1, 2, 4, 8
MIO-E Series Devices	disabled	100,000, 20,000,000	disabled	2<=*n*<= 16,777,216
NB-MIO-16X All Other Windows Devices	disabled	100, 1,000, 10,000, 100,000, 1,000,000	disabled	1<=*n*<= 65,535

* *D S = Default Setting; R = Range*

AI Group Config VI Device-Specific Settings and Ranges

Device	group DS*	group R*	channel scan list DS*	channel scan list R*
Most Windows Devices SB-MIO-16E-4	0	0<=*n*<=15	all channels	0<=*n*<=15
Most Macintosh Devices	0	0<=*n*<=15	all channels	0<=*n*<=15
AT-MIO-64F-5 AT-MIO-64E-3	0	0<=*n*<=15	all channels	0<=*n*<=63
AT-A2150 EISA-A2000	0	0<=*n*<=15	all channels	0<=*n*<=3
AT-DSP2200	0	0<=*n*<=15	all channels	0, 1
Lab-PC+ PCI-1200 DAQCard-1200	0	0<=*n*<=15	all channels	0<=*n*<=7
Lab-LC, Lab-NB	0	0<=*n*<=15	all channels	0<=*n*<=7
NB-A2000 NB-A2150	0	0<=*n*<=15	all channels	0<=*n*<=3
NB-A2100	0	0<=*n*<=15	all channels	0, 1
SB-A2200	0	0<=*n*<=15	all channels	0, 1

* *D S = Default Setting; R = Range*

AI Hardware Config Channel Configuration

Configuration Basis	Array Values	Results
Group	Group **channel scan list** = 1, 3, 4, 5, 7 **channel list** is empty **lower input limit** [0] = -1.0 **upper input limit** [0] = +1.0	All channels in the group have input limits of -1.0 to +1.0 V.
Group	Group **channel scan list** = 1, 3, 4, 5, 7 **channel list** is empty **lower input limit** [0] = -1.0 **upper input limit** [0] = +1.0 **lower input limit** [1] =0.0 **upper input limit** [1] = +5.0 **lower input limit** [2] = -10.0 **upper input limit** [2] = +10.0	Channel 1 has input limits of -1.0 to +1.0 V. Channel 3 has input limits 0.0 to +5.0 V. Channels 4, 5,and 7 have input limits of -10.0 to +10.0 V.
Channel	Group **channel scan list** = 1, 3, 4, 5, 7 **channel list** [0] = 1 **channel list** [1] = 3:5 **lower input limit** [0] = -1.0 **upper input limit** [0] = +1.0	Channels 1, 3, 4, and 5 have input limits of -1.0 to +1.0 V. Channel 7 has the default input limits set by the configuration utility. It is unchanged because it is not listed in **channel list**.
Channel	Group **channel scan list** = 1, 3, 4, 5, 7 **channel list** [0] = 1 **channel list** [1] = 3:5 **lower input limit** [0] = -1.0 **upper input limit** [0] = +1.0 **lower input limit** [1] = 0.0 **upper input limit** [1] = +5.0	Channel 1 has input limits of -1.0 to +1.0 V. Channels 3, 4, and 5 have input limits of 0. 0 to +5.0 V. Channel 7 has the default input limits set by the configuration utility.
Group	Group **channel scan list** = 0, 1, 0, 1 **channel list** is empty **lower input limit** [0] = -1.0 **upper input limit** [0] = +1.0 **lower input limit** [1] = -1.0 **upper input limit** [1] = +1.0 **lower input limit** [2] = -10.0 **upper input limit** [2] = +10.0 **lower input limit** [3] = -10.0 **upper input limit** [3] = +10.0	Channels 0 and 1 have input limits of -1.0 to +1.0 V the first time they are sampled and input limits of -10.0 to +10.0 V the second time they are sampled.

AI Hardware Config VI Device-Specific Settings and Ranges

Device	channel input configuration coupling D	S*	R*	input mode D	S*	R*	number of AMUX D	S*	R*	channel list D	S*
Most Macintosh and Windows Devices SB-MIO-16E-4	1		1	1		1<=*n*<=3	0		0<=*n*<=4	empty	
EISA-A2000 NB-A2000	2		1, 2	2		2	0		0	empty	
PC-LPM-16 Lab-LC, Lab-NB	1		1	2		2	0		0	empty	
Lab and 1200 Series Devices	1		1	2		1<=*n*<=3	0		0	empty	
AT-MIO-16X AT-MIO-64F-5	1		1	1 (no ~)		1<=*n*<=3	0		0<=*n*<=4	empty	
AT-A2150 AT-DSP-2200 NB-A2100 NB-A2150 SB-A2200	1		1, 2	2		2	0		0	empty	
DAQCard-500 DAQCard-700	1		1	2		1, 2	0		0	empty	

DS = Default Setting; R = Range

Note: **Channels 0 and 1 and channels 2 and 3 must have the same coupling for the NB-A2150, AT-DSP2200, AT-A2150, and SB-A2200 devices.**

Click here for device-specific settings and ranges for the Hardware Config VI **group channel settings** cluster.

Hardware Config VI Device-Specific Settings and Ranges for the group channel settings Cluster(Part 1)

Device	upper limit inputs D S*	upper limit inputs R*	lower limit inputs D S*	lower limit inputs R*
Most Macintosh Devices and Windows SB-MIO-16E-4	10	0<=*n*<=10	-10	-10<=*n*<= 0
AT-MIO-16F-5 AT-MIO-16X AT-MIO-64F-5 Lab and 1200 Series Devices Lab-LC, Lab-NB	5	0<=*n*<=10	-5	-10<=*n*<= 0
PC-LPM-16	5 (~)	0<=*n*<=10.0(~)	-5 (~)	-5<=*n*<= 0.0 (~)
DAQCard-500 DAQCard-700	5	2.5<=*n*<=10.0	-5	-10<=*n*<= 2.5
AT-A2150 AT-DSP2200 NB-A2100 NB-A2150 SB-A2200	2.828	2.828	-2.828	-2.828
NB-A2000 EISA-A2000	5	5	-5	-5

DS = Default Setting; R = Range

Click here for Part 2 of this table.

Hardware Config VI Device-Specific Settings and Ranges for the group channel settings Cluster(Part 2)

Device	range ~		polarity ~		gain	
	D S*	R*	D S*	R*	D S*	R*
MIO-16L	20	10, 20	1	1, 2	1	1, 10, 100, 500
MIO-16H	20	10, 20	1	1, 2	1	1, 2, 4, 8
AT-MIO-16F-5	10	10	1	1, 2	1	0.5, 1, 2, 5, 10, 20, 50, 100
AT-MIO-16X	10	10	1 (no ~)	1, 2	1	1, 2, 5, 10, 20, 50, 100
AT-MIO-64F-5	10	10	1 (no ~)	1, 2	1	0.5, 1, 2, 5, 10, 20, 50, 100
MIO-E Series (except AT-MIO-16EX-50)	10	10	1	1, 2	0.5	0.5, 1, 2, 5, 10, 20, 50, 100
AT-MIO-16XE-50 PCI-MIO-16XE-50	20	10, 20	1	1, 2	1	1, 2, 10, 100
PC-LPM-16	10	5, 10	1	1, 2	1	1
Lab and 1200 Series Devices	10	10	1	1, 2	1 (~)	1, 2, 5, 10, 20, 50, 100
DAQCard-500 DAQCard-700	10	5, 10, 20	1	1	1	1
EISA-A2000 NB-A2000	10	10	1	1	1	1
AT-DSP2200 AT-A2150, NB-A2100 NB-A2150, SB-A2200	5.656	5.656	1	1	1	1
NB-MIO-16XL	20	5, 10, 20	1	1, 2	1	1, 10, 100, 500
NB-MIO-16XH	20	5, 10, 20	1	1, 2	1	1, 2, 4, 8

DS = Default Setting; R = Range

Click here for Part 3 of this table.

AI Hardware Config VI Device-Specific Settings and Ranges for the group channel settings Cluster(Part 3)

SCXI Module	gain DS*	R*
SCXI-1100	1~	1, 2, 5, 10, 20, 50, 100, 200, 500, 1,000, 2,000
SCXI-1120 SCXI-1121	1,000	1, 2, 5, 10, 20, 50, 100, 200, 250, 500, 1,000, 2,000
SCXI-1140	1	1, 10, 100, 200, 300, 500, 600, 700, 800
SCXI-1200	1	0.05, 0.02, 0.01, 0.5, 0.2, 0.1, 1, 2, 5, 10, 20, 50, 100, 250, 500, 1000, 2000
SCXI-1141	1	1, 2, 5, 10, 20, 50, 100

DS = Default Setting; R = Range

Digital Trigger Sources for Devices with Fixed Digital Trigger Sources

Device	Posttriggering Start Trigger Pin	Pretriggering Start Trigger Pin	Pretriggering Stop Trigger Pin
MIO-16L/H MIO-16DL/DH	STARTTRIG*	STARTTRIG*	STOPTRIG
NB-MIO-16L/H	STARTTRIG*	no support	no support
AT-MIO-16X AT-MIO-16F-5 AT-MIO-64F-5	EXTTRIG*	EXTTRIG*	EXTTRIG*
Lab and 1200 Series Devices	EXTTRIG	no support	EXTTRIG
PC-LPM-16 DAQCard-500 DAQCard-700	no support	no support	no support
AT-DSP2200 EISA-A2000 AT-A2150 NB-A2000 NB-A2100 NB-A2150	EXTTRIG*	no support	EXTTRIG*

** On the AT-MIO-16X, AT-MIO-16F-5, and AT-MIO-64F-5, the same pin is used for both the start trigger and the stop trigger. Refer to your hardware user manual for more details*

AI Trigger Config VI Device-Specific Settings and Ranges (Part 1)

Device	trigger type		mode		trigger or pause condition		level	
	D S*	R*	D S*	R*	D S*	R*	D S*	R*
Most Macintosh and Windows Devices SB-MIO-16E-4	2	2, 3	1	1<= *n*<=3	no support		no support	
AT-MIO-16E-10 AT-MIO-16DE-10 AT-MIO-16XE-50 PCI-MIO-16XE-50	2	2<= *n*<=4	1	1<= *n*<=3	1	1, 2, 7, 8	no support	
AT-MIO-16E-2 AT-MIO-64E-3 NEC-MIO-16E-4	2	1<= *n*<=6	1	1<= *n*<=3	1	1<= *n*<=8	0	-10<= *n*<=10
Lab and 1200 Series Devices	2	2	1	1<= *n*<=3	no support		no support	
PC-LPM-16 DAQCard-500 DAQCard-700	no support		no support		no support		no support	
AT-DSP2200 AT-A2150 NB-A2100 NB-A2150 SB-A2200	1	1, 2	1	1<=*n*<=3	1	1, 2	0	-2.828<= *n*<=2.828
EISA-A2000 NB-A2000	1	1, 2	1	1<=*n*<=3	1	1, 2	0	-5.12<= *n*<=5.12

DS = Default Setting; R = Range

Click here to go to Part 2 of this table.

AI Trigger Config VI Device-Specific Settings and Ranges (Part 2)

Device	trigger source (analog) D S*	trigger source (analog) R*	additional trigger specifications cluster window size D S*	additional trigger specifications cluster window size R*	additional trigger specifications cluster coupling D S*	additional trigger specifications cluster coupling R*
AT-MIO-16E-2 NEC-MIO-16E-4	0	0<=*n*<=15, PFI0	0	0<=*n*<=20	no support	
AT-MIO-64E-3	0	0<=*n*<=63, PFI0	0	0<=*n*<=20	no support	
EISA-A2000 NB-A2000	0	0 <=*n*<= 3	no support		2	1, 2
AT-A2150 NB-A2100 NB-A2150	0	0<=*n*<=3	0	0<=*n*<=5.656	1	1, 2
AT-DSP2200 SB-A2200	0	0, 1	0	0<=*n*<=5.656	1	1, 2
All Other Macintosh and Windows Devices SB-MIO-16E-4	no support		no support		no support	

DS = Default Setting; R = Range

Click here to go to Part 3 of this table.

AI Trigger Config VI Device-Specific Settings and Ranges (Part 3)

Device	trigger source (digital) DS*	R*
E Series Start Trigger	PFI0	PFI 0~9, RTSI 0~6, GPCTR0
SB-MIO-16E-4 Start Trigger	PFI0	PFI 0~9, GPCTR0
E Series Stop Trigger	PFI1	PFI 0~9, RTSI 0~6
SB-MIO-16E-4 Stop Trigger	PFI1	PFI 0~9
E Series Digital Scan Clock Gate SB-MIO-16E-4 Digital Scan Clock Gate	PFI0	PFI 0~9, RTSI 0~6
All Other Macintosh and Windows Devices	no support	no support

DS = Default Setting; R = Range

Click here to go to Part 4 of this table.

AI Trigger Config VI Device-Specific Settings and Ranges (Part 4)

	additional trigger specifications cluster					
	delay		skip count		time limit	
Device	D S*	R*	D S*	R*	D S*	R*
EISA-A2000 NB-A2000	0	0<=*n*<=655.35	no support		no support	
AT-A2150	0	0<=*n*<=2.05	no support		no support	
NB-A2100 NB-A2150S	0	0<=*n*<=32.77	no support		no support	
NB-A2150C	0	0<=*n*<=16.38	no support		no support	
NB-A2150F	0	0<=*n*<=17.05	no support		no support	
AT-DSP2200 SB-A2200 SB-MIO-16E-4	0	no support	no support		no support	
All Other Devices	no support	no support	no support		no support	

DS = Default Setting; R = Range

AI Buffer Config VI

AI Buffer Config

AI Buffer Read VI

AI Buffer Read

AI Clock Config VI

AI Clock Config

AI Control VI

AI Control

AI Group Config VI

AI Group Config

AI Hardware Config VI

AI Hardware Config

AI SingleScan VI

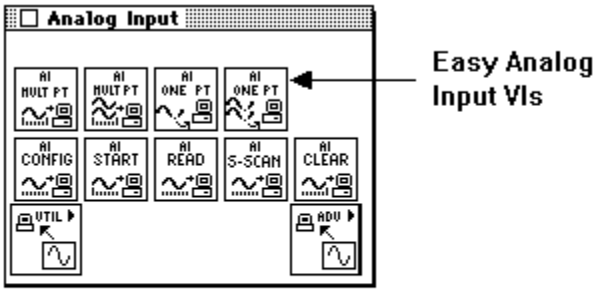
AI SingleScan

AI Trigger Config VI

AI Trigger Config

Easy Analog Input VIs

The Easy Analog Input VIs perform simple analog input operations. You can run these VIs from the front panel or use them as subVIs in basic applications. The Easy Analog Input VIs are located on the top row of the **Easy Analog Input** palette, as shown in the following illustration. Click on a VI icon in this illustration to go to a particular Easy Analog Input VI.



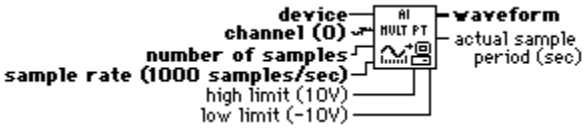
For examples of how to use the Easy Analog Input VIs, open the example library by opening `examples\daq\anlogin.llb`.

Easy VIs

AI Acquire Waveform
AI Acquire Waveforms
AI Sample Channel
AI Sample Channels

AI Acquire Waveform

Acquires a specified number of samples at a specified sample rate from a single input channel and returns the acquired data.



The AI Acquire Waveform VI performs a timed measurement of a waveform (multiple voltage readings at a specified sampling rate) on a single analog input channel. If an error occurs, a dialog box appears, giving you the option to stop the VI or continue.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **channel** is a list of the analog channels you want to use. If `x`, `y`, and `z` refer to channels, you can specify a list of channels by separating the individual channels with commasfor example, `x,y,z`. If `x` refers to the first channel in a consecutive channel range and `y` refers to the last channel, you can specify the range by separating the first and last channels by a colonfor example, `x:y`. Refer to the *Channel Addressing with the AMUX-64T* section of Chapter 5, *Things You Should Know about Analog Input*, in the *LabVIEW Data Acquisition Basics Manual*, for the syntax you use for AMUX-64T channels or SCXI.

I32 **number of samples** is the number of single-channel samples the VI acquires before the acquisition completes. This parameter defaults to 1000 samples.

I32 **sample rate** is the requested number of samples per second the VI acquires from the specified channel. This parameter defaults to a rate of 1000 samples/s.

I32 **high limit** is the highest expected voltage level of the signals you want to measure. The default input is `10` V (`5` V for Lab and 1200 Series devices). LabVIEW uses this value to compute the gain.

I32 **low limit** is the lowest expected voltage level of the signals you want to measure. The default input is `-10` V (`-5` V for Lab and 1200 Series devices). LabVIEW uses this value to compute the gain.

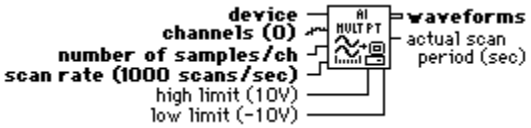
I32 **waveform** is a 1D array that contains scaled analog input data in volts.

I32 **actual sample period** is the actual interval between samples, which is the inverse of the actual sample rate. The **actual sample period** can differ from the requested **sample rate**, depending on the capabilities of your hardware.

Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual,* for the channel numbers and input limits available with your DAQ device.

AI Acquire Waveforms

Acquires data from the specified channels and samples the channels at the specified scan rate.



The AI Acquire Waveforms VI performs a timed measurement of multiple waveforms on the specified analog input channels. If an error occurs, a dialog box appear, giving you the option to abort the operation or continue execution.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **channels** is a list of the analog channels you want to use. If `x`, `y`, and `z` refer to channels, you can specify a list of channels by separating the individual channels with commasfor example, `x,y,z`. If `x` refers to the first channel in a consecutive channel range and `y` refers to the last channel, you can specify the range by separating the first and last channels by a colonfor example, `x:y`. Refer to the *Channel Addressing with the AMUX-64T* section of Chapter 5, *Things You Should Know about Analog Input*, in the *LabVIEW Data Acquisition Basics Manual*, for the syntax you use for AMUX-64T channels or SCXI.

I32 **number of samples/ch** is the number of samples per channel the VI acquires before the acquisition is complete. This parameter defaults to 1000 samples.

I32 **scan rate** is the requested number of scans per second the VI acquires. This parameter defaults to 1000 scans/s. A scan is one sample/channel.

I32 **high limit** is the highest expected voltage level of the signals you want to measure. The default input is `10` V (`5` V for Lab and 1200 Series devices). LabVIEW uses this value to compute the gain.

I32 **low limit** is the lowest expected voltage level of the signals you want to measure. The default input is `-10` V (`-5` V for Lab and 1200 Series devices). LabVIEW uses this value to compute the gain.

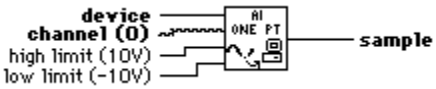
I32 **waveforms** is a 2D array that contains scaled analog input data in volts. Refer to the *Data Organization for Analog Applications* section of Chapter 3*, Basic LabVIEW Data Acquisition Concepts*, in *the LabVIEW Data Acquisition Basics Manual* for more information.

I32 **actual scan period** is the actual interval between scans, which is the inverse of the actual scan rate. The **actual scan period** can differ from the requested **scan rate**, depending on the capabilities of your hardware.

Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual* for the channel numbers and input limits available with your DAQ device.

AI Sample Channel

Measures the signal attached to the specified channel and returns the measured voltage.



The AI Sample Channel VI performs a single, untimed measurement of a channel. If an error occurs, a dialog box appears giving you the option to stop the VI or continue.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **channel** is a list of the analog channels you want to use. If `x`, `y`, and `z` refer to channels, you can specify a list of channels by separating the individual channels with commasfor example, `x,y,z`. If `x` refers to the first channel in a consecutive channel range and `y` refers to the last channel, you can specify the range by separating the first and last channels by a colonfor example, `x:y`. Refer to the *Channel Addressing with the AMUX-64T* section of Chapter 5, *Things You Should Know about Analog Input*, in the *LabVIEW Data Acquisition Basics Manual*, for the syntax you use for AMUX-64T channels or SCXI.

I32 **high limit** is the highest expected voltage level of the signals you want to measure. The default input is `10` V (`5` V for Lab and 1200 Series devices). LabVIEW uses this value to compute the gain.

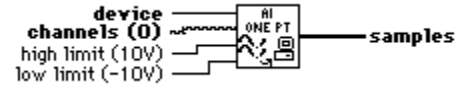
I32 **low limit** is the lowest expected voltage level of the signals you want to measure. The default input is `-10` V (`-5` V for Lab and 1200 Series devices). LabVIEW uses this value to compute the gain.

I32 **sample** contains the scaled analog input data for the specified channel.

Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual* for the channel numbers and input limits available with your DAQ device.

AI Sample Channels

Performs a single voltage reading from each of the specified channels.



The AI Sample Channels VI measures a single voltage from each of the specified analog input channels. If an error occurs, a dialog box appears, giving you the option to stop the VI or continue.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **channels** is a list of the analog channels you want to use. If `x`, `y`, and `z` refer to channels, you can specify a list of channels by separating the individual channels with commasfor example, `x,y,z`. If `x` refers to the first channel in a consecutive channel range and `y` refers to the last channel, you can specify the range by separating the first and last channels by a colonfor example, `x:y`. Refer to the *Channel Addressing with the AMUX-64T* section of Chapter 5, *Things You Should Know about Analog Input*, in the *LabVIEW Data Acquisition Basics Manual*, for the syntax you use for AMUX-64T channels or SCXI.

I32 **high limit** is the highest expected voltage level of the signals you want to measure. The default input is `10` V (`5` V for Lab and 1200 Series devices). LabVIEW uses this value to compute the gain.

I32 **low limit** is the lowest expected voltage level of the signals you want to measure. The default input is `-10` V (`-5` V for Lab and 1200 Series devices). LabVIEW uses this value to compute the gain.

samples is a 1D array that contains scaled analog input data in volts.
Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual* for the channel numbers and input limits available with your DAQ device.

AI Acquire Waveform VI

AI Acquire Waveform

AI Acquire Waveforms VI

AI Acquire Waveforms

AI Sample Channel VI

AI Sample Channel

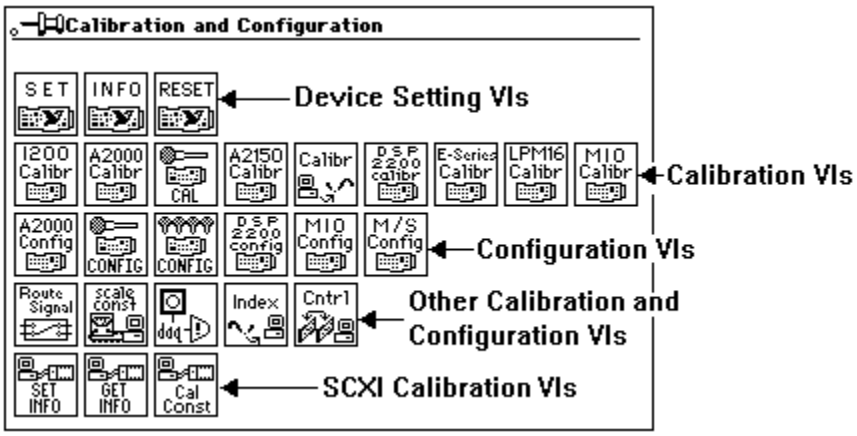
AI Sample Channels VI

AI Sample Channels

Calibration and Configuration VIs

Calibration and Configuration VIs calibrate specific devices and set and return configuration information. You can calibrate certain DAQ devices with the device-specific VIs, but this is not always necessary because National Instruments calibrates all devices at the factory.

You can access the Calibration and Configuration VIs by choosing **Functions**»**Data Acquisition**»**Calibration and Configuration**. The following illustration shows the Calibration and Configuration palette and the different classifications of Calibration and Configuration VIs. Click on the VI labels to go to the particular class of Calibration and Configuration VIs or click on a VI icon to go to a the actual Calibration and Configuration VI.



(Macintosh and Windows) This chapter also includes a VI for controlling the RTSI bus, which is a triggering and timing bus you can use to synchronize, time, and trigger multiple DAQ devices.

(Windows) There is also a VI you can use to set up data acquisition event occurrences.

Calibration and Configuration VI Classifications

Device Setting VIs

Device Reset
Get DAQ Device Information
Set DAQ Device Information (Macintosh and Windows)

Calibration VIs

1200 Calibrate
A2000 Calibrate (Macintosh and Windows)
A2100 Calibrate (Macintosh)
A2150 Calibrate (Windows)
A2200 Calibrate (Sun)
AO-6/10 Calibrate (Windows)
DSP2200 Calibrate (Windows)
E-Series Calibrate (Windows and Sun)
LPM-16 Calibrate (Macintosh and Windows)
MIO Calibrate (Windows)

Configuration VIs

A2000 Configure (Macintosh and Windows)
A2100 Config (Macintosh)
A2150 Config (Macintosh)
DSP2200 Configure (Windows)
Master Slave Config (Macintosh and Windows)

MIO Configure (Windows and Sun)

Other Calibration and Configuration VIs

Channel To Index
DAQ Occurrence Config (Windows)
Route Signal
RTSI Control (Macintosh and Windows)

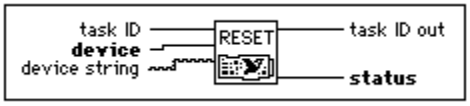
SCXI Calibration VIs

Get SCXI Information
Set SCXI Information
SCXI Cal Constants

Device Setting VIs (Calibration and Configuration)

Device Reset

Resets either an entire device or the particular function identified by **taskID**.



Resetting a **taskID** function has the same result as calling the control VI for that function with **control code** set to clear. When you reset the entire device, LabVIEW clears all tasks and changes all device settings to their default values.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **taskID** identifies the group and the I/O operation.
taskID takes priority over **device**. That is, if you wire a valid value both to **device** and to **taskID**, the VI resets only the function associated with **taskID**. The VI resets the entire device only when **taskID** is unwired.

I32 **device string**. If you want to reset an SCXI chassis or module, you must specify which SCXI device using the channel string syntax described in the Channel, Port, and Counter Addressing section of Chapter 3*, Basic LabVIEW Data Acquisition Concepts*, in the *LabVIEW Data Acquisition Basics Manual*, instead of using the **device** input control. The following table gives examples of the SCXI devices specified by different values wired to **device string**.

SCXI Device String Examples

device string
SCx!MDy
SCx!MD0
SCx

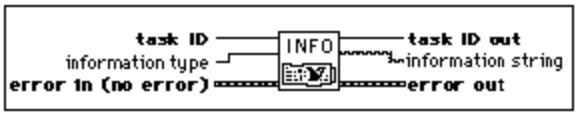
Note: You cannot use device string to specify a plug-in device as a device.

I32 **status** returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes, for a code description.

I32 **taskID out** has the same value as **taskID in.**

Get DAQ Device Information

Returns information about a DAQ device.



Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the transfer methods available with your DAQ device.

I32 **taskID** identifies the group and the I/O operation.
taskID takes priority over **device**. That is, if you wire a valid value both to **device** and to **taskID**, the VI resets only the function associated with **taskID**. The VI resets the entire device only when **taskID** is unwired.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **information type** specifies what information this VI acquires.

0: NI-DAQ Version.
1: device name
2: base address (Macintosh and Windows)
3: device type code (Macintosh)
3: data transfer mode for analog input (Windows)
4: data transfer mode analog output Group 1 (Windows)
5: data transfer mode for analog output Group 2 (Windows)
6: data transfer mode for GPCTR 0 (Windows)
7: data transfer mode for GPCTR 1 (Windows)
8: data transfer mode for digital I/O Group 1 (Windows)
9: data transfer mode for digital I/O Group 2(Windows)
10: device type code (Windows)
11: DMA A level (Windows)
12: DMA B level (Windows)
13: DMA C level (Windows)
14: interrupt A level (Windows)
15: interrupt B level (Windows)
16: interrupt trigger mode (Windows)
17: lpt device mode (Windows)
18: DAQCard-700 counter source (Macintosh and Windows)

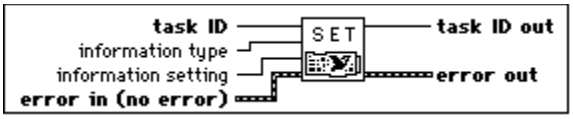
I32 **taskID out** has the same value as **taskID in.**

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

I32 **information string** contains the information you requested. The information string parameter depends on the information type parameter you designate.
Click here to see a list of the various information types and the strings which correspond to each type.

Set DAQ Device Information (Macintosh and Windows)

Sets the data transfer mode for different types of operations.



Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the transfer methods available with your DAQ device.

I32 **taskID** identifies the group and the I/O operation.
taskID takes priority over **device**. That is, if you wire a valid value both to **device** and to **taskID**, the VI resets only the function associated with **taskID**. The VI resets the entire device only when **taskID** is unwired.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already

occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **information type** specifies the what information is being set.

0: Data transfer mode for analog input. (Windows)
1: Data transfer mode for analog output group 1. (Windows)
2: Data transfer mode for analog output group 2. (Windows)
3: Data transfer mode for general purpose counter 0. (Windows)
4: Data transfer mode for general purpose counter 1. (Windows)
5: Data transfer mode for digital I/O group 1. (Windows)
6: Data transfer mode for digital I/O group 2. (Windows)
7: Lpt device mode. (Windows)
8: DAQCard-700 counter 1 source. (Macintosh and Windows)

I32 **information setting** specifies the value to which you set the information.

0: Interrupts
1: Up to one DMA channel
2: Up to two DMA channels
3: Disable
4: Enable
5: Interval timer
6: I/O connector

Click here to see the possible legal combinations of **information type** and **information setting**.

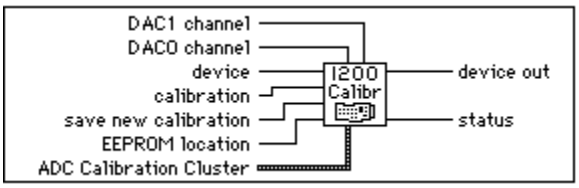
I32 **taskID out** has the same value as **taskID in.**

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Calibration VIs (Calibration and Configuration VIs)

1200 Calibrate

This VI calibrates the gain and offset values for the ADCs and DACs on on 1200 Series devices (i.e., DAQPad-1200, DAQCard-1200, etc.).



You can perform a new calibration (and optionally save the new calibration constants in one of four user areas in the onboard EEPROM) or load an existing set of calibration constants by copying them from their storage location in the onboard EEPROM. LabVIEW automatically loads the calibration constants stored in the onboard EEPROM load area when LabVIEW launches or when you reset the device. By default the EEPROM load area contains a copy of the calibration constants in the factory area

I32 **DAC1 channel** is the analog input channel to which the DAC0 connects when calibration is 3. The range is 0 through 7. The default setting is 1.

I32 **DAC0 channel** is the analog input channel to which the DAC0 connects when calibration is 3. The range is 0 through 7. The default setting is 0.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **calibration** determines the operation to be performed.

0: Reserved.

1: Load the calibration constants from the area specified by the **EEPROM** location control (default setting).

2: Calibrate the ADC. The following controls are used:

- The **reference channel** control names the analog input channel to which the DC reference voltage is connected.
- The **reference channel ground** control names another analog input channel that has been grounded.
- The **reference channel gain** control specifies the gain at which you want to calibrate.
- The **reference voltage** control specifies the value of the DC reference voltage you are using as a calibration reference.
- If the **save new calibration** control is set to 1, then the new calibration constants will be saved in the calibration area named by the **EEPROM location** control. You cannot save new values into the factory areas. If the **save new calibration** control is set to 0, then the new constants will be used in subsequent measurements until you reset your device or quit LabVIEW.

3: Calibrate the DACs. The **DAC0 channel** and **DAC1 channel** controls name the analog input channels to which the analog output channels DAC0 and DAC1 are respectively connected. The gain should be set to 1. If the **save new calibration** control is set to 1, then the new calibration constants will be saved in the calibration area named by the **EEPROM location** control. You cannot save new values into the factory areas. If the **save new calibration** control is set to 0, then the new constants will be used in subsequent measurements until you reset your device or quit LabVIEW.

4: Reserved.

5: Copy the ADC calibration constants from the calibration area named by the **EEPROM location** control to the EEPROM load area.
6: Copy the DAC calibration constants from the calibration area named by the **EEPROM location** control to the EEPROM load area.

I32 **save new calibration** is valid only when calibration is 2 or 3.
0: Do not save the new calibration constants (default setting). The new constants will be used in subsequent measurements until you reset your device or quit LabVIEW.
1: Save the new calibration constants in the calibration area named by the **EEPROM location** control.

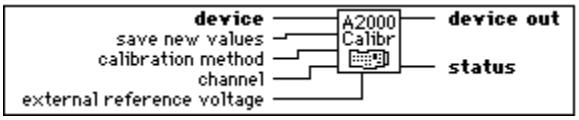
Note: You cannot save new constants in the factory calibration areas.

I32 **EEPROM location** selects the storage location in the onboard EEPROM. You can use different sets of calibration constants to compensate for configuration or environmental changes. You cannot write into EEPROM locations 9 and 10.
0: EEPROM load area.
1: User calibration area 1 (default setting).
2: User calibration area 2.
3: User calibration area 3.
4: User calibration area 4.
5: User calibration area 5.
6: User calibration area 6.
7: User calibration area 7.
8: Reserved.
9: Factory calibration area for unipolar.
10: Factory calibration area for bipolar.

I32 **ADC calibration** contains controls that pertain only to ADC calibrations. You must use this cluster only when the calibration control is set to 2.
I32 **reference channel** refers to the analog input channel to which the DC reference voltage connects when **calibration** is 2. The range is 0 through 7. The default setting is 0.
I32 **reference channel ground** refers to the analog input channel which has been grounded when calibration is 2. The range is 0 through 7. The default setting is 1.
DBL **reference channel gain.** The value of the ADC gain that you which to use when **calibration** is 2. The range is 1, 2, 5, 10, 50, or 100. The default setting is 1.
DBL **reference voltage.** The value of the DC calibration voltage connected to **reference channel** when **calibration** is 2. The minimum voltage is 0 volts. The maximum is determined by dividing the maximum positive voltage (5 V for binary operation and 10 V for unipolar) by the gain. The default setting is 0.0 V.
I32 **device out** has the same value as **device**.
I32 **status** returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes , for a code description.

A2000 Calibrate (Macintosh and Windows)

Calibrates the NB-A2000 or EISA-A2000 A/D gain and offset values or restores them to the original factory-set values.



You can calibrate your NB-A2000 or EISA-A2000 to adjust the accuracy of the readings from the four analog input channels. LabVIEW automatically loads the stored calibration values when it launches or

when you reset your NB-A2000 or EISA-A2000.

Refer to Appendix B, *Hardware Capabilities*, in the *Data Acquisition VI Reference Manual*, for more information on the NB-A2000 or EISA-A2000 DAQ devices.

Warning: Read the calibration chapter in the NB-A2000 or EISA-A2000 User Manual before using the A2000 Calibrate VI.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **save new values** determines whether the VI stores the values it uses in the calibration.

0: Do not save new values in the EEPROM (default input).
1: Save new values in the EEPROM.

If you set **save new values** to 1, then this VI stores the gain and offset calibration values in an EEPROM on the NB-A2000 or EISA-A2000 device, which does not lose its data even if the device loses power. LabVIEW reads these EEPROM values and loads them into the NB-A2000 or EISA-A2000, you can choose to replace the permanent copies of the gain and offset EEPROM values and use the new values until the next calibration, even if you reinitialize the device. You can also choose not to replace the EEPROM values, but to use the new values until the next calibration or initialization.

For example, if you consistently get inaccurate readings from one or more input channels after you reset the device, you can calibrate and save the new gain and offset values as permanent copies in the EEPROM. However, if acquisition results are accurate after initialization but start to drift after a few hours of device operation when the device temperature increases, you can calibrate the device at this operating temperature and retain the current EEPROM values to use after the next initialization.

I32 **calibration method** determines which calibration method the VI uses.

0: Use internal reference to calibrate (default input).
1: Use external reference to calibrate.
2: Reload factory calibration values.

I32 **channel** is the input channel connected to the external reference source. For greatest calibration accuracy, connect the reference source to more than one channel, and set **channel** to -1. The VI averages all channels with input values close to the given **external reference voltage** to find the reference voltage. If you connect the reference source to one input channel only, set **channel** to that channel number.

-1: Determine source channels automatically and average values.
0: Channel 0 (default input).
1: Channel 1.
2: Channel 2.
3: Channel 3.

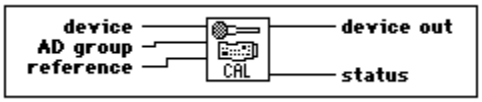
I32 **external reference voltage** is the voltage of the external reference. The range is 3 to 5 V or -5 to -3 V. This parameter defaults to 3.00 V.

I32 **device out** has the same value as **device**.

I32 **status** returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes, for a code description.

A2100 Calibrate (Macintosh)

Selects the desired calibration reference and performs an offset calibration cycle on the ADCs on the NB-A2100 or the NB-A2150.



I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **AD group** selects the A/D channels for calibration.

Device Type	AD group Values	Channels in the Group
NB-A2100	0	0 and 1
NB-A2150	0	0, 1, 2, and 3
	1	0 and 1
	2	2 and 3

I32 **reference** selects the calibration reference you use during the offset calibration cycle.

0: Calibrate the channels using the analog input ground as the reference for each channel (default input).
1: Calibrate the channels using the external signal connected to each channel as the reference for that channel.

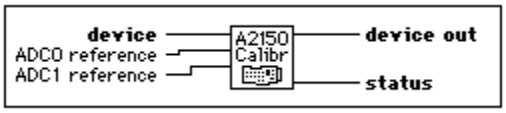
I32 **device out** has the same value as **device**.

I32 **status** returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes, for a code description.

NI-DAQ driver software calibrates the two A/D channels using the analog input ground as the reference for each channel when you turn on the computer.

A2150 Calibrate (Windows)

Performs offset calibrations on the ADCs of the specified AT-A2150.



Refer to Appendix B, Hardware Capabilities, in the LabVIEW Data Acquisition VI Reference Manual, , for more information on the AT-A2150 DAQ device.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **ADC0 reference** specifies the reference source the VI uses to calibrate ADC0 on the AT-A2150.

0: Use the analog input ground as the reference (default setting).
1: Use the external signals connected to ACH0 and ACH1 as references.

I32 **ADC1 reference** specifies the reference source the VI uses to calibrate ADC1 on the AT-A2150.

0: Use the analog input ground as the reference (default setting).

1: Use the external signals connected to ACH2 and ACH3 as references.

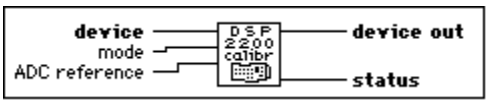
When you launch LabVIEW, or when you reset the AT-A2150, LabVIEW performs an offset calibration using the analog ground as the reference. Use this VI only for device calibration to an external reference or for device recalibration for ground reference after using an external reference.

I32 **device out** has the same value as **device**.

I32 **status** returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes, for a code description.

A2200 Calibrate (Sun)

Performs offset calibrations on the analog input and/or analog output of the SB-A2200.



Refer to Appendix B, Hardware Capabilities, in the LabVIEW Data Acquisition VI Reference Manual, , for more information on the SB-A2200 DAQ device.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **mode** specifies how the VI calibrates the device.

0: Reserved.
1: Calibrate analog input.
2: Calibrate analog output.
3: Calibrate both analog input and output (default setting).

I32 **ADC reference** specifies the reference source the VI uses to calibrate the analog input on the SB-A2200. The VI ignores **ADC reference** unless **mode** is 1 or 3.

0: Use the analog input ground as the reference (default setting).
1: Use the external signals connected to ACH0 and ACH1 as references.

When you launch LabVIEW or reset the AT-DSP2200, LabVIEW performs an offset calibration on both the analog input and output using analog ground as the reference.

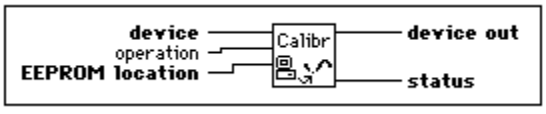
You can use this VI to calibrate the analog input using an external reference or to recalibrate the AT-DSP2200 to compensate for configuration or environmental changes.

I32 **device out** has the same value as **device**.

I32 **status** returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes, for a code description.

AO-6/10 Calibrate (Windows)

Loads a set of calibration constants into the calibration DACs or copies a set of calibration constants from one of four EEPROM areas to EEPROM area 1.



You can load an existing set of calibration constants into the calibration DACs from a storage area in the onboard EEPROM. You can copy EEPROM storage areas 2 through 5 to storage area 1. EEPROM area

5 contains the factory calibration constants. LabVIEW automatically loads the calibration constants stored in EEPROM area 1 upon start-up or when you reset the AT-AO-6/10.

Note: You can also use the calibration utility provided with the AT-AO-6/10 to perform a calibration procedure. Refer to the calibration chapter in the AT-AO-6/10 User Manual for more information.

Refer to Appendix B, *Hardware Capabilities*, in the *Data Acquisition VI Reference Manual*, for more information on the AT-AO-6/10 DAQ devices.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **EEPROM location** selects the storage location in the onboard EEPROM. You can use different sets of calibration constants to compensate for configuration or environmental changes.

0: Reserved.
1: User calibration area 1 (default setting, the default load area).
2: User calibration area 2.
3: User calibration area 3.
4: User calibration area 4.
5: Factory calibration area.

I32 **operation** determines the operation the VI performs.

0: Reserved.
1: Load calibration constants from **EEPROM location** (default setting).
2: Copy calibration constants from **EEPROM location** to EEPROM user calibration area 1.

When LabVIEW initializes the AT-AO-6/10, the DAC calibration constants stored in **EEPROM location** 1 (user calibration area 1) provide the gain and offset values that ensure proper device operation. So, this initialization is the same as running the AO-6/10 Calibrate VI with **operation** set to 1 and **EEPROM location** set to 1. When the AT-AO-6/10 leaves the factory, **EEPROM location** 1 contains a copy of the calibration constants stored in **EEPROM location** 5 (factory calibration).

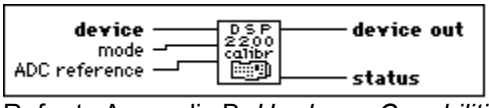
A calibration procedure performed in bipolar mode is not valid for unipolar mode and vice versa. See the calibration chapter of the *AT-AO-6/10 User Manual* for more information.

I32 **device out** has the same value as **device**.

I32 **status** returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes, for a code description.

DSP2200 Calibrate (Windows)

Performs offset calibrations on the analog input and/or analog output of the AT-DSP2200.



Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for more information on the AT-DSP2200 DAQ device.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **mode** specifies how the VI calibrates the device.

0: Reserved.
1: Calibrate analog input.
2: Calibrate analog output.
3: Calibrate both analog input and output (default setting).

I32 **ADC reference** specifies the reference source the VI uses to calibrate the analog input on the AT-DSP2200. The VI ignores **ADC reference** unless **mode** is 1 or 3.

0: Use the analog input ground as the reference (default setting).
1: Use the external signals connected to ACH0 and ACH1 as references.

When you launch LabVIEW or reset the AT-DSP2200, LabVIEW performs an offset calibration on both the analog input and output using analog ground as the reference.

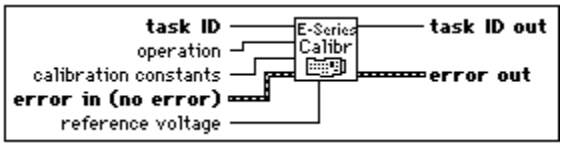
You can use this VI to calibrate the analog input using an external reference or to recalibrate the AT-DSP2200 to compensate for configuration or environmental changes.

I32 **device out** has the same value as **device**.

I32 **status** returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes, for a code description.

E-Series Calibrate (Windows and Sun)

Use this VI to calibrate your E Series device and to select a set of calibration constants to be used by LabVIEW.



Warning: Read the calibration chapter in your device user manual before using the E-Series Calibrate VI.

Your device contains calibration D/A converters (calDACs) that are used for fine-tuning the analog circuitry. The calDACs must be programmed (loaded) with certain numbers, called *calibration constants*. Those constants are stored in non-volatile memory (EEPROM) on your device or are maintained by LabVIEW. To achieve specification accuracy, you should perform an internal calibration of your device just before a measurement session, but after your computer and the device have been powered on and allowed to warm up for at least 15 minutes. Frequent calibration produces the most stable and repeatable measurement performance. The device is not harmed in any way if you recalibrate it as often as you like.

Two sets of calibration constants can reside in two areas inside the EEPROM, called *load areas*. One set of constants is programmed at the factory, the other is left for the user. One load area in the EEPROM corresponds to one set of constants. The load area LabVIEW uses for loading calDACs with calibration constants is called the default load areas. When you get the device from the factory, the default load area is the area that contains the calibration constants obtained by calibrating the device in the factory. LabVIEW automatically loads the relevant calibration constants stored in the load area the first time you call a VI that requires them.

Note: Calibration of your E Series device takes some time. Do not be alarmed if the VI takes several seconds to execute.

Warning: When you run this VI with the operation set to self calibrate or external calibrate, LabVIEW will abort any ongoing operations the device is performing and set all configurations to their defaults. Therefore, you should run this VI before any other DAQ VIs or when no other operations are running.

taskID identifies the group and the I/O operation.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

operation determines the operation the VI performs.

0: No change (default input).
1: Set default load area (default setting).
2: Self calibrate.
3: External calibrate.

Setting the default load area, or value 1, does not perform a calibration, it sets the default load area to the area specified by **calibration constants**.

Self calibrate, or value 2, performs a calibration using the internal voltage reference.

External calibrate, or value 3, performs a calibration using an external voltage reference.

calibration constants specified which set of calibration constants LabVIEW uses.

0: No change (default input).
1: Factory EEPROM area (default setting).
2: NI-DAQ software area.
3: User EEPROM area.

The factory EEPROM area, or value 1, is where factory calibration constants are stored. You cannot modify this area. You can only use it with the set default area operation.

The NI-DAQ software area, or value 2, maintains calibration constants internally; no writing into the EEPROM occurs. You cannot use this setting with the set default area operation. This setting is useful if you want to calibrate your device repeatedly during your program, and if you do not want to store the calibration constants in the EEPROM.

The user EEPROM area, or value 3, is for user calibration constants. If the operation is self calibrate or external calibrate, the new calibration constants will be written into this area and this area becomes the new default load area.

reference voltage is the DC calibration voltage connected to analog input channel 0 when **operation** is 3 (external calibrate). LabVIEW ignores this value for other operation.

taskID out has the same value as **taskID in.**

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.
Set operation to 1 (set default load area) to change the area used for calibration constant loading. The **calibration constants** parameter selects the storage location that becomes the new default load area.

Set **operation** to 2 (self calibrate) if you want to perform self-calibration of your device. The storage location selected by **calibration constants** becomes the new default load area.

Set **operation** to 3 (external calibrate) if you want to perform external calibration of your device. The storage location selected by **calibration constants** becomes the new default load area. Make the following connections before performing an external calibration.

12-bit E Series Devices

- Connect the positive output of your reference voltage source to the analog input channel 8.
- Connect the negative output of your reference voltage source to the AISENSE line.
- Connect DAC0 line (analog output channel 0) with analog input channel 0.

- If your reference voltage source and your computer are floating with respect to each other, connect the AISENSE line with the AIGND line as well as with the negative output of your reference voltage source.

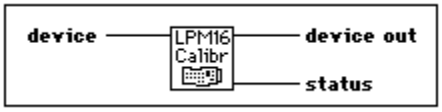
16-bit E Series Devices

- Connect the positive output of your reference voltage source to the analog input channel 0.
- Connect the negative output of your reference voltage source to the analog output channel 8 (by performing those two connections you supply reference voltage to the analog input channel 0, which is configured for differential operation.)

If your reference voltage source and your computer are floating with respect to each other, connect the negative output of your reference voltage source to the AIGND line, as well as to the analog input channel 8.

LPM-16 Calibrate (Macintosh and Windows)

Calibrates the PC-LPM-16, DAQCard-500, or the DAQCard-700 converter. The calibration calculates the correct offset voltage for the voltage comparator, adjusts positive linearity and full-scale errors to less than ±0.5 LSB each, and adjusts zero error to less than ±1 LSB.



Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for more information on the PC-LPM-16, DAQCard-500, or DAQCard-700 device.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

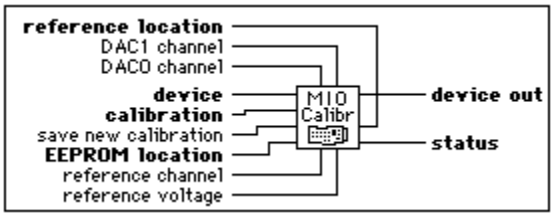
I32 **device out** has the same value as **device**.

I32 **status** returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes, for a code description.

When you call this VI, the ADC goes into a self-calibration cycle. The VI does not return until the ADC finishes this self-calibration. When you launch LabVIEW or reset the PC-LPM-16, DAQCard-500, or DAQCard-700, LabVIEW performs the same function as this VIit calibrates the PC-LPM-16, DAQCard-500, or DAQCard-700 converter.

MIO Calibrate (Windows)

Calibrates the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X gain and offset values for the ADCs and the DACs. You can either perform a new calibration or use an existing set of calibration constants by copying the constants from their storage location in the onboard EEPROM. You can store several sets of calibration constants. LabVIEW automatically loads the calibration constants stored in the EEPROM load area during startup or when you reset the device.



The load area for the AT-MIO-16F-5 is user area 5. The load area for the AT-MIO-64F-5 and AT-MIO-16X is user area 8.

Warning: Read the calibration chapter in your device user manual before using the MIO Calibrate VI.

Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for more information on the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X DAQ devices.

I32 **reference location** is the source of the internal voltage reference constants when **calibration** is 2 or 3. When **calibration** is 4, the VI stores the internal voltage reference constants in **reference location**.

0: Reserved.
1: User reference area 1 (default setting).
2: User reference area 2.
3: User reference area 3 (AT-MIO-16X and AT-MIO-64F-5 only).
4: User reference area 4 (AT-MIO-16X and AT-MIO-64F-5 only).
5: Reserved.
6: Factory reference area (user cannot write into this area).

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **calibration** determines the operation to be performed.

0: Reserved.
1: Load calibration constants from **EEPROM location** (default setting).
2: Calibrate the ADC using internal reference voltage calibration constants in **reference location**.
3: Calibrate the DACs using internal voltage calibration constants in **reference location**. **DAC0 channel** and **DAC1 channel** are the analog input channels to which DAC0 and DAC1 connect, respectively.
4: Calibrate the internal reference voltage. A DC voltage of **reference voltage** must connect to the analog input channel **reference channel**. The VI always stores the calibration constants in **reference location**.
5: Copy ADC calibration constants from **EEPROM location** to EEPROM load area.
6: Copy DAC calibration constants from **EEPROM location** to EEPROM load area.

I32 **EEPROM location** selects the storage location in the onboard EEPROM. You can use different sets of calibration constants to compensate for configuration or environmental changes. You cannot write to **EEPROM location** 9, 10, and 11.

0: Reserved.
1: User calibration area 1 (default setting).
2: User calibration area 2.
3: User calibration area 3.
4: User calibration area 4.
5: User calibration area 5 (load area for the AT-MIO-16F-5).
6: User calibration area 6 (AT-MIO-16X and AT-MIO-64F-5 only).
7: User calibration area 7 (AT-MIO-16X and AT-MIO-64F-5 only).
8: Load area for the AT-MIO-16X and AT-MIO-64F-5.
9: Factory calibration area for unipolar (AT-MIO-16X and AT-MIO-64F-5 only).
10: Factory calibration area for bipolar (AT-MIO-16X and AT-MIO-64F-5 only).
11: AT-MIO-16F-5 factory calibration area.

I32 **DAC1 channel** is the analog input channel to which DAC1 connects when **calibration** is 3. This

parameter is not applicable to the AT-MIO-64F-5 because its DAC1 wraps back internally. This ranges from 0 through 7, with a default setting of 0. If you are using an AT-MIO-16F-5, you must wire DAC1 to the analog input channel in *reverse polarity*. This means you connect the positive terminal of DAC1 to the negative terminal of the analog input channel, and vice versa.


DAC0 channel is the analog input channel that DAC0 connects to when **calibration** is 3. This parameter is not applicable to the AT-MIO-64F-5 because its DAC0 wraps back internally. This parameter ranges from 0 through 7, with a default setting of 0. If you are using an AT-MIO-16F-5, you must wire DAC0 to the analog input channel in *reverse polarity*. This means you connect the positive terminal of DAC0 to the negative terminal of the analog input channel, and vice versa.


save new calibration is valid only when **calibration** is 2 or 3.

0: Do not save new calibration constants in **EEPROM location** (default setting).
1: Save new calibration constants in **EEPROM location**.


reference channel is the analog input channel to which the calibration voltage connects when **calibration** is 4. This parameter ranges from 0 through 7 with a default setting of 0.


reference voltage is the value of the DC calibration voltage connected to **reference channel** when **calibration** is 4. The range is 6 to 10 V. The default setting is 0.0 V.

Note: You should always calibrate the ADC and the DACs after you calibrate the internal reference voltage.

When LabVIEW initializes the AT-MIO-16F-5, AT-MIO-64F-5, or AT-MIO-16X, the ADC and DAC calibration constants stored in appropriate load area provide the gain and offset values to ensure proper device operation. In other words, the initialization performs the equivalent of calling the MIO Calibrate VI with **calibration** set to 1 and **EEPROM location** set to 5 for the AT-MIO-16F-5, and **EEPROM location** set to 8 for the AT-MIO-64F-5 and AT-MIO-16X. When the AT-MIO-16F-5 leaves the factory, **EEPROM location** 5 contains a copy of the calibration constants stored in **EEPROM location** 11 (factory reference).

Unless you previously stored new internal voltage reference constants in **refLoc** (user reference area) 1 or 2 by calling the MIO Calibrate VI with **calibration** set to 4, you must use **reference location** 6 (the factory reference area) when you perform an ADC or a DAC calibration (set **calibration** to 2 or 3, respectively).

A calibration performed in bipolar mode is not valid for unipolar mode and vice versa. The polarity setting determines whether the MIO Calibrate VI performs as bipolar or unipolar calibration, or loads the bipolar or unipolar constants. So, if you change the polarity of your AT-MIO-16F-5, AT-MIO-64F-5, or AT-MIO-16X, you should run the MIO Calibrate VI with **calibration** set to 1 to load the appropriate constants.

Note: If the device takes analog input measurements with the wrong set of calibration constants loaded, you may get erroneous data.

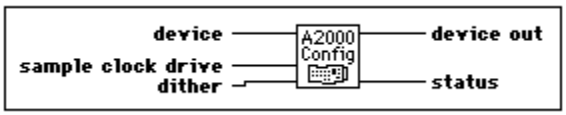

device out has the same value as **device**.


status returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes, for a code description.

Configuration VIs (Calibration and Configuration VIs)

A2000 Configure (Macintosh and Windows)

Configures dithering and whether to drive the SAMPCLK* line for the NB-A2000 or EISA-A2000.



Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for more information on the NB-A2000 or EISA-A2000 DAQ devices.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **sample clock drive** determines whether the device drives the sample clock signal onto the SAMPCLK* line.

0: Sample clock signal does not drive SAMPCLK* line (default input).
1: Sample clock signal drives SAMPCLK* line.

Note: The device cannot receive the sample clock from the SAMPCLK* line and drive it at the same time.

I32 **dither** determines whether the device adds approximately 0.5 LSB rms of white Gaussian noise to the input signal. This is useful for applications that use averaging to increase the resolution of the NB-A2000 or EISA-A2000 to more than 12 bits. Disable dithering for high-speed applications that do not use averaging.

0: Dither disabled (default input).
1: Dither enabled.

I32 **device out** has the same value as **device**.

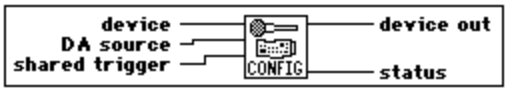
I32 **status** returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes, for a code description.

After system startup, LabVIEW configures the NB-A2000 or EISA-A2000 as follows.

- **sample clock drive** =0:Sample clock signal does not drive SAMPCLK* line.
- **dither** =0:Dither disabled.

A2100 Config (Macintosh)

Selects the signal source used to provide data to the DACs and lets you configure the external digital trigger to be shared by data acquisition and waveform generation operations on the NB-A2100.



I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **DA source** selects the source that supplies data to the DACs.

0: Use the data in the D/A FIFO. (This is the default input setting at startup).

1: Use the data being sampled by the ADCs. With this setting, you can send the data sampled by the ADCs directly to the DACs.

I32 **shared trigger** disables or enables subsequent data acquisition and waveform generation operations to share the external digital trigger.

0: Disable subsequent data acquisition and waveform generation operations to share the external trigger (default input). This indicates that analog input and output tasks should execute independently of each other.

1: Enable subsequent data acquisition and waveform generation operations to share the external trigger. This indicates that the software should recognize the external trigger when both data acquisition and waveform generation operations are ready to receive the trigger. In other words, the computer ignores any trigger applied when you initiate only one operation, and simultaneously accepts both operations when you apply any trigger when both operations have been initiated.

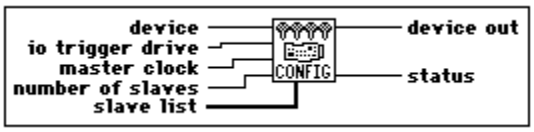
If LabVIEW acquires multiple data acquisition frames and generates multiple waveform cycles with a trigger required at the beginning of each cycle, then the external trigger recognition synchronizes so that each trigger simultaneously initiates the acquisition of the next data frame while generating the output of the next waveform cycle.

I32 **device out** has the same value as **device**.

I32 **status** returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes, for a code description.

A2150 Config (Macintosh)

Selects whether or not LabVIEW should drive an internally generated trigger to the NB-A2150 I/O connector. This VI also determines whether LabVIEW should drive the NB-A2150 sampling clock signal over the RTSI bus to other devices for multiple-device synchronized data acquisition.



I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **io trigger drive** selects whether LabVIEW should connect the trigger signal received over the RTSI bus or the internally generated analog level trigger to EXTTRIG* line at the I/O connector.

0: Do not drive the EXTTRIG* line at the I/O connector (default input).

1: Drive the EXTTRIG* line at the I/O connector.

I32 **master clock** determines whether or not LabVIEW should configure the sampling clock signals of selected devices to be driven over the RTSI bus to other NB-A2150 devices.

0: Do not change the configuration of the sampling clock drive circuitry (default input).

1: Configure the sampling clock circuitry according to the given slave list.

I32 **number of slaves** selects the number of slave devices to be configured and the number of elements in **slave list**.

I32 **slave list** is an array containing the device numbers of the devices that can accept the sampling clock signal of the **device** parameter over the RTSI bus. LabVIEW ignores the **slave list** if **master clock** or **number of slaves** is 0.

Enable **io trigger drive** only if you have executed the RTSI Control VI to receive the RTSITRIG* signal

over the RTSI bus, or if you have enabled the analog level trigger using the AI Trigger Config VI. In these cases, you can monitor the signal being sent to the A/D trigger circuitry at the EXTTRIG* line of the I/O connector after starting the acquisition. A high-to-low edge of the signal triggers the data acquisition.

The NB-A2150 uses signals over the RTSI bus for sampling clock synchronization between two or more NB-A2150 devices. The sampling clock synchronization circuitry makes simultaneous sampling possible on more than four channels using additional NB-A2150 devices. If **master clock** is 1, **slave list** should contain the list of devices that accept the sampling clock from **device**. After you run A2150 Config with **master clock** equal to 1 and **number of slaves** greater than 0, you cannot use the AI Clock Config to set the scan rate for devices in **slave list** until you run A2150 Config again on **device** with **master clock** equal to 1 and **number of slaves** equal to 0.

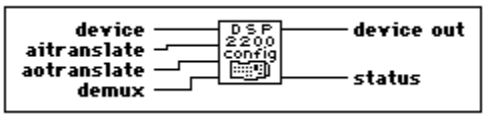
Note: Executing A2150 Config with master clock equal to 1 and number of slaves equal to 0 deconfigures the devices previously in the slave list and sets them up to use their own sampling clock signal.

I32 **device out** has the same value as **device**.

I32 **status** returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes , for a code description.

DSP2200 Configure (Windows)

Specifies data translation and demultiplexing operations that the AT-DSP2200 performs on analog input and output data.



Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for more information on the AT-DSP2200 DAQ device.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **aitranslate** specifies the floating-point and scaling operations that the device performs on analog input data before writing the data to memory.

0: Do not translate. Write unscaled 16-bit integer data to memory (default input).
1: Translate to 32C floating-point format.
2: Translate to 32C floating-point format and scale to volts.
3: Translate to IEEE single-precision floating-point format.
4: Translate to IEEE single-precision floating-point format and scale to volts.

I32 **aotranslate** specifies the floating-point and scaling operations that the device performs on analog output data as it reads data from memory.

0: Do not translate. Source data must be 16-bit integers (default input).
1: Translate from 32C floating-point format.
2: Scale from volts to DAC values and translate from 32C floating-point format.
3: Translate from IEEE single-precision floating-point format.
4: Scale from volts to DAC values and translate from IEEE single-precision floating-point format.

I32 **demux** enables and disables the sorting of analog input data as the device writes it to DSP memory. If you enable **demux**, the device stores data from channel 0, followed by data from channel 1. If you disable **demux**, the device interleaves data from the two channels in DSP memory.

0: Demux disabled (default input).

1: Demux enabled.

I32 **device out** has the same value as **device**.

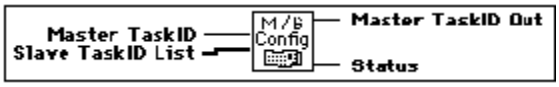
I32 **status** returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes, for a code description.

Because software running locally on the AT&T WE DSP32C DSP chip reads data from the ADCs and writes data to the DACs, you can manipulate the data during these transfers. When you write analog input data to DSP memory, you can write the data as unscaled 16-bit integers, unscaled 32C floating-point numbers, or scaled 32C floating-point voltages. You can use the **demux** option only when you write analog input data to DSP memory. When you enable **demux**, the device writes data from channel 0 consecutively into DSP memory, beginning at the start of each buffer, and writes channel 1 data consecutively beginning at the half-way point of each buffer. When the device writes analog input data to PC memory, it can write the data as unscaled 16-bit integers, unscaled IEEE single-precision floating-point numbers, or scaled IEEE single-precision voltages.

The analog output translations in the opposite directions from the analog input translations. If **aotranslate** is 0, the source data must be in a format suitable for the DACs (16-bit integer DAC values). If **aotranslate** is 1 or 3, the source data are DAC values in 32C format in DSP memory or in IEEE single-precision format in PC memory. If **aotranslate** is 2 or 4, the source data are voltages in 32C format in DSP memory or in IEEE single-precision format in PC memory.

Master Slave Config (Macintosh and Windows)

Configures one device as a master device and any remaining devices as slave devices for multiple-buffered analog input operations.



Makes sure LabVIEW always re-enables the slave devices *before* the master device in a multiple-buffer analog input operation. Only the following devices, which support multiple buffered acquisitions, can use this VI.

- (Windows) EISA-A2000, AT-2150, and AT-DSP2200
- (Macintosh) NB-A2000, NB-A2100, and NB-A2150.

The master device sends a trigger or clock signal to the slave device(s) to control the slave device sampling. In a multiple-buffer acquisition, you must enable the slave device before the master device to make sure the slave device always responds to a master signal. If you enable the master device first, it can send a signal to the slave devices before they can respond. You are responsible for the initial startup order. You should always start the master device last. The Master Slave Configuration VI makes sure LabVIEW arms the master device last for each subsequent buffer acquired during a multiple-buffer acquisition.

I32 **Master TaskID** is the taskID of the device that controls the slave device sampling. This parameter defaults to 0.

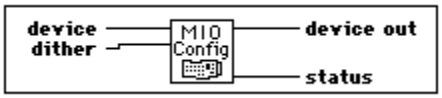
I32 **Slave TaskID List** contains the taskIDs of the slave devices that the master device controls.

I32 **Master TaskID Out** has the same value as **Master TaskID**.

I32 **status** returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes, for a code description.

MIO Configure (Windows and Sun)

Turns dithering on and off. This VI supports the following devices: AT-MIO-16F-5, AT-MIO-64F-5, all 12-bit E Series devices, and all 1200 Series devices.



Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for more information on the devices supported by this VI.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **dither** determines whether the device adds approximately 0.5 LSB rms of white Gaussian noise to the input signal. This is useful for applications that use averaging to increase the resolution of your DAQ device to more than 12 bits. Disable dithering for high-speed applications that do not use averaging.

0: Dither disabled (default input).
1: Dither enabled.

I32 **device out** has the same value as **device**.

I32 **status** returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes , for a code description.

Other Calibration and Configuration VIs

Channel To Index

Uses the current group configuration for the specified task to produce a list of indices into the groups scan or update list for each channel specified in the channel list.

You can use this list of channel indices to locate data for a particular channel within a multiple channel buffer. You can also use the indices to read or write to a group subset with the buffer read and write VIs.

Refer to your specific device information in Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the channel limitations that apply to your device.

I32 **taskID in** identifies the group and the I/O operation.

I32 **channel list** is a list of the analog channels you want to use. If `x`, `y`, and `z` refer to channels, you can specify a list of channels by separating the individual channels with commasfor example, `x,y,z`. If `x` refers to the first channel in a consecutive channel range and `y` refers to the last channel, you can specify the range by separating the first and last channels by a colonfor example, `x:y`. Refer to the *Channel Addressing with the AMUX-64T* section of Chapter 5, *Things You Should Know about Analog Input*, in the *LabVIEW Data Acquisition Basics Manual*, for the syntax you use for AMUX-64T channels or SCXI.

I32 **buffer number**. Use **buffer number** when **taskID** refers to an analog output group and you assign output channels within that group to different buffers. This VI defaults to -1, which means LabVIEW leaves the current **buffer number** setting unchanged. The VI defaults to a **buffer number** setting of 1.

I32 **taskID out** has the same value as **taskID in.**

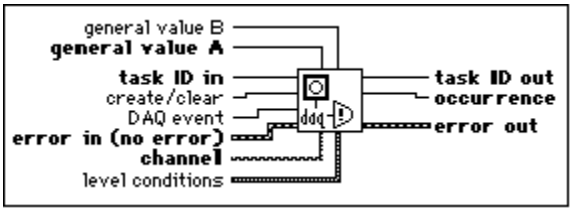
U32 **channel indices** is the list of indices in the scan or update group of the channels specified in **channel list**.

I32 **status** returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes, for a code description.

Click here to see the possible values for the channel scan list, channel list, and channel indices parameters for the Macintosh and Windows. The **channel scan list** parameter is an input for the group configuration VIs.

DAQ Occurrence Config (Windows)

Creates occurrences that are set by data acquisition events.



A DAQ event can be the completion of an acquisition, the acquisition of a certain number of scans, an analog signal meeting certain trigger conditions, a periodic event, an aperiodic (externally driven) event, or a digital pattern match or mismatch. Your VI can sleep while waiting for an occurrence to be set, freeing your computer to execute other VIs.

When you set the **create/clear** control to 1 (create) and call the VI, this VI creates an occurrence. Use the **DAQ event** control to select the event that sets the occurrence. Wire the occurrence this VI produces to the Wait on Occurrence function. Anything you wire to the output of the Wait on Occurrence function does

not execute until the occurrence is set. The occurrence is set each time the event occurs. The occurrence does not clear until you set the **create/clear** control to 0 (clear) and call this VI, or call the Device Reset VI for the device.

LabVIEW returns a Not a Refnum file I/O constant along with a non-zero status code if it cannot create the occurrence.

For each computer platform, LabVIEW limits the number of occurrences per second that you can set. Although this limit depends on the speed of your computer, avoid exceeding 500 occurrences per second.

For some of the events, you must perform your operation using interrupts instead of DMA. Refer to the description of the **DAQ event** control in this section for more information.


general value A specifies a scan count for **DAQ event** 0 and 1, specifies the interrupt group for a **DAQ event** 4 when you use this event with the PC-TIO-10, and specifies one of the digital patterns for **DAQ event** 5 and 6. If **DAQ event** is 0 or 1, the value of **general value A** must be greater than or equal to 1. If **DAQ event** is 4 (used with PC-TIO-10), **general value A** must be either 1 (EXTIRQ1) or 2 (EXTIRQ2). The default value is 100.


taskID identifies the group and the I/O operation.


error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.


channel specifies the analog input channel for **DAQ event** 3, and the counter to which **DAQ event** 4 applies, and the digital input or output port for **DAQ event** 5 and 6. The default input is 0. See the *Channel, Port, and Counter Addressing* section of Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, in the *LabVIEW Data Acquisition Basics Manual* for the valid channel string syntax.


general value B specifies the second digital pattern for **DAQ event** 5 and 6.


create/clear specifies whether you should create a new occurrence or clear all occurrences for the device identified by **taskID in**.

0: Clear all occurrences for the device.
1: Create a new occurrence (default input).


DAQ event selects the reason for setting the occurrence. When LabVIEW detects the event, it sets the occurrence and executes anything waiting on the occurrence. **DAQ event**, **channel**, **general value**, and **level conditions** specify the event conditions. Events 0 through 3 work with an analog input application. Event 4 must include a counter pulse generation application for periodic events or you must set the output state of the counter to high impedance for aperiodic (externally driven) events. The following are types of events and their numeric codes.

0: Set the occurrence exactly once when LabVIEW acquires a number of scans equal to of the **general value A** control. The VI ignores the **channel**, **general value B**, and **level conditions** controls.
1: Set the occurrence every time LabVIEW acquires the number of scans equal to the **general value A** control. LabVIEW ignores the **channel**, **general value B**, and **level conditions** controls (default input). If your system uses DMA, **general value A** must divide evenly into your buffer size.
2: Set the occurrence when the acquisition completes or stops because of an error. LabVIEW ignores the **channel**, **general value A**, **general value B**, and **level conditions** controls.
3: Set the occurrence when the analog input data obtained from any channel list in the **channel** control meets the conditions specified by the **level conditions** cluster. For this event, **level A** specifies a trigger level and **level B** sets a window size. When you set **slope** to positive only, the data must go below the value of **level A** - **level B** and then go above **level A** before LabVIEW sets the occurrence. When you set **slope** to negative only, the data must go above the value of **level A** + **level B** and then go below **level A** before LabVIEW sets the occurrence. When you set **slope** to either positive or negative, either condition sets the occurrence. You must use interrupts, not DMA, when using this data acquisition event.

LabVIEW ignores the **general value B** control.

4: Set the occurrence every time the output of the counter specified by the **channel** changes state so that it generates an interrupt. Some counters require state change from low to high, others from high to low. The table below lists the state change that causes an interrupt for each DAQ device. For the PC-TIO-10, you must also set **general value A** to 1 (EXTIRQ1) or 2 (EXTIRQ2) to specify which of the two interrupt groups you want to use and you must wire the signal that you want to interrupt to that pin. You can use **DAQ event** 4 along with the counter VIs to generate timed occurrences and execute portions of your diagram periodically. First, call the DAQ Occurrence Config VI with **DAQ event** set to 4 and the **channel** control set to the counter you want to use. You must configure the occurrence before you start the counter. That counter must be idle when the DAQ Occurrence Config VI executes. Then, call the counter VIs to set up a timed pulse generation. The occurrences are set at a rate equal to the rate of the pulse generation. Be careful not to exceed 1,000 pulses/s. You can also use this event to respond to aperiodic external events. Use the CTR Control VI to set the output of the counter to high impedance. Connect your signal to the counter output. Now, each time the output undergoes the proper state change, the VI sets the occurrence. Only devices with Am9513 counter/timer chips with certain counters support this event. For the MIO devices, these are the same counters that are used for timed analog output waveform generation, so you cannot use **DAQ event** 4 when generating waveforms. If you are using the AT-MIO-16F-5, the initial state of the counter output must be high or the VI sets the occurrence immediately. Table 20-3 lists the devices and counters that work with **DAQ event** 4.

5: Set the occurrence when data from any port in **channel** causes this statement to be true:data *and* **general value A** is *not equal* to **general value B.** You must have started a buffered digital operation (as with the Digital Buffer Control VI) that is interrupt-driven (i.e., the AT-DIO-32F is not supported).

6: Set the occurrence when data from any port in **channel** causes this statement to be true: data *and* **general value A** *equals* **general value B.** You must have started a buffered digital operation (as with the Digital Buffer Control VI) that is interrupt-driven (i.e., the AT-DIO-32F is not supported).

DAQ Devices Supporting DAQ Event 4

DAQ Device	Counter	Counter State Change
AT-MIO-16	2	low to high
AT-MIO-16D	2	low to high
AT-MIO-16F-5	1, 2, or 5	high to low
AT-MIO-16X	1, 2, or 5	high to low
AT-MIO-64F-5	1, 2, or 5	high to low
PC-TIO-10	any counter (specify EXTIRQ 1 or EXTIRQ 2)	high to low

AT-A2150, AT-DSP2200, and EISA-A2000 do not support **DAQ event** values 0 through 4.

level conditions is a cluster containing three controls that describe the conditions for **DAQ event** 3.

level A (Volts) specifies a voltage. For **DAQ event** 3, this voltage is a trigger level. The default input is 0.000 V.

level B (Volts) specifies a voltage. For **DAQ event** 3, this voltage is a window size. The default

input is 0.000 V.

slope specifies the direction in which the data must cross the trigger window to set the occurrence.

0: Either positive or negative.
1: Positive only. To set the occurrence, the data must go below **level A** - **level B** before crossing **level A** (default input).
2: Negative only. To set the occurrence, the data must go above **level A** + **level B** before crossing **level A**.

taskID out has the same value as **taskID in.**

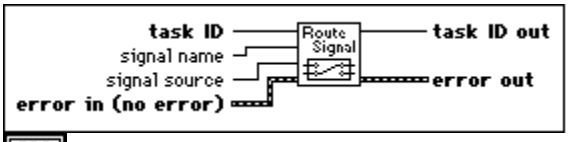
occurrence is the occurrence value this VI creates when you set **create/clear** to 1 (create a new occurrence**)**. Wire this output to the Wait on Occurrence function. Wire the output of Wait on Occurrence to the part of your diagram that you want to execute in response to the event. When you set **create/clear** to 0 (clear all occurrences**)** or the VI is unable to create the occurrence and returns an error code, LabVIEW returns the Not a Refnum file I/O constant.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Route Signal

Use this VI to route an internal signal to the specified I/O connector or RTSI bus line, or to enable clock sharing through the RTSI bus clock line.

Note: This VI is supported by E Series devices only.



taskID identifies the group and the I/O operation.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

signal name allows you to select the I/O connector line, RTSI line, or on-board clock line to drive. Click here to see a list of the possible **signal name** values.

signal source is the signal that LabVIEW routes to the location designated in signal name. There is only one valid **signal source** for most **signal names**. Next to each **signal source** is the valid **signal name**. Click here to see a list of the possible **signal source** values.

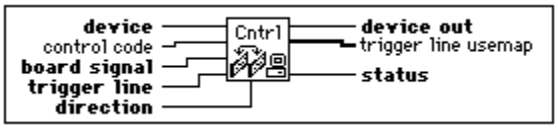
taskID out has the same value as **taskID in.**

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

RTSI Control (Macintosh and Windows)

Connects or disconnects trigger and timing signals between DAQ devices along the Real-Time System Integration (RTSI) bus.

This VI is not supported for E Series devices. For E Series devices, multiple RTSI connections can be set directly in the analog input, analog output, and counter VIs and used along with the Route Signal VI. Other RTSI connections must be made using the Route Signal VI.



I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **board signal** values depend on device type. Refer to the tables at the end of this VI description for the DAQ device signals you can connect to the RTSI bus trigger lines. This parameter defaults to 0.

I32 **trigger line** is the RTSI bus trigger line that is connected to the signal. The **trigger line** parameter can be 0 through 6. This parameter defaults to 0.

I32 **direction**.

0: The board receives the signal from the bus (default input).
1: The board transmits the signal to the bus.

If **control code** is 2, **direction** specifies whether the device receives the signal from the bus or transmits the signal to the bus. If **control code** is 3, **direction** must match the direction used upon connection.

I32 **control code**.

0: Do not change the **control code** setting (default input).
1: Clear.
2: Connect (default input).
3: Disconnect.
4: Construct the **trigger line usemap** only.

Set **control code** to 1 to clear all connections between the RTSI bus and the device identified by **device**, regardless of direction. Set **control code** to 2 to connect the device signal to the trigger line as identified by these two input parameters. Set **control code** to 3 to disconnect the device signal from the trigger line as identified by these two input parameters. If you set **control code** to 4, the VI returns **trigger line usemap** and does not change any RTSI connections.

I32 **device out** has the same value as **device**.

I32 **status** returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes, for a code description.

I32 **trigger line usemap** provides a list of free and busy RTSI trigger lines. If trigger line *i* is not busy, **trigger line usemap**[*i*] shows a value of 0. If trigger line *i* is busy, the VI sets **trigger line usemap**[*i*] to the device number of the device driving the line. Making only a receive connection to trigger line *i* does not set the *ith* element of **trigger line usemap**.

(Macintosh) The NB-MIO-16 contains nine signals that you can connect to the RTSI bus trigger. Click here to refer to specific information about the NB-MIO-16 RTSI bus trigger signals.

GATE1, OUT1, OUT2, OUT5, and FOUT are also available on the NB-MIO-16 I/O connector.

Some restrictions apply for connection of onboard signals to the RTSI bus. OUT2 and EXTCONV* should not be connected to the RTSI bus at the same time. Doing so results in the OUT2 signal driving the EXTCONV* line. Similarly, OUT5 and EXTGATE should not be connected to the RTSI bus at the same time. Doing so results in the OUT5 signal driving the EXTGATE line. For more information about the NB-MIO-16 signals, see the *NB-MIO-16 User Manual*.

(Macintosh) The NB-MIO-16X contains nine signals that you can connect to the RTSI bus trigger. Click here to refer to specific information about these RTSI bus signals.

GATE1, OUT1, OUT2, and OUT5 are also available on the NB-MIO-16X I/O connector.

Some restrictions apply for connection of onboard signals to the RTSI bus. OUT2 and GATE1 should not be connected to the RTSI bus at the same time. Doing so results in the OUT2 signal driving the GATE1 line. Similarly, OUT5 and STOPTRIG should not be connected to the RTSI bus at the same time. Doing

so results in the OUT5 signal driving the STOPTRIG line. For more information about the NB-MIO-16X signals, see the *NB-MIO-16X User Manual*.

(Macintosh) The NB-DMA-8-G and NB-DMA2800 contain 14 signals that you can connect to the RTSI bus trigger. Click here to refer to specific information about the NB-DMA-8-G or NB-DMA2800 RTSI bus signals.

For more information about the NB-DMA-8-G or NB-DMA2800 signals, see the NB-DMA-8-G User Manual or the *NB-DMA2800 User Manual.*

(Macintosh) The NB-DIO-32F contains six signals that you can connect to the RTSI bus trigger. Click here to refer to specific information about the NB-DIO-32F RTSI bus signals.

RQNI1, RQNI2, XAK1, and XAK2 permit handshaking with the NB-DIO-32F over the RTSI bus. For more information about the NB-DIO-32F signals, see the *NB-DIO-32F User Manual*.

(Macintosh) The NB-AO-6 contains two signals connected to the RTSI bus trigger Click here to refer to specific information about the NB-AO-6 RTSI bus signals.

For more information about the NB-AO-6 signals, see the *NB-AO-6 User Manual.*

(Macintosh) The NB-A2000 contains seven signals that can be connected to the RTSI bus trigger lines. Click here to refer to specific information about the NB-A2000 RTSI bus signals.

The START*, TRIGGER*, CLOCKO, and CLOCKI signals can be generated locally on the NB-A2000 or may be controlled from the RTSI bus.

Note: If the RTSI switch drives any of the START*, TRIGGER*, or CLOCKI signals, then locally generated signals are overwritten. TRIGGER* should be driven from the RTSI switch only if the NB-A2000 is configured for pretrigger mode.

(Macintosh) The NB-A2100 has three signals that you can connect to the RTSI bus trigger lines. Click here to refer to specific information about the NB-A2100 RTSI bus signals.

(Macintosh) The NB-A2150 has five signals that you can connect to the RTSI bus trigger lines. Click here to refer to specific information about the NB-A2150 RTSI bus signals.

(Macintosh) The NB-TIO-10 has 14 signals that you can connect to the RTSI bus trigger lines. All 14 signals are input and output signals from the Am9513 on the NB-TIO-10 Click here to refer to specific information about the NB-TIO-10 RTSI bus signals.

SOURCE1, SOURCE2, SOURCE6, and SOURCE7 are counter clock source inputs.

Warning: If you configure the output of a counter as an input over the RTSI bus, you must call CTR Control VI and put the counter back in high impedance mode to avoid multiple sources driving the signal and possibly causing damage to the board.

For more information about these signals, refer to the *NB-TIO-10 User Manual*.

(Windows) The MIO-16 devices contain nine signals you can connect to the RTSI bus trigger lines. 10 Click here to refer to specific information about the MIO-16 RTSI bus signals.

GATE1, OUT1, OUT2, OUT5, and FOUT are also available on the device I/O connector.

(Windows) The AT-MIO-16F-5 contain ten signals you can connect to the RTSI bus trigger lines. Click here to refer to specific information about these RTSI bus signals.The SYSTEM CLOCK signal is available only on the AT-MIO-64F-5 and AT-MIO-16X.

GATE1, OUT1, OUT2, OUT5, and FOUT are also available on the I/O connector of the AT-MIO-16F-5, AT-

MIO-64F-5, and AT-MIO-16X.

(Windows) The AT-AO-6/10 contains seven signals you can connect to the RTSI bus trigger lines. Click here to refer to specific information about the AT-AO-6/10 RTSI bus signals.

The EXTUPDATE* signal is shared with the I/O connector.

For more information about the AT-AO-6/10 signals, see the *AT-AO-6/10 User Manual*.

(Macintosh and Windows) The DIO-32F contains five signals you can connect to the RTSI bus trigger lines. Click here to refer to specific information about the DIO-32F RTSI bus signals.

For more information about the DIO-32F signals, see your *DIO-32F User Manual*.

(Windows) The EISA-A2000 contains eight signals you can connect to the RTSI bus trigger lines.

Click here to refer to specific information about the EISA-A20000 RTSI bus signals.

You can generate the START*, TRIGGER*, CLOCKO, and CLOCKI signals locally on the EISA-A2000 or control them from the RTSI bus.

Note: If the RTSI switch drives any of the START*, TRIGGER*, or CLOCKI signals, locally generated signals are overwritten. The RTSI switch should drive TRIGGER* only if the EISA-A2000 is configured for pretrigger mode.

(Windows) The AT-A2150 contains six signals you can connect to the RTSI bus trigger lines. Click here to refer to specific information about the AT-A2150 RTSI bus signals.

The HWTrig*, SWTrig*, RTSI_SWTrig* signals may be generated locally or controlled from the RTSI bus.

When synchronizing multiple AT-A2150s, take the following steps.

1. Call the Master Slave Config VI to specify the relationship.
2. If all AT-A2150s use a common timebase, call the RTSI Control VI to synchronize the ADC clock signals.
3. Connect trigger signals from the master to the slave.
   - For pretrigger mode, connect HWTrig* from the master to HWTrig* of the slave(s). Connect SWTrig* from the master to RTSI_SWTrig* of the slave(s).
   - For posttrigger mode or posttrigger with delay mode, connect HWTrig* from the master to HWTrig* of the slave(s).
- For software posttrigger mode, connect SWTrig* from the master to RTSI_SWTrig* of the slave(s).

To remove synchronization of multiple AT-A2150s, use the following steps.

1. Call the RTSI Control VI to disconnect trigger lines.
2. Call the Master Slave Config VI to remove the relationship.
3. If you used a common timebase, call the RTSI Control VI to disconnect the ADC clock signals.

(Windows) The AT-DSP2200 has four signals you can connect to the RTSI bus trigger lines. Click here to refer to specific information about the AT-DSP2200 RTSI bus signals.

The AT-DSP2200 can receive the HWTrig* signal from the RTSI bus and use the signal as a trigger, or the device can use its internal level-and-slope trigger circuit as a trigger and also send this trigger to other

boards through the RTSI bus.

When synchronizing multiple AT-DSP2200s, perform the following steps.

1. Call the Master Slave Config VI to specify the relationship.
2. If all AT-DSP2200s use a common timebase, call the RTSI Control VI to synchronize the ADC clock signals.

To remove synchronization of multiple AT-DSP2200s, perform the following steps:

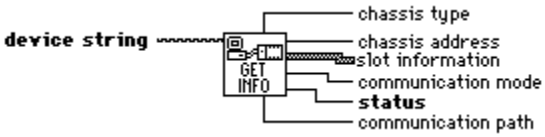
1. Call the Master Slave Config VI to remove the relationship.
2. If you used a common timebase, call the RTSI Control VI to disconnect the ADC clock signals.

Note: You can synchronize multiple AT-DSP2200s and AT-A2150s with the previous steps.

SCXI Calibration VIs

Get SCXI Information

Returns the SCXI chassis configuration information that you set using the configuration utility or the Set SCXI Information VI.



I32 **device string** specifies the chassis ID you want to use. The syntax requires the string SC or SCXI followed by the numerical chassis ID. For example, SC1 specifies the chassis with ID of 1. The default input is SCXI1.

I32 **status** returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes, for a code description.

I32 **chassis type** is a code that indicates what kind of SCXI chassis corresponds to the chassis ID specified by **device string**.

0: No chassis is configured for this ID.
1: SCXI-1000 4-slot chassis.
2: SCXI-1001 12-slot chassis.

I32 **chassis address** is the hardware-jumpered address of an SCXI chassis.

I32 **slot information** is an array containing a cluster for each slot of the chassis. Each cluster contains the following parameters.

I32 **module type** is a code indicating what kind of SCXI module is in the corresponding slot.

-1: The slot is empty.
2: SCXI-1121.
4: SCXI-1120.
6: SCXI-1100.
8: SCXI-1140.
10: SCXI 1122.
12: SCXI-1160.
14: SCXI-1161.
16: SCXI-1162.
18: SCXI-1163.
20: SCXI-1124.
24: SCXI-1162HV.
28: SCXI-1163R.
30: SCXI-1102.
32: SCXI-1141.
38: SCXI-1200. (Windows)

Any other value indicates a custom module that is not explicitly supported by this version of LabVIEW.

I32 **operating mode**. See the *SCXI Operating Modes* section of Chapter 17, *Hardware and Software Setup for Your SCXI System*, in the *LabVIEW Data Acquisition Basics Manual*, for more information on operating mode.

0: Multiplexed mode.
1: Parallel mode.

I32 **cabled device** is the device number of the DAQ device that is cabled to the corresponding

module. If you have not cabled a device directly to the module, **cabled device** is 0.

(Windows) If the module in the corresponding slot is an SCXI-1200, cabled device is the logical device number of the SCXI-1200.

I32 **communication mode** indicates how LabVIEW communicates with the chassis.

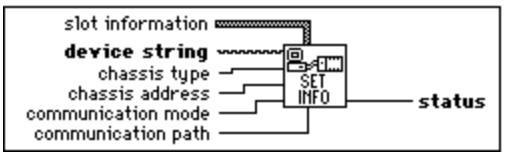
0: Communication disabled.
1: LabVIEW uses a digital port of a DAQ device that is cabled to a module in the chassis to communicate serially with the chassis.
2: LabVIEW uses a parallel port of the PC that is cabled to the SCXI-1200 module to communicate with the chassis. (Windows)

I32 **communication path**. When **communication mode** is 1, **communication path** is the device number of the DAQ device whose digital port LabVIEW uses to communicate with the chassis. You must cable this DAQ device to a module in the chassis.

(Windows)When communication mode is 2, communication path is the logical device number of the SCXI-1200 module.

Set SCXI Information

Sets the SCXI chassis configuration information.



Use this VI to override the configuration already set with the configuration utility You can use this VI *instead* of using the configuration utility to enter the chassis configuration information. If you do not use this VI, the first VI that accesses an SCXI chassis automatically tries to load information from the configuration file.

I32 **device string** indicates the chassis ID you want to use. The syntax requires the string SC or SCXI followed by the numerical chassis ID. For example, SC1 specifies the chassis with ID of 1. This parameter defaults SCXI1.

I32 **slot information** is an array containing a cluster for each slot of the chassis. Each cluster contains the following elements.

I32 **module type** is a code indicating what kind of SCXI module is in the corresponding slot.

-1: The slot is empty.
2: SCXI-1121.
4: SCXI-1120.
6: SCXI-1100.
8: SCXI-1140.
10: SCXI-1122
12: SCXI-1160.
14: SCXI-1161.
16: SCXI-1162.
18: SCXI-1163.
20: SCXI-1124.
24: SCXI-1162HV.
28: SCXI-1163R.
30: SCXI-1102.
32: SCXI-1141.
38: SCXI-1200. (Windows)

Any other value indicates a custom module that this version of LabVIEW does not

explicitly support.

I32 **operating mode**.

0: Multiplexed mode.
1: Parallel mode.

I32 **cabled device** is the DAQ device number that is cabled to the corresponding module. If you have not cabled a device directly to the module, **cabled device** is 0.

(Windows) If the module in the corresponding slot is an SCXI-1200, **cabled device** is the logical device number of the SCXI-1200.

I32 **chassis type** code indicates the kind of SCXI chassis you want to configure. The **device string** parameter specifies the **chassis type** code.

0: No chassis.
1: SCXI-1000 4-slot chassis (default).
2: SCXI-1001 12-slot chassis.

I32 **chassis address** is the hardware-jumpered address of an SCXI chassis. The default address is 0.

I32 **communication mode** indicates the method of communication with the chassis.

0: Disable communication. This option disables the chassis.
1: LabVIEW uses a digital port of a DAQ device cabled to a module in the chassis to communicate serially with the chassis (default).
2: LabVIEW uses a parallel port of the PC that is cabled to the SCXI-1200 module to communicate with the chassis. (Windows)

I32 **communication path**. When **communication mode** is 1, **communication path** is the DAQ device number whose digital port LabVIEW uses to communicate with the chassis. You must cable this DAQ device to a module in the chassis. This parameter defaults to 0.

(Windows)When **communication mode** is 2, **communication path** is the logical device number of the SCXI-1200 module.

I32 **status** returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes, for a code description.

SCXI Cal Constants

Calculates calibration constants for the given channel and range or gain using measured voltage/binary pairs. You can use this VI with any SCXI module.

I32

For information about SCXI calibration, refer to Chapter 20, *SCXI CalibrationIncreasing Signal Measurement Precision*, in the *LabVIEW Data Acquisition Basics Manual*.

I32 **TB Gain** is the gain applied to the SCXI channel, if any. For terminal blocks that do not apply gain to your SCXI channels, set the TB Gain to 1.0.

I32 **Cal Constant In 1** is the first calibration constant to be stored in the calibration area, only when Op Code is 3.

I32 **Volt/Amp 2** is the second voltage you apply at the input channel to take the binary reading (Binary 2) or the voltage/current you measured at the output channel after writing a binary value (Binary 2).

I32 **Volt/Amp 1** is the first voltage you apply at the input channel to take the binary reading (Binary 1) or the voltage/current you measured at the output channel after writing a binary value (Binary 1).

I32 **SCXI Chassis Cluster** identifies the SCXI chassis, module, and channel to be calibrated.

I32 **SCXI Chassis ID** is the chassis ID assigned during configuration.

I32 **Module Slot** is the module slot number in the SCXI chassis.

I32 **Channel** is the channel number on the module.

0 to n-1: Where n is the number of channels available on the module.
-1: All channels on the module (SCXI-1100 and SCXI-1122 only).
-2: The voltage and current excitation channels on the module (SCXI-1122 and when **Op Code**=0 only).

I32 **Task ID** identifies the group and the I/O operation.

I32 **Op Code** specifies the type of calibration to be performed.

0: Retrieve calibration constants for the given channel and range or gain from calibration area and return them in **Cal Constant Out 1** and **Cal Constant Out 2**.
1: Perform a one-point offset calibration calculation using (**Volt/Amp 1**, **Binary 1**) for the given channel and range or gain and write calibration constants to calibration area.
2: Perform a two-point calibration calculation using (**Volt/Amp 1**, **Binary 1**) and (**Volt/Amp 2**, **Binary 2**) for the given channel and range or gain and write calibration constants to calibration area.
3: Write the calibration constants passed in **Cal Constant In 1** and **Cal Constant In 2** to calibration area for the given channel and range or gain.
4: Copy the entire calibration table in calibration area to the module EEPROM default load area so that it will be loaded automatically into memory during subsequent application runs.
5: Copy the entire calibration table in calibration area to memory so the table can be used in subsequent scaling operations.

I32 **Cal Area** is the location LabVIEW uses to retrieve or store the calibration constants.

0: NI-DAQ memory. A calibration table is maintained in memory for subsequent scaling operations for the module.
1: Default EEPROM load area. (SCXI-1102, SCXI-1122, SCXI-1124 and SCXI-1141 only).
2: Factory EEPROM area. You cannot write to this area, but you can read or copy from it (SCXI-1102, SCXI-1122, SCXI-1124 and SCXI-1141 only).
3: User EEPROM area (SCXI-1102, SCXI-1122, SCXI-1124 and SCXI-1141 only).

I32 **Range Code** is the voltage or current range of the analog output channel. Applies to analog output modules only.

0: 0 to 1V.
1: 0 to 5V.
2: 0 to 10V.
3: -1 to 1V.
4: -5 to 5V.
5: -10 to 10V.
6: 0 to 20mA.

I32 **SCXI Gain** is the SCXI module or channel gain setting. Applies to analog input modules only.

SCXI-1100: 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000.
SCXI-1102: 1, 100.
SCXI-1120: 1, 2, 5, 10, 20, 50, 100, 200, 250, 500, 1000, 2000.
SCXI-1121: 1, 2, 5, 10, 20, 50, 100, 200, 250, 500, 1000, 2000.
SCXI-1122: 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000.
SCXI-1140: 1, 10, 100, 200, 500.
SCXI-1141: 1, 2, 5, 10, 20, 50, 100.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **DAQ Board Cluster** contains the setup information of the DAQ board that you are using to take

measurements. You must use these same settings whenever you use the new calibration constants.

I32 **DAQ Board** is the DAQ board device number you used to take your volt/binary measurements for an analog input module channel, when Op Code is 0, 1, 2, or 3.

I32 **DAQ Chan** is the DAQ board channel number you use to take your volt/binary measurements for an analog input module channel, when Op Code is 0, 1, 2, or 3.

I32 **DAQ Gain** is the DAQ board gain code you used to take your volt/ binary measurements for an analog input module channel, when Op Code is 0, 1, 2, or 3.

I32 **Binary 1** is the binary value you read from the input channel with a known voltage of Volt/Amp 1 applied at the input or the binary value you write to the output channel to measure the voltage or current Volt/Amp 1.

I32 **Binary 2** is the binary value you read from the input channel with a known voltage of Volt/Amp 2 applied at the input or the binary value you write to the output channel to measure the voltage or current Volt/Amp 2.

I32 **Cal Constant In 2** is the second calibration constant to be stored in the calibration area, only when Op Code is 3.

I32 **TaskID Out** has the same value as **taskID in.**

DBL **Cal Constant Out 1** is the first calibration constant. For analog output modules, **Cal Constant Out 1** is the binary value that will generate the voltage or current at the lower end of the voltage or current range. For analog input modules, **Cal Constant Out 1** is the binary reading that would result from an input voltage of 0(binary zero offset). If **Op Code** is 0 and **Channel** is -2, **Cal Constant Out 1** is a return current constant retrieved from the calibration area. If **Op Code** is 1 or 2, **Cal Constant Out 1** is a return value calculated from the voltage/binary pairs.

DBL **Cal Constant Out 2** is the second calibration constant. For analog output modules, **Cal Constant Out 2** is the binary value that will generate the voltage or current at the upper end of the voltage or current range. For analog input modules, **Cal Constant Out 2** is the ratio of the real gain to the ideal gain setting (gain adjust factor). If **Op Code** and **Channel** are -2, **Cal Constant Out 2** is a return voltage constant retrieved from the calibration area. If **Op Code** is 1 or 2, **Cal Constant Out 2** is a return value calculated from the voltage/binary pairs.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Get DAQ Device Information VI Valid Information Types and Strings

Information Type	Information String
0: NI-DAQ Version	Returns the version number of NI-DAQ (i.e., for version 4.6.1, the version reads 0x461).
1: device name	Returns the name of the device.
2: base address	Returns a string, which gives the hexadecimal address (i.e., 0x220 is 220 Hex).
3-9: all the data transfer modes	There are three possible strings that can be returned: interrupts, up to 1 DMA channel, and up to 2 DMA channels.
10: device type code	Returns a string that contains a number (i.e., 0 or 1). The number depends on the type of device. Refer to the *NI-DAQ Software Reference Manual* for a list of the device codes, which are called device NumberCodes in NI-DAQ.
11-15: DMA and interrupt levels	Returns a string that contains a number (i.e., 5 or 6). This number indicates which DMA channel or interrupt level your device is using. When the interrupt level is not appropriate for your device, the program returns a not applicable string.
16: interrupt trigger mode	Returns one of two possible strings: edge sensitive and level sensitive.
17: lpt device mode	
18: DAQCard-700 counter source	

Possible Legal Combinations of information type and information setting

	Information Setting						
Information Type	**0:Interrupts**	**1:Up to 1 Channel**	**2:Up to 2 Channels**	**3:Disable**	**4:Enable**	**5:Internal Timer**	**6:I/O Connector**
0: Data Transfer Mode for Analog Input							
1: Data Transfer Mode for Analog Output Group 1							
2: Data Transfer Mode for Analog Output Group 2							
3: Data Transfer Mode for General Purpose Counter 0							
4: Data Transfer Mode for General Purpose Counter 1							
5: Data Transfer Mode for Digital I/O Group 1							
6: Data Transfer Mode for Digital I/O Group 2							
7: LPT Device Mode							
8: DAQCard-700 Counter 1 Source							

Channel to Index VI Parameter Examples for Macintosh and Windows

channel scan list	channel list	channel indices
1, 3, 4, 5, 7	channel list[0] = 5	channel indices[0] = 3. Data for channel 5 is at position 3 within a scan. Indices are zero-based.
1, 3, 4, 5, 7	channel list is of 0 length.	channel indices is of 0 length. (In this case, status is non-zero.)
1, 2, 1, 3, 1, 4 (The device samples channel 1 three times during a scan.)	channel list[0] = 1,1,1	channelindices[0] = 0, channelindices[1] = 2, and channelindices[2] = 4. The first occurrence of channel 1 within a scan is

		at index 0, the second at index 2, and the third at index 4
0, 1, 3, 4 (For this example, channel scan list is a digital input group.)	channel list[0] = 3	channel indices[0] = 2. The eight bits of data from port 3 are at index 2 in the scan list.
0:3 (One AMUX-64T in use.)	channel list[0] = AM1!9	channel indices[0] = 9. Data obtained from channel 9 on AMUX-64T device number 1 is at index 9 in the data buffer.
SC1!MD1!CH0:7, SC1!MD2!CH0:4	channel list[0] = SC1!MD2!CH3	channel indices[0] = 11. Data obtained from channel 3 of the SCXI module in slot 2 is at index 11 in the data buffer.

Click here to see the possible values for the **channel scan list**, **channel list**, and **channel indices** parameters for the Sun.

Channel to Index VI Parameter Examples for Sun

channel scan list	channel list	channel indices
1, 3, 4, 5, 7	channel list[0] = 5	channel indices[0] = 3. Data for channel 5 is at position 3 within a scan. Indices are zero-based.
1, 3, 4, 5, 7	channel list is of 0 length.	channel indices is of 0 length. (In this case, status is non-zero.)
1, 2, 1, 3, 1, 4 (The device samples channel 1 three times during a scan.)	channel list[0] = 1,1,1	channel indices[0] = 0, channel indices[1] = 2, and channel indices[2] = 4. The first occurrence of channel 1 within a scan is at index 0, the second at index 2, and the third at index 4

Route Signal VI Possible Signal Name Values

0: Do not change signal name (default input).
1: PFI 0.
2: PFI 1.
3: PFI 2.
4: PFI 3.
5: PFI 4.
6: PFI 5.
7: PFI 7.
8: PFI 8.
9: PFI 9.
10: GPCTR0 Output.
11: RTSI 0.
12: RTSI 1.
13: RTSI 2.
14: RTSI 3.
15: RTSI 4.
16: RTSI 5.
17: RTSI 6.
18: RTSI clock.
19: Board clock.

Route Signal VI Possible Signal Source Values

0: Do not change signal source (default input).
1: None (default setting).
2: AI Start Trigger.
3: AI Stop Trigger.
4: AI convert.
5: GPCTR1 Source.
6: GPCTR1 Gate.
7: AO Update.
8: AO Start Trigger.
9: AI scan start.
10: GPCTR0 Source.
11: GPCTR0 Gate.
12: GPCTR0 Output.
13: RTSI 0.
14: RTSI 1.
15: RTSI 2.
16: RTSI 3.
17: RTSI 4.
18: RTSI 5.
19: RTSI 6.
20: RTSI clock.
21: Board clock

NB-MIO-16 RTSI Bus Trigger Signals

Signal Name	Signal Direction	Signal Code	Signal Description
GATE1	Bidirectional	0	Gating signal from the Am9513 Counter/Timer for Counter 1.
FOUT	Source	1	Programmable frequency output signal from the Am9513 Counter/Timer.
EXTCONV*	Receiver	2	The channel (sample) clock, which can be used as a receiver or a transmitter signal. As a receiver, you take another RTSI signal and use it as an external sample clock. As a transmitter, you can send on the channel (sample) clock to another RTSI receiver on the RTSI bus.
OUT2	Source	3	Output signal from the Am9513 Counter/Timer for Counter 2.
SOURCE4	Bidirectional	4	Counter source input from the Am9513 Counter/Timer for Counter 4.
EXTGATE	Receiver	5	A level-triggered input signal, which acts as the the external gate input. When this signal is low, it holds off data acquisition timing. This signal can only be setup as a receiver. As a receiver, a logic high signal into this pin would allow the data acquisition timing circuitry to take samples, and a logic low signal would prevent taking samples.
OUT5	Source	6	Output signal from the Am9513 Counter/Timer for Counter 5.
OUT1	Bidirectional	7	Output signal from the Am9513 Counter/Timer for Counter 1.
EXTTRIG*	Bidirectional	8	An edge-triggered signal that initiates a data acquisition sequence. As a receiver, a falling edge into this pin would cause a post-triggered data acquisition to start. As a transmitter, you can send on the trigger signal that either comes from the I/O connector, or comes as a result of starting the post-triggered acquisition, to another RTSI receiver on the RTSI bus.

NB-MIO-16X RTSI Bus Signals

Signal Name	Signal Direction	Signal Code	Signal Description
EXTCONV*	Bidirectional	0	Signal on the I/O connector that is supplied once every sample interval, causing an A/D conversion to be initiated.
RTSIWG	Receiver	1	
GATE1	Receiver	2	Gating signal from the Am9513 Counter/Timer for Counter 1.
OUT2	Source	3	Output signal from the Am9513

			Counter/Timer for Counter 2.
SOURCE5	Bidirectional	4	Counter source input from the Am9513 Counter/Timer for Counter 5.
STOPTRIG	Receiver	5	The stop trigger signal, which can only be setup as a receiver. A rising edge into this pin would cause a pre-triggered data acquisition to stop.
OUT5	Source	6	Output signal from the Am9513 Counter/Timer for Counter 5.
OUT1	Bidirectional	7	Output signal from the Am9513 Counter/Timer for Counter 1.
STARTTRIG*	Bidirectional	8	The start trigger signal, which can be used as a receiver or transmitter. A falling edge into this pin would cause a pre- or post-triggered data acquisition to start. As a transmitter, you can send on the trigger signal that either comes from the I/O connector, or comes as a result of starting the pre- or post-triggered acquisition, to another RTSI receiver on the RTSI bus.

NB-DMA-8-G and NB-DMA2800 RTSI Bus Signals

Signal Name	Signal Direction	Signal Code	Signal Description
FOUT	Source	0	Programmable frequency output signal from the Am9513 Counter/Timer.
SOURCE5	Receiver	1	Counter source input from the Am9513 Counter/Timer for Counter 5.
SOURCE4	Receiver	2	Counter source input from the Am9513 Counter/Timer for Counter 4.
SOURCE3	Receiver	3	Counter source input from the Am9513 Counter/Timer for Counter 3.
GATE5	Receiver	4	Gating signal from the Am9513 Counter/Timer for Counter 5.
GATE3	Receiver	5	Gating signal from the Am9513 Counter/Timer for Counter 3.
GATE1	Receiver	6	Gating signal from the Am9513 Counter/Timer for Counter 1.
TOUT3	Source	7	Signal that supplies a 5 MHz clock signal.
TOUT2	Source	8	Signal that supplies a 1 MHz clock signal.
OUT5	Source	9	Output signal from the Am9513 Counter/Timer for Counter 5
OUT4	Source	10	Output signal from the Am9513 Counter/Timer for Counter 4
OUT3	Source	11	Output signal from the Am9513 Counter/Timer for Counter 3

OUT2	Source	12	Output signal from the Am9513 Counter/Timer for Counter 2
OUT1	Source	13	Output signal from the Am9513 Counter/Timer for Counter 1.

NB-DIO-32F RTSI Bus Signals

Signal Name	Signal Direction	Signal Code	Signal Description
RRQ1	Source	0	Request signals received from the I/O connector. An external device drives these signals during handshaking.
RRQ2	Source	1	Request signals received from the I/O connector. An external device drives these signals during handshaking
RQNI1	Receiver	2	Handshaking line for the NB-DIO-32F over the RTSI bus.
RQNI2	Receiver	3	Handshaking line for the NB-DIO-32F over the RTSI bus.
RRQ2	Source	4	Handshaking line for the NB-DIO-32F over the RTSI bus.
RQNI1	Source	5	Handshaking line for the NB-DIO-32F over the RTSI bus.

NB-AO-6 RTSI Bus Signals

Signal Name	Signal Direction	Signal Code	Signal Description
TRIGUP	Receiver	0	Signal used to trigger an update on the NB-AO-6 for updating of the DACs
UPDATE	Source	1	UPDATE is the update signal generated.

NB-A2000 RTSI Bus Signals

Signal Name	Signal Direction	Signal Code	Signal Description
START*	Bidirectional	0	START* is an active-low signal that initiates a data acquisition sequence. If data acquisition is locally controlled, the START* signal pulses low when a trigger is generated from software (pretrigger mode) or from the NB-A2000 analog trigger or digital trigger circuitry (posttrigger or pretrigger mode). When locally generated, START* is a signal source. Alternatively, if the NB-A2000 is configured to receive START* from the RTSI bus, it initiates a data acquisition sequence when a low pulse is received.
TRIGGER*	Bidirectional	1	TRIGGER* is an active-low signal that activates the sample counter. If data acquisition is locally controlled,

			TRIGGER* pulses low when a trigger is received either through software (posttrigger mode only) or from the NB-A2000 analog trigger or digital trigger circuitry (posttrigger or pretrigger mode). In pretrigger mode, TRIGGER* pulses sometime after START*. In posttrigger mode, START* drives the TRIGGER* signal directly. The TRIGGER* signal may be driven from the RTSI switch when the NB-A2000 is configured for pretrigger mode only. In this case, the NB-A2000 begins to acquire posttrigger samples after a low pulse is received on the TRIGGER* input rather than from the local analog or digital circuitry.
CLOCKO	Source	2	CLOCKO is the active-high, sample-clock output signal. The rising edge of this signal initiates a scanning sequence in which all active channels are sampled simultaneously. Any locally generated sample clock or any clock received from the I/O connector SAMPCLK* input will appear on the CLOCKO signal.
CLOCKI	Receiver	3	CLOCKI is the active-high, sample-clock input signal. If CLOCKI is driven from the RTSI Switch, the rising edge of this signal initiates a scanning sequence in which all active channels are sampled simultaneously. The locally generated sample clock and I/O connector SAMPCLK* signal are ignored when CLOCKI is driven by the RTSI switch. With the exception of master/slave clock operation, the NB-A2000 must be configured to use external sample clock if CLOCKI is to be driven from the RTSI switch (see the A2000 Configure VI).
GATE2	Receiver	4	Gating signal from the Am9513 Counter/Timer for Counter 2.
SOURCE2	Receiver	5	Counter source input from the Am9513 Counter/Timer for Counter 2
OUT2	Source	6	Output signal from the Am9513 Counter/Timer for Counter 2

NB-A2100 RTSI Bus Signals

Signal Name	Signal Direction	Signal Code	Signal Description
EXTTRIG*	Bidirectional	0	The RTSI bus trigger lines share the signal EXTTRIG* with the RCA jack on the I/O connector panel. So, if you configure EXTTRIG* as a source, the signal applied at the I/O connector appears at the RTSI trigger lines. Similarly, if you configure EXTTRIG* as a receiver, the signal applied at EXTTRIG*

			through a RTSI trigger line also appears at the I/O connector.
WCAD	Source	1	The A/D and D/A sampling clock signals.
WCDA	Source	2	The A/D and D/A sampling clock signals.

NB-A2150 RTSI Bus Signals

Signal Name	Signal Direction	Signal Code	Signal Description
RTSITRIG*	Bidirectional	0	RTSITRIG* is the signal that activates the sample counter when a high-to-low edge is received at the signal. In posttrigger mode, a high-to-low edge starts the data acquisition on the NB-A2150. In pretrigger mode, the NB-A2150 begins to acquire posttrigger samples after a high-to-low pulse is received. Alternatively, if the RTSITRIG* signal is configured to be the source, it is connected to the local trigger signal of the board. The local trigger signal is connected to the EXTTRIG* pin on the I/O connector if the external digital trigger is enabled or to the analog trigger circuitry if analog level triggering is enabled. The analog triggering circuitry generates an active low pulse when the analog trigger conditions are met.
WCAD	Source	1	The A/D word clock signal.
RTSI_A2	Source	2	This signal is register-controlled and is always low.
SWSTART*	Source	3	SWSTART* is the signal that generates an active low pulse when a trigger is generated from software (pretrigger or posttrigger mode). This signal can be driven to the RTSISTART* signal on other NB-A2150 boards to simultaneously start data acquisition on all boards when a software trigger is generated at the board that is driving this signal.
RTSISTART*	Receiver	4	RTSISTART* is the signal that starts the data acquisition when a high-to-low edge is received at the signal. In posttrigger mode, this signal has the same effect as the RTSITRIG* signal. In pretrigger mode, a high-to-low edge at this signal starts continuous data acquisition. However, the sample counter is not started until a trigger is received from the RTSITRIG* signal, the EXTTRIG* pin on the I/O connector, or the analog triggering circuitry.

NB-TIO-10 RTSI Bus Signals

Signal Name	Signal Direction	Signal Code	Signal Description

SOURCE1	Bidirectional	0	Counter source input from the Am9513 Counter/Timer for Counter 1.
GATE1	Bidirectional	1	Gating signal from the Am9513 Counter/Timer for Counter 1.
OUT1	Bidirectional	2	Output signal from the Am9513 Counter/Timer for Counter 1.
SOURCE2	Bidirectional	3	Counter source input from the Am9513 Counter/Timer for Counter 2.
OUT2	Bidirectional	4	Output signal from the Am9513 Counter/Timer for Counter 2.
GATE5	Bidirectional	5	Gating signal from the Am9513 Counter/Timer for Counter 5.
OUT5	Bidirectional	6	Output signal from the Am9513 Counter/Timer for Counter 5.
SOURCE6	Bidirectional	7	Counter source input from the Am9513 Counter/Timer for Counter 6.
GATE6	Bidirectional	8	Gating signal from the Am9513 Counter/Timer for Counter 6.
OUT6	Bidirectional	9	Output signal from the Am9513 Counter/Timer for Counter 6.
SOURCE7	Bidirectional	10	Counter source input from the Am9513 Counter/Timer for Counter 7.
GATE10	Bidirectional	11	Gating signal from the Am9513 Counter/Timer for Counter 10.
OUT10	Bidirectional	12	Output signal from the Am9513 Counter/Timer for Counter 10.
FOUT1	Source	13	Programmable frequency output signal from the Am9513 Counter/Timer.

AT-MIO-16 and AT-MIO-16D RTSI Bus Signals

Signal Name	Signal Direction	Signal Code	Signal Description
EXTCONV*	Bidirectional	0	Input signal that indicates whether the timing of A/D conversions during the DAQ sequence is controlled externally or internally with the sample-interval and/or scan-interval clocks.
FOUT	Source	1	Programmable frequency output signal from the Am9513 Counter/Timer.
OUT2	Source	2	Output signal from the Am9513 Counter/Timer for Counter 2.
GATE1	Receiver	3	Gating signal from the Am9513 Counter/Timer for Counter 1.
SOURCE5	Receiver	4	Counter source input from the Am9513 Counter/Timer for Counter 5.
OUT5	Source	5	Output signal from the Am9513

			Counter/Timer for Counter 5.
STOP TRIG	Receiver	6	Input signal that indicates whether to enable or disable the pretriggered mode of data acquisition.
OUT1	Source	7	Output signal from the Am9513 Counter/Timer for Counter 1.
START TRIG*	Bidirectional	8	Input signal that indicates whether the trigger to initiate DAQ sequences is generated externally.

AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X RTSI Bus Signals

Signal Name	Signal Direction	Signal Code	Signal Description
EXTCONV*	Bidirectional	0	Input signal that indicates whether the timing of A/D conversions during the DAQ sequence is controlled externally or internally with the sample-interval and/or scan-interval clocks.
FOUT	Source	1	Programmable frequency output signal from the Am9513 Counter/Timer.
OUT2	Source	2	Output signal from the Am9513 Counter/Timer for Counter 2.
GATE1	Receiver	3	Gating signal from the Am9513 Counter/Timer for Counter 1.
SOURCE5	Bidirectional	4	Counter source input from the Am9513 Counter/Timer for Counter 5.
OUT5	Source	5	Output signal from the Am9513 Counter/Timer for Counter 5.
DACUPTRIG*	Receiver	6	The DAQUPTRIG* signal and one of the OUTx signals (usually OUT5) are for waveform generation.
OUT1	Bidirectional	7	Output signal from the Am9513 Counter/Timer for Counter 1.
EXTTRIG*	Bidirectional	8	
SYSTEM CLOCK	Bidirectional	10	

AT-AO-6/10 RTSI Bus Signals

Signal Name	Signal Direction	Signal Code	Signal Description
OUT0*	Source	0	Output signal from the MSM82C53 Counter/Timer for Counter 0.
GATE2	Receiver	1	Gating signal from the MSM82C53 Counter/Timer for Counter 2.
EXTUPD*	Source	2	Externally updates selected DACs.
OUT2*	Source	3	Output signal from the MSM82C53 Counter/Timer for Counter 2.

OUT1*	Source	4	Output signal from the MSM82C53 Counter/Timer for Counter 1.
EXTUPDATE*	Bidirectional	5	Externally updates selected DACs.
SYSTEM CLOCK	Bidirectional	10	

DIO-32F RTSI Bus Signals

Signal Name	Signal Direction	Signal Code	Signal Description
REQ1	Receiver	0	Request signals received from the I/O connector. An external device drives these signals during handshaking.
REQ2	Receiver	1	Request signals received from the I/O connector. An external device drives these signals during handshaking.
ACK1	Source	2	Supplied for handshaking with the DIO-32F over the RTSI bus.
ACK2	Source	3	Supplied for handshaking with the DIO-32F over the RTSI bus.

EISA-A2000 RTSI Bus Signals

Signal Name	Signal Direction	Signal Code	Signal Description
START*	Bidirectional	0	START* is an active-low signal that initiates a data acquisition sequence. If data acquisition is locally controlled, the START* signal pulses low when a trigger is generated from software or from the EISA-A2000 analog trigger or digital trigger circuitry (posttrigger or pretrigger mode). When locally generated, START* is a signal source. Alternatively, if the EISA-A2000 is configured to receive START* from the RTSI bus, a data acquisition sequence is initiated when the board receives a low pulse.
TRIGGER*	Bidirectional	1	TRIGGER* is an active-low signal that activates the sample counter. If data acquisition is locally controlled, TRIGGER* pulses low when the board receives a trigger either through software (posttrigger mode only) or from the EISA-A2000 analog trigger or digital trigger circuitry (posttrigger or pretrigger mode). In pretrigger mode, TRIGGER pulses sometime after START*. In posttrigger mode, START* drives the TRIGGER* signal directly. The RTSI switch may drive the TRIGGER* signal when the EISA-A2000 is configured for pretrigger mode only.
CLOCKO	Source	2	CLOCKO is the active-high, sample-clock output signal. The rising edge of this signal initiates a scanning sequence in which all active channels are simultaneously sampled. Any locally generated sample clock or any clock received from the I/O

			connector SAMPCLK* signal can drive CLOCKO.
CLOCKI	Receiver	3	CLOCKI is the active-high, sample-clock input signal. If the RTSI switch drives CLOCKI, the rising edge of this signal initiates a scanning sequence in which all active channels are simultaneously sampled. The locally generated sample clock and I/O connector SAMPCLK* signal are ignored when the RTSI switch drives CLOCKI. The EISA-A2000 must use an external sample clock if the RTSI switch drives CLOCKI (see the A2000 Configure VI).
GATE2	Receiver	4	Gating signal from the Am9513 Counter/Timer for Counter 2.
SOURCE2	Receiver	5	Counter source input from the Am9513 Counter/Timer for Counter 2.
OUT2	Source	6	Output signal from the Am9513 Counter/Timer for Counter 2.
SYSTEM CLOCK	Bidirectional	10	

AT-A2150 RTSI Bus Signals

Signal Name	Signal Direction	Signal Code	Signal Description
HWTrig*	Bidirectional	0	HWTrig* is the digital trigger signal. When configured as a source, it can transmit the digital signal from the external I/O connector or the signal generated by the internal level-and-slope trigger circuit. When used as a receiver, the AT-A2150 can act on this signal as the trigger when configured for pretrigger, posttrigger, or posttrigger with delay mode. Only one board (the master) should have this line configured as a source. All other boards using this signal should be slaves.
WCAD	Source	1	The conversion pulse signal that clocks the ADC.
RTSITrig*	Source	2	The AT-A2150 generates this signal by writing to the RTSI Trigger Register. Currently, LabVIEW does not have this functionality.You must explicitly write to the RTSI Trigger Register.
SWTrig*	Source	3	SWTrig* is a signal generated when a pretrigger, or software posttrigger acquisition has started. This can be connected to RTSI_SWTrig* on another AT-A2150 to initiate simultaneous acquisitions.
RTSI_SWTrig*	Receiver	4	When RTSI_SWTrig* receives a signal, it has the same effect as writing to the A/D FIFO Start Register on the AT-A2150. This signal can start a pretrigger or software posttrigger acquisition.

SYSTEM CLOCK	Bidirectional	10

AT-DSP2200 RTSI Bus Signals

Signal Name	Signal Direction	Signal Code	Signal Descriptions
HWTrig*	Bidirectional	0	The digital trigger signal.
WCAD	Source	1	The conversion pulse signal that clocks the ADC.
RTSITrig*	Source	2	A signal generated by writing to the RTSI Trigger Register on the AT-DSP2200. Currently, LabVIEW does not have this functionality. You must explicitly write to the RTSI Trigger Register.
SYSTEM CLOCK	Bidirectional	10	

Device Reset VI

Device Reset

Get DAQ Device Information VI

Get DAQ Device Information

Set DAQ Device Information VI

Set DAQ Device Information (Macintosh and Windows)

1200 Calibrate VI

1200 Calibrate

A2000 Calibrate VI

A2000 Calibrate VI

A2100 Calibrate VI

A2100 Calibrate (Macintosh)

A2150 Calibrate VI

A2150 Calibrate (Windows)

A2200 Calibrate VI

A2200 Calibrate (Sun)

AO-6/10 Calibrate VI

AO-6/10 Calibrate (Windows)

DSP2200 Calibrate VI

DSP2200 Calibrate (Windows)

E-Series Calibrate VI

E-Series Calibrate (Windows and Sun)

LPM-16 Calibrate VI

LPM-16 Calibrate (Macintosh and Windows)

MIO Calibrate VI

MIO Calibrate (Windows)

A2000 Configure VI

A2000 Configure (Macintosh and Windows)

A2100 Configure VI

A2100 Config (Macintosh)

A2150 Config VI

A2150 Config (Macintosh)

DSP2200 Configure VI

DSP2200 Configure (Windows)

Master Slave Config VI

Master Slave Config (Macintosh and Windows)

MIO Configure VI

MIO Configure (Windows and Sun)

Channel To Index VI

Channel To Index

DAQ Occurrence Config VI

DAQ Occurrence Config (Windows)

Route Signal VI

Route Signal

RTSI Control VI

RTSI Control (Macintosh and Windows)

Get SCXI Information VI

Get SCXI Information

Set SCXI Information VI

Set SCXI Information

SCXI Cal Constants VI

SCXI Cal Constants

Data Acquisition Common Questions

Where is the best place to get up to speed quickly with data acquisition and LabVIEW?

Read the LabVIEW Data Acquisition Basics Manual and look at the run_me.llb examples (in examples->daq) included with the package.

What is the easiest way to address my AMUX-64T board with my MIO board?

Set the number of AMUX boards used in the configuration utility (wdaqconf.exe on Windows or NI-DAQ control panel on Macintosh). Then in the channel string inputs specify the onboard channel. For example, with one AMUX-64T board, the channel string 0:1 will acquire data from AMUX channels 0 through 7, and so on.

What are the advantages/disadvantages of reading AI Read's backlog rather than a fixed amount of data?

Reading the backlog is guaranteed not to cause a synchronous wait for the data to arrive. However, it adds more delay until the data is processed (because the data was really available on the last call) and it can require constant reallocation or size adjustments of the data acquisition read buffer in LabVIEW.

How can I tell when a continuous data acquisition operation does not have enough buffer capacity?

The scan backlog rises with time, either steadily or in jumps, or takes a long time to drop to normal after an interrupting activity like mouse movement. If you can open another VI during the operation without receiving an overrun error you should have adequate buffer capacity.

I want to group two or more ports using my DIO32F, DIO24, or DIO-96 board, but I do not want to use handshaking. I just want to read one group of ports just once. How can I set it up in software?

Use Easy I/O VIs (Write to Digital Port or Read from Digital Port) or Advanced Digital VIs (DIO Port Config, DIO Port Write or DIO Port Read), and set multiple ports in the port list. For Easy I/O VIs, you can specify up to four ports in the port list. Whatever data you try to output to each port of your group will correspond to each element of the data array. This also applies for input.

I want to use the OUT1, OUT 2, OUT3 and IN1, IN2, IN3 pins on my DIO-32F board. How do I address those pins using the Easy I/O Digital VIs in LabVIEW?

These output and inputs pins are addressed together as port 4. OUT1 and IN1 are referred to as bit 0, OUT2 and IN2 are referred to as bit 1, and OUT3 and IN3 are referred to as bit 2. Only the NB-DIO-32F has 3 pins for each direction. If you use the Write To Digital Port VI, you will output on the OUT pins, and if you use the Read From Digital Port VI, you will input from the IN pins.

I want to use a TTL digital trigger pulse to start data acquisition on my DAQ device. I noticed there are two types of triggers: Digital Trigger

A, and Digital Trigger A&B. Which digital trigger setting should I use and where should I connect the signal?

You should use Digital Trigger A, which stands for first trigger," to start a data acquisition. Digital Trigger B, which stands for second trigger," should only be used if you are doing both a start AND a stop trigger for your data acquisition. Connect your trigger signal to either STARTTRIG* (pin 38) if you are using an AT-MIO-16, AT-MIO-16D, NB-MIO-16X, or EXTTRIG* or DTRIG for any other board that has that pin. The only analog input boards on which you cannot do a digital trigger are the PC-LPM-16, DAQCard-700, and the DAQCard-500. Refer to the AI Trigger Config description for more information on the use of digital triggers on your DAQ device.

Note: The NB-MIO-16 has an EXTTRIG* pin, but cannot support start and stop triggering.

When are the data acquisition boards initialized?

All data acquisition boards are initialized automatically when the first DAQ VI is loaded in on a diagram when you start LabVIEW. You can also initialize a particular board by calling the Device Reset VI.

(Windows) I open a VI that calls a DAQ VI, or drop a DAQ subVI on a block diagram, and crash.

The first time a DAQ VI is loaded into memory in LabVIEW, LabVIEW opens the library (dll) that controls data acquisition. A crash at this time indicates a problem in communicating with the driver. This may indicate that there is a conflict with another device in the machine.

To determine the source of the problem, quit LabVIEW and Windows, re-launch Windows, and run wdaqconf.exe. Run a simple configuration test with the NI boards in the machine. If this results in a crash, there is probably a conflict with another device in the machine or the drivers file versions do not correspond for some reason. If not, you need to obtain the latest version of the DAQ driver from NI BBS, World Wide Web, or FTP site.

We have also seen cases where the video driver conflicts with both WDAQCONF and LabVIEW. You can obtain the *Error Messages and Crashes Common Questions* document from the NI FaxBack system.

(Windows) I am having problems accessing ports 2 or higher on the AT-MIO-16D or PC-DIO-96.

A problem was found in version 3.0 with addressing the higher-numbered ports on these boards. To fix the problem, get the updated version of atwdaq.dll, and use the updated DIO Port Read.vi. These updated files are included with LabVIEW for Windows version 3.0.1. LabVIEW for Windows version 3.0 users can obtain these files by downloading the file win30up2.zip from the NI BBS or FTP sites. The file is in the directory support/labview/windows/LVWin3.0/updates.

(Windows)While performing analog input, I get memory allocation errors (-10444) even though I have a large amount of memory on my machine.

Buffers for data acquisition arrays, unlike arrays for LabVIEW buffers, must be available in physical RAM, not in virtual memory. For example, assume the machine has 32 MB of RAM, and LabVIEW is allocated 8 MB of RAM. The memory allocated to LabVIEW wires will come out of the 8 MB; however, data acquisition buffers will be allocated out of the remaining 24 MB of RAM.

On an AT style machine (ISA) using DMA, the DMA controller can only address the first 16 MB of RAM. If you get "out of memory" errors on your machine, try setting the board to use programmed I/O (interrupts

only) with the Set Device Information.vi (in DAQ->Calibration and Config). An alternative is to switch to an EISA bus machine. The DMA controller can address up to 4 GB of RAM on EISA machines.

(Windows 3.1)I am having problems running Windows for Workgroups with my data acquisition program.

Remark out nivisrd.386 in the [386 Enhanced] section of your system.ini file. To mark out the line, place a number sign (#) at the beginning of the line which reads:

device = c:\windows\system\nivisrd.386

nivisrd.386 normally improves performance by reducing interrupt latencies in Windows enhanced mode.

(Windows 3.1)I bought LabVIEW for Windows and also have a slightly older DAQ device from National Instruments. I installed the entire LabVIEW package, but should I go ahead and install my NI-DAQ for Windows drivers that I originally got with the board?

In most cases, the answer is no. The LabVIEW installer installs a set of DAQ driver files that are guaranteed to work with LabVIEW, whereas if you happen to install an older version of the drivers, you may run into many problems. You may even end up crashing your computer every time you do any data acquisition. If you buy a new DAQ device and if you already have LabVIEW installed, it is safe to install the NI-DAQ for Windows drivers from those disks. In any case, make sure you install and use the latest version of the NI-DAQ drivers.

(Macintosh) My analog input VIs returns error -10845 (buffer overflow). What is the problem?

If you do not have an NB-DMA-2800 or NB-DMA8G board in your Macintosh and are trying to acquire at sampling rates (not scan rates) greater than 8 kHz, you may get this error. Even at sampling rates under 8 kHz, depending on the type of machine, you may run into overflow error problems if there is a lot of other interrupts that need to be serviced. This is all due to the long interrupt latencies. If you do have either DMA board in your Macintosh, make sure that you have a RTSI cable connecting your DAQ device and the DMA board. Even after you connect a RTSI cable, restarting LabVIEW may help.

Also, if you have a Quadra, your errors may be caused by prolonged network interrupt latencies, which prevents the NI-DAQ driver from copying the data in the DAQ device resident memory to the memory on your computer. In this case, you can either disable AppleTalk in the Chooser and disconnect your AppleTalk cable or contact National Instruments and ask for the newest revision of the NB-MIO-16X (which has a larger device resident memory) if you are using either of those DAQ devices.

Common Data Acquisition Parameter Definitions

This topic explains some of the common parameters used in most LabVIEW data acquisition VIs. These parameters are not explained within each VI they appear in, so you should review them before wiring any VIs in LabVIEW.

buffer size is the number of scans (with input operations) or updates (with output operations) you want the circular buffer to hold.

channel, channels, channel list is a list of the analog channels you want to use. If `x`, `y`, and `z` refer to channels, you can specify a list of channels by separating the individual channels with commasfor example, `x,y,z`. If `x` refers to the first channel in a consecutive channel range and `y` refers to the last channel, you can specify the range by separating the first and last channels by a colonfor example, `x:y`. Refer to the Channel Addressing with the AMUX-64T section of Chapter 5, *Things You Should Know about Analog Input*, in the *LabVIEW Data Acquisition Basics Manual*, for the syntax you use for AMUX-64T channels or SCXI.

(***Macintosh and Windows***) The AT-AO-6/10, NB-MIO-16, and NB-AO-6 devices impose certain restrictions on channel group assignments. Refer the hardware user manual for your device for more information on channel group assignments.

clear acquisition determines whether the VI clears the task after reading the specified number of scans. You should pass a value of `TRUE` for this parameter when reading the last set of scans for a given acquisition. The default is `TRUE`, which means that if you leave thisinput unwired, the VI reads data only once. You normally wire this input to the terminating condition of a loop, so that when the loop finishes, the VI clears the acquisition.

counter, counter list identifies the counter(s) the VI configures. The counter(s) specified by counter list must belong to the group identified by **taskID**. If counter list is empty, the VI programs all counters in the group. See the CTR Group Config VI description in this chapter for the counter list element syntax and valid counter numbers.

counter size is set to `0` (default) to use a single 16-bit Am9513 counter or 24-bit DAQ-STC counter, or is set to 1 to use two Am9513 counters as a 32-bit counter. Leave this input set to 0 for a DAQ-STC counter.

device is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. The **error in** cluster contains the following parameters.

status is `TRUE` if an error occurred. If **status** is `TRUE`, the VI does not perform any operations.

code is the error code number identifying an error. A value of 0 means no error, a negative value means a fatal error, and a positive value is a warning. Refer to the Data Acquistion VI Error Codes for a code description.

source identifies where an error occurred. The **source** string is usually the name of the VI that produced the error.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

input limits is an array of clusters. Each array element contains the expected signal limits for the channels specified by the corresponding element of **channels**. If there are fewer elements in this array than in **channels**, the VI uses the last array element for the rest of the channels. The **input limits** array defaults to an empty array, which means the **input limits** keep their default settings. Each cluster contains the following parameters

high limit is the highest expected voltage level of the signals you want to measure. The default input is `10` V (`5` V for Lab and 1200 Series devices). LabVIEW uses this value to compute the gain.

low limit is the lowest expected voltage level of the signals you want to measure. The default input is `-10` V (`-5` V for Lab and 1200 Series devices). LabVIEW uses this value to compute the gain.

number of scans to read is the number of scans the VI retrieves from the ongoing acquisitions buffer each time. This is equivalent to the number of points per channel. If the VI times out, it returns the number of scans available.

port list is a list of digital ports, each of which is 8 lines wide. If element *n* of **port list** contains the value 1, port 1 is a member of the group. The order of **port list** determines how data that is read from or written to the group maps to the ports.

(***Macintosh and Windows***) The last port in the list controls the handshaking lines if the device is a Lab and 1200 Series, DIO-24, or DIO-96 device.

If the device is a DIO-32F, the following rules apply. If you combine ports 0 and 1 or use them separately, the REQ1 and ACK1 signals control handshaking. If you combine ports 2 and 3 or use them separately, the REQ2 and ACK2 signals control handshaking. If you combine ports 0, 1, 2, and 3, the REQ1 and ACK1 signals control handshaking. Because the two sets of handshaking signals (REQ1/ACK1 and REQ2/ACK2) are independent, you can run two groups simultaneously on a DIO-32F device.

The syntax for **port list** is `port[:port|,port]`. Items in brackets are optional. A vertical bar character (`|`) separating items means you must choose one option or the other, but not both.

The following table gives examples of the meanings of **port list** elements.

Value of port list	Meaning
port list[0] = `0`	Port 0
port list[0] = `0` **port list**[1] = 1	Ports 0 and 1
port list[0] = `0:3`	Ports 0, 1, 2, and 3
port list[0] = `0,1,3,4`	Ports 0, 1, 3, and 4

port number is the port this VI configures. If you use a digital SCXI module, use the `SCx!MDy!0` syntax, where `x` is the chassis ID and `y` is the module number, to specify the port on a module.

port width is the total width or the number of lines of the port in bits. For example, you can combine two 4-bit ports into an 8-bit port on an MIO device by setting **port width** to `8`.

When **port width** is greater than the physical width of a digital port, the following restrictions apply. The **port width** must be an integral multiple of the physical port width, and the port numbers in the combined port must begin with the port named by **port number** and must increase consecutively. For example, if **port number** is 3 and **port width** is 24 (bits), the VI uses ports 3, 4, and 5.

sample contains the scaled analog input data for the specified channel.

status returns any error or warning condition from the VI. Refer the Data Acquistion VI Error Codes for a code description.

taskID in identifies the group and the I/O operation.

taskID out has the same value as **taskID in**.

timebase is the internal clock signal to use (default 1 MHz). If the counter overflows because **timebase** is too high, lower it until a valid reading occurs or until the lowest timebase is used and a timeout occurs.

update rate is the number of voltage updates to generate per second. The default rate is `1000` updates/s.

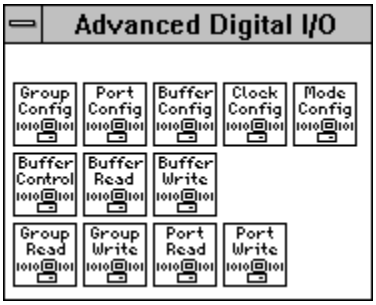
voltage data is a 2D array that contains analog data. (The AI Read One Scan, AI Single Scan, AO Write One Update, and AO Single Update VIs return only a 1D array in the **voltage data** parameter.) This array is scan-ordered which means that each row contains the data of a single scan or value. If you want to index the array in a loop, you will get a scan on each iteration. Each column holds the data of a single channel. The upper array index selects the scan and the lower index selects the channel. Refer to the Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, in the *LabVIEW Data Acquisition Basics Manual* for more information about the array format.

Advanced Digital I/O VIs

The Advanced Digital I/O VIs include the digital port and digital group VIs. You use the digital port VIs for immediate reads and writes to digital lines and ports. Use the digital group VIs for immediate, handshaked, or clocked I/O for multiple ports. These VIs are the interface to the NI-DAQ software and the foundation of the Easy and Intermediate Digital I/O VIs. For general information about these VIs, see the Advanced Digital I/O VIs Overview.

Access the Advanced Digital I/O palette by choosing **Functions**»**Data Acquisition**»**Digital I/O**»**Advanced Digital I/O**. The icon that you must select to access the Advanced Digital I/O VIs is on the bottom row of the **Digital I/O** palette.

Click on an icon in the following illustration to go to that particular Advanced Digital I/O VI.



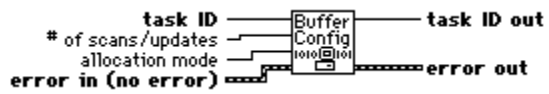
For examples of how to use the Advanced Digital I/O VIs, open the example library by opening `examples\daq\digital\digio.llb`.

Advanced VIs

Digital Buffer Config
Digital Buffer Control
Digital Buffer Read
Digital Buffer Write
Digital Clock Config
Digital Group Config
Digital Mode Config
Digital Single Read
Digital Single Write
DIO Port Config
DIO Port Read
DIO Port Write

Digital Buffer Config (Macintosh and Windows)

Allocates memory for a digital input or output buffer.



I32 **taskID in** identifies the group and the I/O operation.

I32 **# of scans/updates** tells LabVIEW how much memory to allocate for the buffer. For example, if you want to acquire 1,000 scans of input data, set this control to 1000.

(Windows) If you are acquiring data indefinitely, **# of scans/updates** controls how many scans LabVIEW can hold in memory while waiting for you to process them.

(Macintosh) If you are using DMA, the buffer size cannot be greater than 224- 1, which is 16,777,215 bytes.

The **# of scans/updates** parameter defaults to -1, which means LabVIEW leaves the current **# of scans/updates** setting unchanged. The default setting for **# of scans/updates** is 1000.

I32 **allocation mode** can have the following values.

0: Do not change the **allocation mode** setting (default input).
1: Reserved.
2: Deallocate the buffer assigned to the group.
3: Allocate PC memory (default setting).

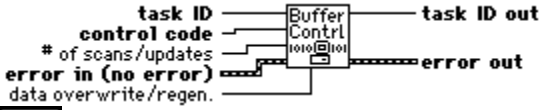
I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Digital Buffer Control

Starts an input or output operation.



I32 **taskID in** identifies the group and the I/O operation.

I32 **control code** can have the following values.

0: Start (default input).
1: Pause. *LabVIEW does not currently support this option.*
2: Reserved.
3: Resume. *LabVIEW does not currently support this option.*
4: Clear.

The start operation, **control code** 0, starts the input or output operation.

For input or output operations, **control code** 1 and 3, either pause or resume the operation.

The clear operation, **control code** 4, stops the digital I/O operation and frees any internal buffers allocated by LabVIEW.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **# of scans/updates** is the amount of data that you want the VI to acquire or generate. In Windows, a value of 0 tells the VI to acquire or generate data indefinitely. A value of -1 tells the VI to leave the current setting unchanged. The default setting equals the number of scans or updates in the buffer, which you established when you called Digital Buffer Config VI. The **# of scans/updates** parameter cannot exceed the buffer size.

I32 **data overwrite/regeneration** has the following values. *LabVIEW does not currently support this parameter.*

0: Do not change the current setting (default input).
1: Do not overwrite input data or regenerate output data (default setting).
2: Allow input overwrites or output regeneration.

When you set **data overwrite/regeneration** to 1 for a handshaked operation, the operation pauses until the Digital Buffer Read VI reads the data that could be overwritten or until the Digital Buffer Write VI writes new data into the output buffer.

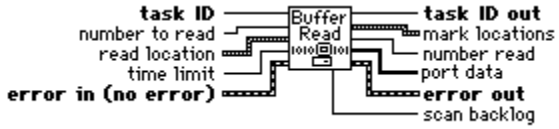
When you set **data overwrite/regeneration** to 1 for a clocked operation, the operation stops when data is about to be overwritten or regenerated.

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Digital Buffer Read (Macintosh and Windows)

Returns digital input data from the internal data buffer.



I32 **taskID in** identifies the group and the I/O operation.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **number to read** specifies the number of scans that you want the VI to read. The **number to read** parameter defaults to -1, which means LabVIEW leaves the **number to read** setting unchanged. The default setting is the number of scans to acquire, which you specify by calling the Digital Buffer Control VI. If you set **number to read** to 0, you can check the scan backlog to see how many scans have accumulated. If the number of scans to acquire was 0, starting a continuous operation, **number to read** must be a multiple of one half of the buffer size specified in Digital Buffer Config.

I32 **read location** contains the following parameters.

I32 **read offset**. The VI adds **read offset** to the mark specified by **read mode** to determine the starting point for the read. You express **read offset** in scans. The default setting and input is 0.

I32 **read mode** indicates which reference mark within an input buffer provides the reference point for the read. This reference can be the read mark, the beginning of the buffer, or the most recently acquired data (end of data).

0: Do not change the **read mode** setting (default input).
1: Relative to the read mark (default setting).
2: Relative to the start of the buffer.
3: Relative to the end of the data.

The read mark is similar to a file I/O pointer, in that it moves every time you make a read call to get input buffer data. Reading always begins at the scan to which the read mark points. After the read, the read mark points to the next unread scan.

The end of data mark points to the most recently acquired scan. The start of buffer points to scan number 1 of the buffer.

I32 **time limit**. You express **time limit** in seconds. This parameter defaults to -1.0, which means LabVIEW leaves the **time limit** setting unchanged if the transfer is handshaked or calculates a time limit if the transfer is clocked and the clock rate is known. The default setting is 1 s for handshaked transfers. The resolution of the timeout clock is about 17 ms ***(Macintosh)*** or 55 ms ***(Windows)***.

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

I32 **mark locations** contains the following parameters.

I32 **read mark scan** is the number of the scan that the VI will read the next time you call this VI.

I32 **end of data scan** is the number of the most recently acquired scan.

acquisition state has the following values.

0: Running.
1: Finished with backlog.
2: Finished with no backlog.
3: Paused.
4: No acquisition.

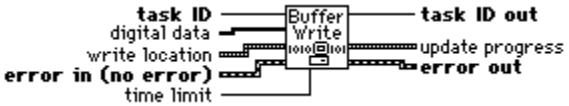
number read is the number of scans returned.

port data is a 1D array that contains the digital data that the VI obtained from the internal buffer.

scan backlog is the amount of buffer data that calls to this VI have not returned.

Digital Buffer Write (Macintosh and Windows)

Writes digital output data to the buffer created by the Digital Buffer Config VI. The write always begins at the write mark. After a write, the write mark points to the update following the last update written.



(Macintosh) Fill the buffer with data before you use the Digital Buffer Control VI to begin the digital output operation. You can call the Digital Buffer Write VI after the transfer begins to retrieve status information.

taskID in identifies the group and the I/O operation.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

digital data is a 1D array that contains digital data for output. If the output operation is continuous, the length of the array must be a multiple of one half of the buffer size specified in the Digital Buffer Config VI. If you call this VI with an empty array, you can examine the contents of the update progress cluster and determine how the updates are proceeding.

write location contains the following parameters.

write offset. The VI adds the value of **write offset** to the write mark to determine the position at which the write begins. This parameter defaults to -1, which means LabVIEW leaves the **write offset** setting unchanged. The default setting is 0.

write mode has the following values.

0: Do not change the **write mode** setting (default input).
1: Relative to the write mark (default setting).
2: Relative to the beginning of the buffer.

Setting **write mode** to 2 moves the write mark to the beginning of the buffer before the VI adds **write offset** to the write mark.

time limit is expressed in seconds. The VI cannot write new data to a region of the output buffer if LabVIEW is already reading data from the same region. Normally, the VI waits until LabVIEW finishes writing to this region. You can use **time limit** to limit this waiting period. This parameter defaults to -1, which means LabVIEW leaves the **time limit** setting unchanged if the output is handshaked or calculates a **time limit** if the output is clocked and the clock rate is known. The default setting is 1 s for handshaked output. The resolution of the timeout clock is about 17 ms ***(Macintosh)*** 55 ms ***(Windows)***.

taskID out has the same value as **taskID in**.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

update progress contains the following parameters.

write mark points to the next update to which the VI will write.

I32 **output mark** points to the next update the VI will generate.

I32 **buffer iterations** indicates the number of complete buffer iterations that the VI has produced so far.

I32 **buffer state** has the following values.

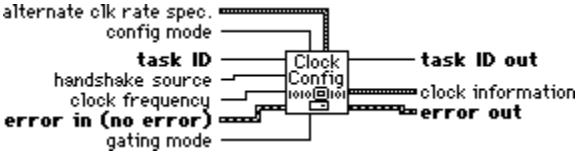
0: No such buffer (default setting).
1: Waiting for data.
2: Ready.
3: Active.
4: Finished.

The **buffer state** parameter is 1 when LabVIEW has allocated a buffer but has not yet written to it. The **buffer state** parameter is 2 when LabVIEW has written to a buffer but has not yet started it.

The total number of updates written to a buffer before you start it can be less than the number of updates you allocated the buffer to hold when you called the Digital Buffer Config VI. The VI generates only the updates written to the buffer.

Digital Clock Config

Configures a DIO-32F device to produce handshake signals based on the output of a clock for timed digital I/O.



Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual,* for the clocks available with your DAQ device.

I32 **taskID in** identifies the group and the I/O operation.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **alternate clock rate specification** contains the following parameters.

I32 **clock period** is expressed in s/scan or s/update. A **clock period** value of -1.0 tells the VI not to use this input and to check the three timebase parameters for rate information. The default input for **clock period** is
-1.0. The default setting is undefined.

I32 **timebase source** can have the following values.

0: Do not change the timebase source (default input).
1: Internal frequency in hertz (default setting).

I32 **timebase signal**, in conjunction with the **timebase source**, fully specifies the timebase. If **timebase source** is 1 (internal frequency in hertz), **timebase signal** is that frequency in Hz. The default input for **timebase signal** is -1.0, which tells the VI not to change the **timebase signal** setting. The default setting for **timebase signal** is undefined. The list of valid timebase signals is:

10,000,000
1,000,000
100,000
10,000
1,000
100

I32 **timebase divisor** is the divide-down value the VI uses to create the clock rate. The default input for **timebase divisor** is -1, which tells the VI not to change the **timebase divisor** setting. The default setting is undefined.
The following example illustrates how to use the three timebase parameters to specify a clock rate. Assume these parameters have the following settings.

timebase source:	1
timebase signal:	1,000,000.0 Hz
timebase divisor:	25

In this case, the ticks per second rate is 1,000,000.0 divided by 25, so LabVIEW updates the digital group 40,000 times per second.

I32 **config mode** can have the following values.

0: Do not change the **config mode** setting (default input).
1: Change rate immediately (default setting).
2: Change rate when current buffer is finished.
3: Do not change rate. This option is useful for rate translation only.

I32 **handshake source** can have the following values.

0: Do not change the **handshake source** setting (default input).
1: Internal clock. This mode is not valid for the NB-DIO-32F device, unless a National Instruments DMA device is associated with the DIO device, and the two devices share a common RTSI bus.
2: I/O connector (default setting).
3: RTSI connection.

When **handshake source** is an I/O connector or RTSI connection, you must connect a handshake signal to the REQ1 or REQ2 input through either the I/O connector or the RTSI bus. Refer to the Digital Group Config VI description in this chapter for more information on the REQ1 and REQ2 inputs. When **handshake source** is an internal clock, do not connect the REQ inputs to anything.

You can express clock rates three wayswith **clock frequency**, with **clock period**, or with **timebase source**, **timebase signal**, and **timebase divisor**. The VI determines which method to use by searching for the first input that is not equal to -1.0 in the order of **clock frequency**, **clock period**, and then the three timebase parameters.

I32 **clock frequency** is expressed in scans/s or updates/s, depending on the input or output direction of the group. A **clock frequency** value of -1.0 tells the VI not to use this input and to check **clock period** for rate information. This parameter defaults to -1.0 and the default setting is undefined.

I32 **gating mode** specifies whether the VI externally gates an internal counter.
(Windows) If you enable this gating mode, the signal connected to the IN1 or IN2 lines gates the internal counter. If REQ1 is producing the handshake signals, use IN1. If REQ2 is producing the handshake signals, use IN2. For the AT-DIO-32F, the signal must be high to enable the counter.

(Macintosh) If you enable this gating mode, the NB-DIO-32F does not produce request signals until the signal connected to the IN1 input on the I/O connector is at a logical low state.

0: Do not change the **gating mode** setting (default input).
1: Disable external gating (default setting).
2: Enable external gating.

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster

contains the same information. Otherwise, **error out** describes the error status of this VI.

I32 **clock information** contains the following parameters.

I32 **actual clock frequency** is the actual clock frequency the device was able to provide.

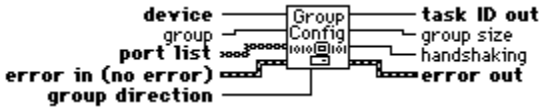
I32 **actual clock period** is the actual clock period the device was able to provide.

I32 **actual timebase signal** is the value the VI calculated from the frequency or period input and actually used to program the clock.

I32 **actual timebase divisor** is the value the VI calculated from the frequency or period input and actually used to program the clock.

Digital Group Config

Defines a digital input or output group. You can use the **taskID** this VI returns only in the digital group VIs.



Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual,* for the ports and directions available with your DAQ device.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **port list** is a list of digital ports, each of which is 8 lines wide. If element *n* of **port list** contains the value 1, port 1 is a member of the group. The order of **port list** determines how data that is read from or written to the group maps to the ports.

(***Macintosh and Windows***) The last port in the list controls the handshaking lines if the device is a Lab and 1200 Series, DIO-24, or DIO-96 device.

If the device is a DIO-32F, the following rules apply. If you combine ports 0 and 1 or use them separately, the REQ1 and ACK1 signals control handshaking. If you combine ports 2 and 3 or use them separately, the REQ2 and ACK2 signals control handshaking. If you combine ports 0, 1, 2, and 3, the REQ1 and ACK1 signals control handshaking. Because the two sets of handshaking signals (REQ1/ACK1 and REQ2/ACK2) are independent, you can run two groups simultaneously on a DIO-32F device.

The syntax for **port list** is `port[:port|,port]`. Items in brackets are optional. A vertical bar character (`|`) separating items means you must choose one option or the other, but not both.

The following table gives examples of the meanings of **port list** elements.

Value of port list	Meaning
port list[0] = `0`	Port 0
port list[0] = `0`	Ports 0 and 1
port list[1] = 1	
port list[0] = `0:3`	Ports 0, 1, 2, and 3
port list[0] = `0,1,3,4`	Ports 0, 1, 3, and 4

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **group direction** sets the direction for the group.

0: Do not change the **group direction** setting (default input).
1: Input (default setting).
2: Output.
3: Bidirectional (port 0 only on DIO-24, Lab and 1200 Series devices; port 2 only on AT-MIO-16D and AT-MIO-16DE-10; ports 0, 3, 6, and 9 only on DIO-96). *LabVIEW for Sun does not currently support this option.*

I32 **group** is the number the VI assigns to the set of ports, ranging from 0 to 15. The default input and setting for **group** is 0. The default **group** 0 contains ports 0 through *n*, where *n* is the maximum number of ports minus 1. The maximum number of ports varies with device type.

(Macintosh and Windows) If the device supports handshaking, the default group is the largest group that allows handshaking.

Note: The same port cannot belong to two different groups. If you configure a group to use a specified port, that port must be one that is not already defined in another group or you will get an error.

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

I32 **group size** is the number of 8-bit ports assigned to the group.

I32 **handshaking** indicates whether this group can use handshaking.

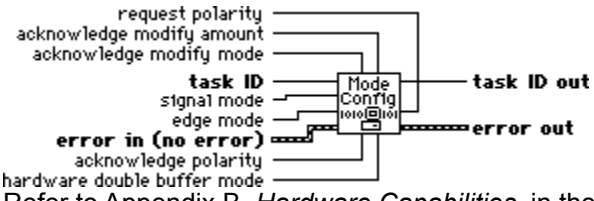
0: This group does not allow handshaking.
1: This group allows handshaking.

MIO devices (except for the AT-MIO-16D and the AT-MIO-16DE-10), as well as the NB-TIO-10, PC-LPM-16, DAQCard-500, DAQCard-700, PC-TIO-10, AO-2DC devices, and AT-AO-6/10, do not allow handshaking. The digital port VIs are more appropriate for these devices. The AT-MIO-16D and AT-MIO-16DE-10 do not allow handshaking if **port list** includes ports 0, 1, and/or 4. The DIO-96 devices do not allow handshaking if **port list** includes ports 2, 5, 8, and/or 11. The DIO-24 and Lab and 1200 Series devices do not allow handshaking if **port list** includes port 2. The DIO-32F allows handshaking for the following configurations only.

- A group containing any one port.
- A group containing ports 0 and 1, or ports 2 and 3, in that order.
- A group containing ports 0, 1, 2, and 3, in that order.

Digital Mode Config (Macintosh and Windows)

Configures the handshaking characteristics for DIO-32F devices.



Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual,* for the handshake modes available with your DAQ device.

I32 **taskID** identifies the group and the I/O operation.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **request polarity** specifies active high or active low handshaking request signals.

0: Do not change the **request polarity** setting (default input).
1: Active low requests (default setting).
2: Active high requests.

I32 **acknowledge modify amount** specifies the length of delay or the pulse width, depending on the setting of **acknowledge modify mode**. Only the following values are valid for **acknowledge modify amount**.

0 ns
100 ns (default input and setting)
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns

I32 **acknowledge modify mode** specifies whether LabVIEW delays issuing the acknowledge pulse or changes the duration of the pulse. You can change the duration of the pulse only when **signal mode** specifies edge handshaking signals.

0: Do not change the **acknowledge modify mode** setting (default input).
1: No modification (default setting).
2: Delay the acknowledge pulse.
3: Change the duration of the acknowledge pulse.

I32 **signal mode** specifies level or edge handshaking signals.

0: Do not change the **signal mode** setting (default input).
1: Level (default setting).
2: Edge.

I32 **edge mode** is valid only if **signal mode** specifies edge signals. The **edge mode** parameter specifies whether LabVIEW uses the leading or trailing edges of the handshaking signal.

0: Do not change the **edge mode** setting (default input).
1: Leading edges (default setting).
2: Trailing edges.

I32 **acknowledge polarity** specifies active high or active low handshaking acknowledge signals.

0: Do not change the **acknowledge polarity** setting (default input).
1: Active low acknowledges (default setting).
2: Active high acknowledges.

I32 **hardware double buffer mode** turns hardware double buffering on and off. When the direction is input and hardware double buffering is on, LabVIEW latches data into the internal buffer when the handshake signal becomes active. When the direction is output and hardware double buffering is on, you do not see data written to a group at the output port until the handshake signal becomes active. You typically use output hardware double buffering when a clock produces the handshake signals (that is, in a pattern generation operation.)

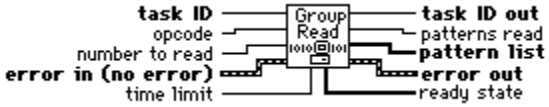
0: Do not change the **hardware double buffer mode** setting (default input).
1: Off (default setting for output groups).
2: On (default setting for input groups).

taskID out has the same value as **taskID**.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Digital Single Read

Reads the ports that belong to the group identified by **taskID** and returns the patterns read.



taskID identifies the group and the I/O operation.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.error in Cluster Parameters

opcode can have the following values.

0: Do not change the **opcode** setting (default input).
1: Read when data is ready for transfers (default setting).
2: Check **ready state** only. LabVIEW for Sun does not currently support this option.
3: Read data immediately, regardless of **ready state**.

If the group identified by **taskID** does not allow handshaking, the group is always ready for reading. When **opcode** is 1 or 3, the VI reads data into **pattern list** and returns immediately, and the value of **ready state** is always 1.

(Macintosh and Windows) If the group identified by **taskID** allows handshaking, the state of the handshaking lines determines whether the data is ready to be read. When **opcode** is 1, the VI waits until the group is ready or a timeout occurs. If the group is ready before the timeout occurs, the VI reads the data into **pattern list**. When **opcode** is 3, the VI reads the data into **pattern list** immediately, without waiting for the group to be ready. When handshaking is allowed, **ready state** reflects the handshaking states of each port and can have the values 0 or 1.

number to read is the number of patterns you want the VI to read before returning. The number of bytes in each pattern equals the number of ports in the group. The VI always reads complete patterns. This parameter defaults to -1, which means LabVIEW leaves the **number to read** setting unchanged. The default setting is 1, which tells the VI to read a single pattern.

time limit is expressed in seconds and defaults to 1 s. You can use **time limit** to limit waiting periods when the group is handshaking (when **opcode** is 1).

taskID out has the same value as **taskID in**.

pattern list contains the data the VI read from the group ports. The port order of the **pattern list** data is the order of ports in the group.

(Macintosh and Windows) For example, if ports 3 and 5 are in a group, **pattern list** 0 is data from port 3, and **pattern list** 1 is data from port 5.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

patterns read is the number of complete patterns the VI read. The value of this parameter ranges from zero up to the value of **number to read**.

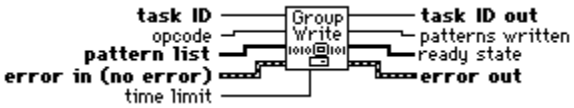
ready state. Each element in the **ready state** array corresponds to a port in the group. If **ready state** [0] is 0, the first port in the group is not ready to be read. If **ready state** [1] is 1, the second port is ready to be read. You may want to make use of the **And Array Elements** function in the **Array & Cluster** palette to and together all elements in the **ready state** array.

(Macintosh and Windows) On the DIO-32F, all ports in a group are ready at the same time, so

you only need to check **ready state** [0]. If the group does not allow handshaking, **ready state** is always 1.

Digital Single Write

Writes the data in **pattern array** to the ports that belong to the group identified by **taskID**.



taskID identifies the group and the I/O operation.

pattern list contains the data the VI writes to the group ports. The port order of the **pattern list** data is the same as the order of the ports in the group.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

opcode can have the following values.

0: Do not change the **opcode** setting (default input).
1: Write data when group is ready for transfers (default setting).
2: Check **ready state** only. *This option is currently not supported.*
3: Write data immediately, regardless of **ready state**.

If the group identified by **taskID** does not allow handshaking, the group is always ready for writing. When **opcode** is 1 or 3, the VI writes data into **pattern list** and returns immediately.

If the group identified by **taskID** allows handshaking, the state of the handshaking lines determines whether the data is ready. When **opcode** is 1, the VI waits until the group is ready or a timeout occurs. If the group is ready before a timeout occurs, the VI writes the data to the output ports. When **opcode** is 3, the VI writes the data to the output ports immediately, without waiting for the group to be ready. When handshaking is allowed, **ready state** reflects the handshaking states of each port and can have the values 0 or 1.

time limit is expressed in seconds and defaults to 1 s. You can use **time limit** to limit waiting periods when the group is handshaking (when **opcode** is 1).

taskID out has the same value as **taskID in**.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

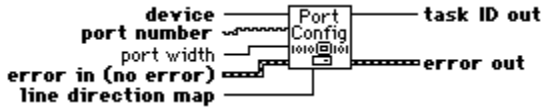
patterns written returns the number of patterns the VI was able to write. The VI writes only complete patterns. For example, if the group contains two ports, but **pattern list** contains five bytes (2.5 patterns), the VI does not write the last byte to the ports.

ready state. Each element in the **ready state** array corresponds to a port in the group. If **ready state** [0] is 0, the first port in the group is not ready for the VI to write to it. If **ready state** [1] is 1, the second port is ready for the VI to write to it. You may want to make use of the **And Array Elements** function in the **Array & Cluster** palette to and together all elements in the **ready state** array.

(Macintosh and Windows) On the DIO-32F, all ports in a group are ready at the same time, so you only need to check **ready state** [0]. If you disable handshaking, **ready state** is invalid.

DIO Port Config

Establishes a port configuration. You can use the **taskID** that this VI returns only in digital port VIs.



Note: When you call this VI on a digital I/O port that is part of an 8255 PPI when your iteration terminal is left at 0, the 8255 PPI goes through a configuration phase, where all the ports within the same PPI chip get reset to logic low, regardless of the data direction. The data direction on other ports, however, is maintained. To avoid this effect, connect a value other than 0 to the iteration terminal once you have configured the desired ports.

Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the ports and directions available with your DAQ device.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **port number** is the port this VI configures. If you use a digital SCXI module, use the `SCx!MDy!0` syntax, where `x` is the chassis ID and `y` is the module number, to specify the port on a module.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **line direction map** specifies the direction of each line in the port. If a bit is 0 in the line map, the line is an input line. If a bit is 1, the line is an output line. Set **line direction map** to -1 to make all the lines in a port output lines. Set **line direction map** to 0 to make all the lines in a port input lines. The NB-TIO-10, PC-TIO-10, AO-2DC devices, and the E Series devices are the only devices on which you can configure lines within a port for different directions. The **line direction map** parameter defaults to 0.

I32 **port width** is the total width or the number of lines of the port in bits. For example, you can combine two 4-bit ports into an 8-bit port on an MIO device by setting **port width** to `8`.

When **port width** is greater than the physical width of a digital port, the following restrictions apply. The **port width** must be an integral multiple of the physical port width, and the port numbers in the combined port must begin with the port named by **port number** and must increase consecutively. For example, if **port number** is 3 and **port width** is 24 (bits), the VI uses ports 3, 4, and 5..

(Macintosh and Windows) For example, you can combine two 4-bit ports into an 8-bit port on an MIO device by setting **port width** to 8. When **port width** is greater than the physical port width of a digital port, the following restrictions apply. The **port width** must be an integral multiple of the physical port width. The port numbers in the combined port must begin with the port named by **port number** and must increase consecutively. For example, if **port number** is 3 and **port width** is 24 (bits), the VI uses ports 3, 4, and 5.

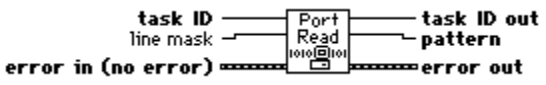
Click here to see the physical port widths for most DAQ devices.

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

DIO Port Read

Reads the port identified by **taskID** and returns the pattern read in **pattern**.



I32 **taskID** identifies the group and the I/O operation.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

line mask determines which lines this VI reads. If you set **line mask** to -1, the VI reads the entire port. If you set only bit *n* in **line mask**, the VI reads only port line *n*. If the state of line *n* is high, the VI sets bit *n* of **pattern**. The VI sets the bits of **pattern** that correspond to lines the VI has not read to 0. The default setting for **line mask** is -1.

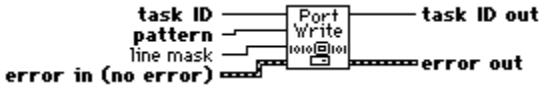
taskID out has the same value as **taskID in**.

pattern specifies the new state of the lines in the port.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

DIO Port Write

Writes the value in **pattern** to the port identified by **taskID**.



taskID identifies the group and the I/O operation.

pattern specifies the new state of the lines in the port.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

line mask determines which lines in the port the write affects. If you set **line mask** to -1, the VI writes to the entire port and sets all lines in the port to the states specified by the bits in **pattern**. If you set only bit *n* in **line mask**, the VI changes only port line *n* to the state that bit *n* of **pattern** specifies. The default setting for **line mask** is -1.

taskID out has the same value as **taskID in**.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Digital Group Config VI

Digital Group Config

DIO Port Config VI

DIO Port Config

Digital Buffer Config VI (Macintosh and Windows)

Digital Buffer Config

Digital Clock Config VI

Digital Clock Config

Digital Mode Config VI (Macintosh and Windows)

Digital Mode

Digital Buffer Control VI (Macintosh and Windows)

Digital Buffer Control

Digital Buffer Read VI (Macintosh and Windows)

Digital Buffer Read

Digital Buffer Write VI (Macintosh and Windows)

Digital Buffer Write

Digital Single Read VI

Digital Single Read

Digital Single Write VI

Digital Single Write

DIO Port Read VI

DIO Port Read

DIO Port Write VI

DIO Port Write

DAQ Device Physical Port Widths

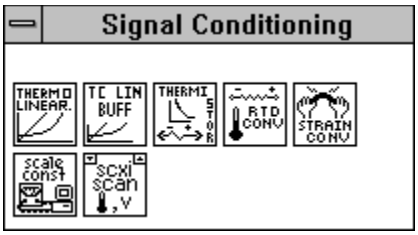
Device	Ports	Physical Port Width
MIO-16L/H	0, 1	4 bits
AT-MIO-16D	0, 1 2, 3, 4	4 bits 8 bits
Most E Series Devices SB-MIO-16E-4	0	8 bits
AT-MIO-10DE-10	0, 2, 3, 4	8 bits
AT-AO-6/10	0, 1	4 bits
PC-TIO-10 NB-TIO-10 A0-2DC Devices	0, 1	8 bits
PC-LPM-16 DAQCard-700	0, 1	8 bits (cannot be combined)
DAQCard-500	0, 1	4 bits
Lab and 1200 Series Devices DIO-24 Devices	0, 1, 2	8 bits
DIO-96 Devices	0 through 11	8 bits
AT-DIO-32F NB-DIO-32F	0 through 3	8 bits
	4	3 bits (cannot be combined)
SCXI-1160 SCXI-1122	0	16 bits (cannot be combined)
SCXI-1161	0	8 bits (cannot be combined)

Signal Conditioning VIs

The Signal Conditioning VIs are used to convert analog input voltages read from resistance temperature detectors (RTDs), strain gauges, or thermocouples into units of strain or temperature.

You can edit the conversion formulas used in these VIs or replace them with your own to meet the specific accuracy requirements of your application. If you edit or replace the formulas, you should save the new VI in one of your own directories or folders outside of `vi.lib`.

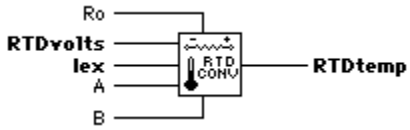
Access the Signal Conditioning VIs by choosing **Functions**»**Data Acquisition**»**Signal Conditioning**. Click on an icon in the following graphic to go to that particular VI description.



Signal Conditioning VIs

Convert RTD Reading

Converts a voltage you read from an RTD into temperature in Celsius.



I32 **RTD volts** is the voltage you read from the RTD.

I32 **Iex** is the excitation current you used with the RTD. This parameter defaults to an excitation current of 0.15 mA.

I32 **Ro** is the RTD resistance at 0º C. The default is 100 .

I32 **A** and **B** are the coefficients of the Callendar Van-Dusen equation that fit your RTD. The default coefficients are those for the European curve (also called the DIN 43760 standard) in the table later in this VI description.

I32 **RTDTemp** is the return temperature value in degrees Celsius.

This VI first finds the RTD resistance by dividing **RTDVolts** by **Iex**. The VI then converts the resistance to temperature using the following solution to the Callendar Van-Dusen equation for RTDs.

Rt = Ro[1 At Bt^2 C(t-100)t^3]

For temperatures above 0º C, the C coefficient is 0, and the preceding equation reduces to a quadratic equation for which the algorithm implemented in the VI gives the appropriate root. So, this conversion VI is accurate only for temperatures above 0º C.

Your RTD documentation should give you **Ro** and the **A** and **B** coefficients for the Callendar Van-Dusen

equation. The most common RTDs are 100-Ω platinum RTDs that either follow the European temperature curve (DIN 43760) or the American curve. The following table gives the values for **A** and **B** for the European and American curves.

European Curve (DIN 43760)	**American Curve**
A = 3.90802e-03 **B** = -5.80195e-07 (α = 0.00385; ∂ = 1.492)	**A** = 3.9784e-03 **B** = -5.8408e-07 (α = 0.00392; ∂ = 1.492)

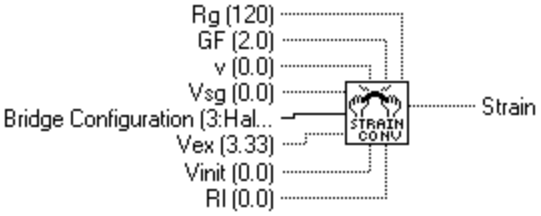
Some RTD documentation gives values for α and ∂, from which you can calculate **A** and **B** using the following equations.

A = α(1 ∂/100)

B = -α∂/100^2

Convert Strain Gauge Reading

Converts a voltage you read from a strain gauge to units of strain.



I32 **Rg (120)** is the strain gauge nominal resistance value in ohms. The default is 120 W.

I32 **Vsg** is the voltage you read from the strain gauge.

I32 **Bridge Configuration (3)** indicates the type of bridge configuration in which the strain gauge is mounted. The default is **Half Bridge II**.

0: Qtr Bridge I
1: Qtr Bridge II
2: Half Bridge I
3: Half Bridge II (default)
4: Full Bridge I
5: Full Bridge II
6: Full Bridge III

The following figures show all the different bridge configurations and the corresponding values that you should pass in **Bridge Configuration**.

I32 **Vex (3.33)** is the excitation voltage you used. This parameter defaults to `3.333` V. (***Macintosh and Windows***) The SCXI-1121 module provides for excitation voltages of `10` V and `3.333` V. The SCXI-1122 module provides for an excitation voltage of `3.333` V.

I32 **Vinit** is the unstrained voltage of the strain gauge after you mount it in its bridge configuration. You should read this voltage at the beginning of your application and save it to pass to this VI.

I32 **GF** is the gauge factor of the strain gauge.

I32 **v** is Poisson's Ratio. (You need this parameter only in certain bridge configurations).

I32 **RI** is the lead resistance. The **RI** parameter defaults to 0. In many cases, the lead resistance is negligible and you can leave this terminal unwired. Otherwise, you can measure **RI** to be more accurate.

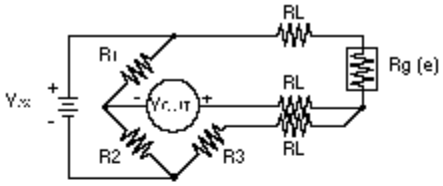
I32 **Strain** is the return strain value.
The conversion formula the VI uses is based solely on the bridge configuration. The following figures show the seven bridge configurations you can use and the corresponding formulas. For all bridge configurations, the VI uses the following formula to obtain **Vr**.

Vr = (**Vsg** - **Vinit**) / **Vex**

In the circuit diagrams, VOUT is the voltage you measure and pass to the conversion VI as the **Vsg** parameter. In the quarter-bridge and half-bridge configurations, R1 and R2 are dummy resistors that are not directly incorporated into the conversion formula. (***Macintosh and Windows***) The SCXI-1121 and SCXI-1122 modules provide R1 and R2 for a bridge-completion network, if needed.

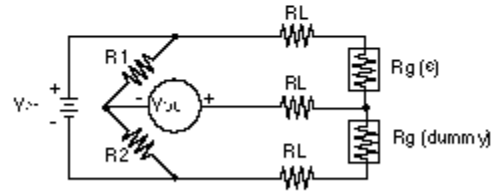
Refer to your *Getting Started with SCXI* manual for more information on bridge-completion networks and voltage excitation.

The following figures illustrate the bridge-completion networks available.



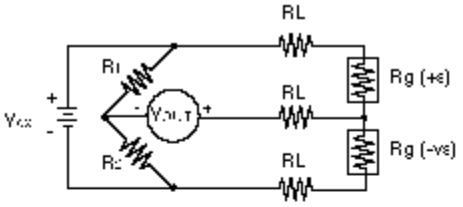
bridgeConfig = 1 (Qtr Bridge I)

$$\text{strain}(\varepsilon) = \frac{-4V_r}{GF(1+2V_r)} \cdot \left(1+\frac{RL}{Rg}\right)$$



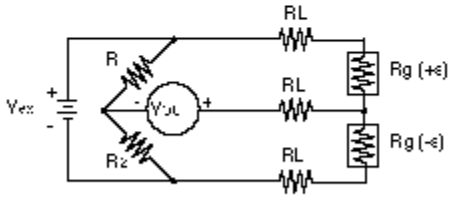
bridgeConfig = 1 (Qtr Bridge II)

$$\text{strain}(\varepsilon) = \frac{-4V_r}{GF(1+2V_r)} \cdot \left(1+\frac{RL}{Rg}\right)$$



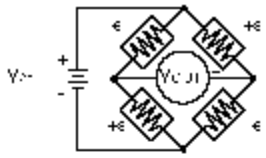
bridgeConfig = 2 (Half Bridge I)

$$\text{strain } (\varepsilon) = \frac{-4V_r}{GF\,[(1+\nu)-2\nu\,(n-1)]} \cdot \left(1+\frac{RL}{Rg}\right)$$



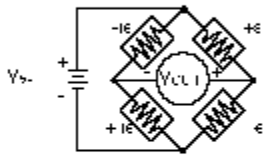
bridgeConfig = 3 (Half Bridge II)

$$\text{strain } (\varepsilon) = \frac{-2V_r}{GF} \cdot \left(1+\frac{RL}{Rg}\right)$$



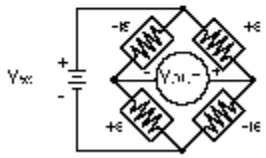
bridgeConfig = 4 (Full Bridge I)

$$\text{strain } (\varepsilon) = \frac{-V_r}{GF}$$



bridgeConfig = 5 (Full Bridge II)

$$\text{strain } (\varepsilon) = \frac{-2V_r}{GF\,(\nu+1)}$$

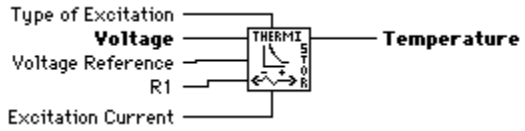


bridgeConfig = 6 (Full Bridge III)

$$\text{strain } (\varepsilon) = \frac{-2V_r}{GF\,[(\nu+1)-\nu\,(n-1)]}$$

Convert Thermistor Reading

Converts a thermistor voltage into temperature. This VI has two different modes of operation for voltage-excited and current-excited thermistors.



I32 **Voltage** is the voltage read from a thermistor.

I32 **Type of Excitation** distinguishes the two modes of operation of the VI. This VI defaults to the `Voltage Reference` mode. Use the **Voltage Reference** (V^REF) and the **R1** values only in this mode. You use the **Excitation Current** (I^EX) value only in the `Current Excitation` mode.

0: Voltage Reference
1: Current Excitation

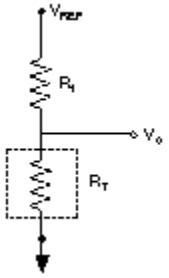
I32 **Voltage Reference** (V^REF) is the reference you apply across a resistor of known value in series with your thermistor. This parameter defaults to `2.5` V, which is the reference voltage used on the SCXI terminal blocks. Use this value only when the **Type of Excitation** is set to `Voltage Reference`.

I32 **R1** is the value of the resistor in series with your thermistor expressed in Ohms. The default is `5` k, which is the **R1** value used on the SCXI terminal blocks. Use this value only when **Type of Excitation** is set to `Voltage Reference`.

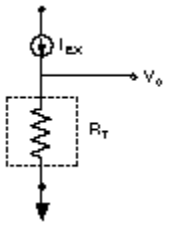
I32 **Excitation Current** (I^EX) is the current excitation applied to the thermistor. This parameter defaults to `100` mA, which is the constant reference current provided by the DAQPad-MIO-16XE-50 device. Use this value only when you set the **Type of Excitation** to `Current Excitation`.

I32 **Temperature** is the return temperature value in degrees Celsius.

This VI has two modes of operation for use with different types of thermistor circuits. The following figure shows how the thermistor can be connected to a voltage reference. This is the setup used in the SCXI-1303, SCXI-1322, SCXI-1327, and SCXI-1328 terminal blocks, which use an onboard thermistor for cold-junction compensation.



The following figure shows a circuit where the thermistor is excited by a constant current source. An example of this setup would be the use of the DAQPad-MIO-16XE-50, which provides a constant current output. The DAQPad-TB-52 has a thermistor for cold-junction sensing.



If the thermistor is excited by voltage, the following shows equation relating the thermistor resistance, RT, to the input values.

$$R_T = R_1 \left(\frac{V_0}{V_{REF} - V_0} \right)$$

If the thermistor is current excited, the equation is

$$R_T = \frac{V_0}{I_{EX}}$$

The following equation is the standard formula the VI uses for converting a thermistor resistance to temperature.

$$T_K = \frac{1}{A + B(\ln R_T) + c(\ln R_T)^3}$$

The values used by this VI for *a*, *b*, and *c* are given below. These values are correct for the thermistors provided on the SCXI and DAQPad-TB-52 terminal blocks. If you are using a thermistor with different values for *a*, *b*, and *c* (refer to your thermistor data sheet), you can edit the VI diagram to use your own *a*, *b*, and *c* values.

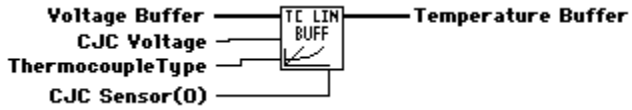
a = 1.295361E-3
b = 2.343159E-4
c = 1.018703E-7

The VI produces a temperature in degrees Celsius. Therefore, TC = TK - 273.15.

I32

Convert Thermocouple Buffer

Converts a voltage buffer read from a thermocouple into a temperature buffer value in degrees Celsius.



I32 **Voltage Buffer** is an array of voltages read from a thermocouple.

I32 **CJC Voltage** is the cold-junction compensation reference voltage. The **CJC Voltage** parameter is either the IC sensor or the thermistor sensor, as specified by **CJC Sensor**.

To obtain **CJC Voltage** on an AMUX-64T, set jumper W2 to the TEMP position and read the voltage at analog input channel 0 in Differential mode. To obtain **CJC Voltage** on an SCXI terminal, read the MTEMP or DTEMP channel, depending on which terminal you have jumper-configured. For more information, refer to the *Temperature Sensors* section in Chapter 19, *Common SCXI Applications*, in the *LabVIEW Data Acquisition Basics Manual*.

I32 **Thermocouple Type** can be B, E, J, K, R, S, or T. Refer to your thermocouple user manual for more information.

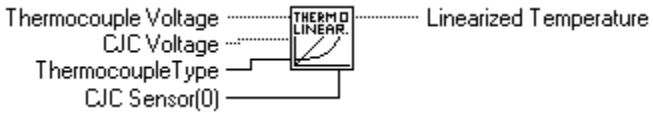
0: B
1: E
2: J
3: K
4: R
5: S
6: T

I32 **CJC Sensor** is the type of cold-junction compensation sensor you are using. A **CJC Sensor** value of 0 selects the IC sensor. This sensor is on the AMUX-64T, SCXI-1300, SCXI-1320, SCXI-1321, and SC-207X series. A **CJC Sensor** value of 1 selects the thermistor sensor. (***Macintosh and Windows***) This sensor is on the SCXI-1328, SCXI-1303, SCXI-1322, and SCXI 1327.

I32 **Temperature Buffer** returns the linearized temperature buffer value in degrees Celsius.

Convert Thermocouple Reading

Converts a voltage read from a thermocouple into a temperature value in degrees Celsius.



I32 **Thermocouple Voltage** is the voltage read from a thermocouple.

I32 **CJC Voltage** is the cold-junction compensation reference voltage. The **CJC Voltage** parameter is either the IC sensor or the thermistor sensor, as specified by **CJC Sensor**.

To obtain **CJC Voltage** on an AMUX-64T, set jumper W2 to the TEMP position and read the voltage at analog input channel 0 in Differential mode. To obtain **CJC Voltage** on an SCXI terminal, read the MTEMP or DTEMP channel, depending on which terminal you have jumper-configured. For more information, refer to the *Temperature Sensors* section under *General Techniques* in Chapter 19, *Common SCXI Applications*, in the *LabVIEW Data Acquisition Basics Manual*.

I32 **Thermocouple Type** can be B, E, J, K, R, S, or T. Refer to your thermocouple user manual for more information.

0: B
1: E
2: J
3: K
4: R
5: S
6: T

I32 **CJC Sensor** is the type of cold-junction compensation sensor you are using. A **CJC** *Sensor value of 0 selects the IC sensor.*

*(**Macintosh and Windows***) This sensor is on the AMUX-64T, SCXI-1300, SCXI-1320, SCXI-1321, and SC-207X series.

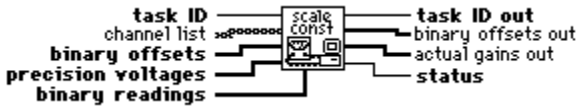
A **CJC Sensor** value of 1 selects the thermistor sensor.

(***Macintosh and Windows***) This sensor is on the SCXI-1328, SCXI-1303, SCXI-1322, and SCXI-1327.

I32 **Linearized Temperature** is the return temperature value in degrees Celsius.

Scaling Constant Tuner

Adjusts the scaling constants, which LabVIEW uses to account for offset and non-ideal gain, to convert analog input binary data to voltage data.



To use this VI correctly, you must first take two analog input readingsa zero offset reading and a known-voltage reading.

The default binary offset for each channel in the group is 0. To determine the actual binary offset for a channel path, ground the channel inputs and take a binary reading, or take multiple binary readings and average them to get fractional LSBs of the offset.

(***Macintosh and Windows***) If you use SCXI, ground the inputs of the SCXI channels to measure the

offset of the entire signal path, including both the SCXI module and the DAQ device. The SCXI-1100, SCXI-1122, SCXI-1160, and SCXI-1141 modules have an internal switch you can use to ground the amplifier inputs without actually wiring the terminals to ground. To use this feature, type the special SCXI string CALGND in your SCXI channel string as described in the *Amplifier Offset* section of Chapter 19, *Common SCXI Applications*, in the *LabVIEW Data Acquisition Basics Manual*. Use intermediate or advanced analog input VIs to get binary data instead of voltage data.

Note: If your device supports dithering, you should enable dither on your DAQ device when you take multiple readings and average them.

LabVIEW assumes the DAQ devices gain settings and SCXI modules are ideal when it scales binary readings to voltage, unless you use this VI to determine actual gain values for the channels. Apply a known precision voltage to each channel and take a binary reading, or take multiple readings from each channel and compute an average binary reading for each channel. Your precision voltage should be about ten times as accurate as the resolution of your DAQ device to produce meaningful results. When you wire **binary readings**, **precision voltages**, and **binary offsets** to this VI, LabVIEW determines the actual gain using the following formula.

$$\text{actual gain} = \frac{\text{voltage resolution} * (\text{binary reading} - \text{binary offset})}{\text{precision voltage}}$$

In this formula, the **voltage resolution** value expressed in volts per LSB and is a value that varies depending on the DAQ device type, the polarity setting, and the input range setting. (***Macintosh and Windows***) For example, the voltage resolution for an NB-MIO-16 or AT-MIO-16 DAQ device in bipolar mode with an input range of -10 to 10 V is 4.88 mV. The VI returns an array of the actual gain values that the VI stores for each channel.

Note: When you take readings to determine the offset and actual gain, you should use the same input limits settings and clock rates that you use to measure your input signals.

LabVIEW uses the following equation to scale binary readings to voltage.

$$\text{actual gain} = \frac{\text{voltage resolution} * (\text{binary reading} - \text{binary offset})}{\text{gain}}$$

When you run the AI Group Config VI, it sets the attributes of all the channels in the group to their defaults, including the binary offset and gain values.

You can wire **channel list** if you want to adjust the scaling constants for a subset of the channels in the group. If you leave **channel list** unwired, the VI adjusts the scaling constants for all channels in the group. The VI uses the same method as the AI Hardware Config VI to apply values in the **binary offsets**, **precision voltages**, and **binary readings** input arrays That is, if you wired channel list first (at index 0) of the input arrays apply to the channels listed at index 0 of **channel list** if you wired **channel list**, or to the channels listed at index 0 of **channel list**. If you leave **channel list** unwired, the first values of the input arrays apply to the first channel in the group. The VI applies the values of each input array to **channel list** channels or the group in this manner until the VI exhausts the arrays. If channels in **channel list** or in the group remain unconfigured, the VI applies the final values in the arrays to all the remaining unconfigured channels.

If you want to adjust only the channel offsets, and you want to assume the gain settings on the DAQ device and SCXI modules are ideal, wire only **binary offsets** and leave **precision voltages** and **binary readings** unwired.

You can also use this VI to retrieve the binary offset and actual gain values for all the channels in the group by wiring **taskID** only.

After you use this VI to adjust the scaling constants for a channel path, any analog input VIs that return

voltage data use the adjusted constants for scaling. You can use the AI Group Config VI to reset the scaling constants for each channel in the group to their default values (zero offset and ideal gain).

I32 **taskID** identifies the group and the I/O operation.

I32 **binary offsets** contains the binary readings taken from each channel after grounding the inputs of the channels. Because the binary offsets have a single precision floating point data type, you can use fractional LSBs of offset.

I32 **precision voltages** contains the precision voltages values that you connect to the channels. If you use the same precision voltage for all the channels, this array can contain one element, which the VI then applies to all the channels.

I32 **binary readings** contains the binary readings taken from each channel after connecting precision voltages to the channels. Because the binary readings have a single precision floating point data type, you can take multiple readings and average them to produce a binary reading with a fractional component.

I32 **channel list**. If **channel list** is empty, the VI adjusts the scaling constants for the entire group. If **channel list** contains one or more channels, the VI adjusts the scaling constants only for the channels listed. See the Channel, Port, and Counter Addressing section of Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, in the *LabVIEW Data Acquisition Basics Manual*, for the syntax used in channels specifications.

I32 **taskID out** has the same value as **taskID in**.

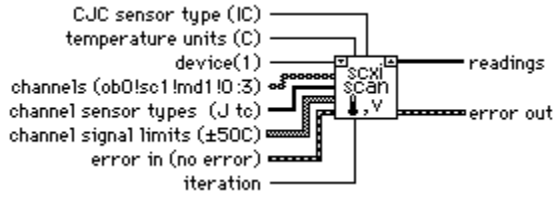
I32 **status** returns any error or warning condition from the VI. Refer to the Data Acquistion VI Error Codes, for a code description.

I32 **binary offsets out** contains the stored binary offsets for all the channels in the group.

I32 **actual gains out** contains the stored actual gains for all the channels in the group. (***Macintosh and Windows***) If any of the channels are SCXI channels, the gain is the total gain on the signal path.

SCXI Temperature Scan

This VI returns a single scan of temperature data from a list of SCXI channel. The SCXI Temperature Scan VI uses averaging to reduce 60 Hz and 50 Hz noise, performs thermocouple linearization, performs offset compensation for the SCXI-1100 module



I32 **CJC sensor type (IC)** is the type of temperature sensor(s) on the SCXI terminal blocks for the modules to be scanned. The VI reads the temperature sensor and uses it to perform cold-junction compensation for the thermocouple readings.

0: IC sensor
1: Thermistor

The SCXI-1300, SCXI-1320, and SCXI-1321 have IC sensors and the SCXI-1303 and SCXI-1328 have thermistors.

I32 **temperature units (C)** specifies the temperature units to use for the return readings from thermocouple channels. The channel signal limits are also in temperature units for thermocouple channels.

0: Celsius
1: Fahrenheit

2: Kelvin
3: Rankine

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **channels (ob0!sc1!md1!0:3)** specifies the list of SCXI analog input channels to scan. The default channel list contains channels 0 through 3 on the module in slot 1 of SCXI chassis 1.

The syntax for SCXI channel strings is:

`obn!scx!mdy!a:b`

where: n is the onboard data acquisition channel
x is the SCXI chassis ID (from the configuration utility)
y is the SCXI slot number of the module
a is the first channel of the SCXI channel range
b is the ending channel in the SCXI channel range

For single-chassis applications, n=0. If you have more than one chassis daisy-chained to the same DAQ device, n=1 for the second DAQ device, and so on. If you only want to acquire from one SCXI channel, omit the `:b` from the end of the string.

When you are scanning multiple SCXI modules, you must scan channels in ascending consecutive order, which means `b` must be greater than `a`. To scan multiple modules, enter an SCXI string like the one above for each module in the channels array, with one element in the array for each module to be scanned.

You can also enter strings for modules in different chassis to be scanned at the same time. Remember to use the correct `n` and `x` values.

I32 **channel sensor types (J tc)** array specifying the sensor type for each channel or range of channels in the channel array. Element i from the channel sensor types array specifies the sensor type for all channels in element i from the channels array. The valid sensor types include the following.

0: B thermocouple
1: E thermocouple
2: J thermocouple
3: K thermocouple
4: R thermocouple
5: S thermocouple
6: T thermocouple
7: Voltage

The sensor type determines the units for return readings, as well as the units expected for channel signal limits.

I32 **channel signal limits (±50C)** specifies the input signal limits for each channel or range of channels in the channels array. LabVIEW uses the limits to select optimum gain, range, and polarity settings for the DAQ device and SCXI modules (if those settings are programmable).

Element `i` from the channel signal limits array specifies the expected high and low input signal limits for all channels in element `i` from the channels array.

The limit values are expected in units determined by the **channel sensor types**. If the channel sensor type for the corresponding channels is a thermocouple type, the signal limits are expected in temperature units (determined by the **temperature units** control). If the sensor type is voltage, the limits should be in volts.

If you leave the array empty, LabVIEW uses a default gain of 1 for the DAQ device and the default SCXI-1100 gain from the configuration utility. If you specify high and low limits, LabVIEW ignores the configuration utility gain settings and selects new gains settings to achieve your designated input limits.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

iteration. When you call this VI inside a loop in your diagram, you should wire the loop iteration variable to this iteration input. However, if this VI is to be executed for more than 2^31 iterations, you should use a different technique to avoid integer overflow. Once the loop executes 2^31 times, the value of the iteration terminal overflows and becomes negative, causing this VI to reinitialize.

The first time your loop executes and calls the VI (when **iteration** = 0), the VI will read the cold-junction temperature sensor(s), perform amplifier offset compensation (if you are using SCXI-1100), and will configure the analog input task with the channel list and signal limits. Then, it will acquire, retrieve, average, and convert the data. On subsequent calls to this VI (when iteration>0), the VI will only acquire, retrieve, average, and convert the data, thus executing much faster.

If you are only calling this VI one time in your diagram, you can leave iteration unwired so it defaults to 0 and it will perform all configuration by default.

readings is a return array containing one data point from each channel specified in the channels array. The data for thermocouple channels is in temperature units and for voltage channels is in volts.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Convert RTD Reading VI

Convert RTD Reading

Convert Strain Gauge Reading VI

Convert Strain Gauge Reading

Convert Thermistor Reading VI

Convert Thermistor Reading

Convert Thermocouple Buffer VI

Convert Thermocouple Buffer

Convert Thermocouple Reading VI

Convert Thermocouple Reading

Scaling Constant Tuner VI

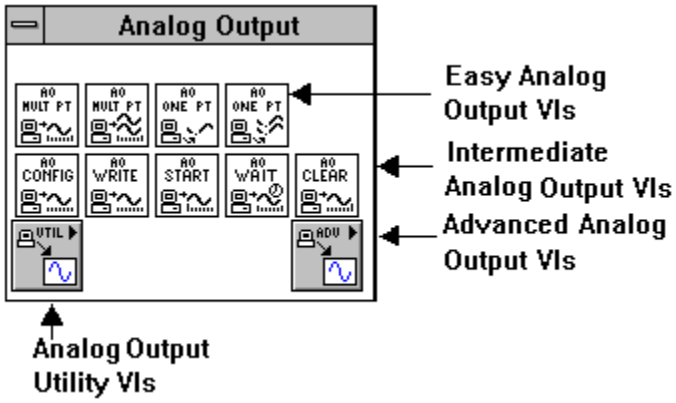
Scaling Constant Tuner

SCXI Temperature Scan VI

SCXI Temperature Scan

Analog Output VIs

The Analog Output VIs can be found by choosing **Functions**»**Data Acquisition**»**Analog Output**. When you click on the **Analog Output** icon in the **Data Acquisition** palette, the Analog Output palette pops up. Click on the labels in the following illustration for overview information about the VIs and on the actual VI icon for information on the VI.



For specific Analog Output VI information, see the following topics:

Easy Analog Output VIs
Intermediate Analog Output VIs
Analog Output Utility VIs
Advanced Analog Output VIs

Easy Analog Output VIs (Overview)

The Easy Analog Output VIs are stand-alone executables, which means you only need one Easy Analog Output VI to perform each basic analog output operation. Unlike intermediate- and advanced-level VIs, Easy Analog Output VIs automatically alert you to errors with a dialog box that asks you to stop the execution of the VI or to ignore the error.

The Easy Analog Output VIs are actually composed of Intermediate Analog Output VIs, which are in turn composed of Advanced Analog Output VIs. The Easy Analog Output VIs provide a basic, convenient interface with only the most commonly used inputs and outputs. For more complex applications, use the intermediate- or advanced-level VIs for more functionality and performance.

Click here to go to the Easy Analog Output VIs for specific VI information.

Intermediate Analog Output VIs (Overview)

You can find intermediate-level Analog Output VIs in two different places in the Analog Output palette. You can find the Intermediate Analog Output VIs in the second row of the Analog Output palette. The other intermediate-level VIs are in the Analog Output Utilities palette. The Intermediate Analog Output VIs are built from the fundamental building block layer, called the Advanced Analog Output VIs. These VIs offer almost as much power as the advanced-level VIs, and they conveniently group the advanced-level VIs into a tidy, logical sequence.

Click here to go to the Intermediate Analog Output VIs for specific VI information.

Analog Output Utility VIs (Overview)

You can access the **Analog Output Utilities** subpallete by choosing the Analog Output Utility icon from the **Analog Output** palette. The Analog Output Utility are single-VI solutions to common analog output problems. These VIs are convenient, but they lack flexibility. These three VIs are built from the Intermediate Analog Output VIs.

Click here to go to the Analog Output Utility VIs for specific VI information.

Advanced Analog Output VIs (Overview)

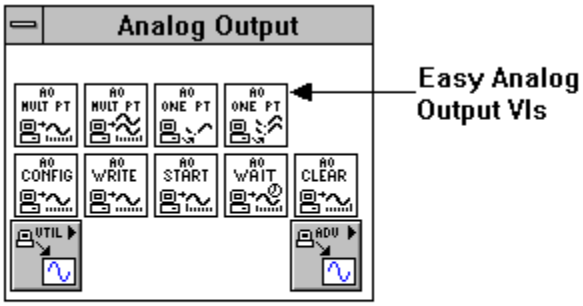
You can access the **Advanced Analog Output** subpalette by choosing the Advanced Analog Output icon from the **Analog Output** palette. These VIs are the interface to the NI-DAQ software and are the foundation of the Easy, Utility, and Intermediate Analog Output VIs. Because all these VIs rely on the advanced-level VIs, refer to Advanced Analog Output VIs, for additional information on the inputs and outputs and how they work.

Click here to go to the Advanced Analog Output VIs for specific VI information.

Easy Analog Output VIs

The Easy Analog Output VIs perform simple analog output operations in LabVIEW. You can run these VIs from the front panel or use them as subVIs in basic applications.

Access the Easy Analog Output VIs by choosing **Functions**»**Data Acquisition**»**Analog Output**. The Easy Analog Output VIs are the VIs on the top row of the **Analog Output** palette. Click on an icon in the following illustration to go to a particular Easy Analog Output VI.



For examples of how to use the Easy Analog Output VIs, open the example library by opening `examples\daq\anlogout.llb`.

Easy VIs

AO Generate Waveform
AO Generate Waveforms
AO Update Channel
AO Update Channels

AO Generate Waveform

Generates a voltage waveform on an analog output channel at the specified update rate.

The AO Generate Waveform VI generates a multipoint voltage waveform on a specified analog output channel. If an error occurs, a dialog box appears, giving you the option to stop the VI or continue.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **channel** is a list of the analog channels you want to use. If `x`, `y`, and `z` refer to channels, you can specify a list of channels by separating the individual channels with commasfor example, `x,y,z`. If `x` refers to the first channel in a consecutive channel range and `y` refers to the last channel, you can specify the range by separating the first and last channels by a colon (for example, `x:y)`. Refer to the *Channel Addressing with the AMUX-64T* section of Chapter 5, *Things You Should Know about Analog Input*, in the *LabVIEW Data Acquisition Basics Manual*, for the syntax you use for AMUX-64T channels or SCXI.

I32 **update rate** is the number of voltage updates to generate per second. The default rate is `1000` updates/s.

I32 **waveform** is a 1D array that contains data in volts to be written to the analog output channel. You

must supply this data.
Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the channel numbers you can use with your DAQ device.

AO Generate Waveforms

Generates multiple voltage waveforms on the specified analog output channels at the specified update rate.

If an error occurs, a dialog box appears, giving you the option to stop the VI or continue.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

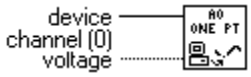
I32 **channels** is a list of the analog channels you want to use. If `x`, `y`, and `z` refer to channels, you can specify a list of channels by separating the individual channels with commas (for example, `x,y,z)`. If `x` refers to the first channel in a consecutive channel range and `y` refers to the last channel, you can specify the range by separating the first and last channels by a colon (for example, `x:y)`. Refer to the *Channel Addressing with the AMUX-64T* section of Chapter 5, *Things You Should Know about Analog Input*, in the *LabVIEW Data Acquisition Basics Manual*, for the syntax you use for AMUX-64T channels or SCXI.

I32 **update rate** is the number of voltage updates to generate per second. The default rate is `1000` updates/s.

I32 **waveforms** is a 2D array that contains data to be written to the analog output channels in volts. The array must be ordered by updates. That is, each row contains the data for one update of all the channels and each column has the updates of one channel. The channel order of the data must be identical to the channel order you specify in **channels**. You must specify **waveforms**, where the row is the update number and the column is the channel number. Refer to the *Data Organization for Analog Applications* section of Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, in the *LabVIEW Data Acquisition Basics Manual* for more information.
Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the channel numbers you can use with your DAQ device.

AO Update Channel

Writes a specified voltage value to an analog output channel.



The AO Update Channel VI writes a single update to an analog output channel. If an error occurs, a dialog box appears, giving you the option to stop the VI or continue.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

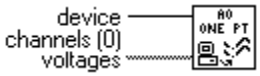
I32 **channel** is a list of the analog channels you want to use. If `x`, `y`, and `z` refer to channels, you can specify a list of channels by separating the individual channels with commas (for example, `x,y,z)`. If `x` refers to the first channel in a consecutive channel range and `y` refers to the last channel, you can specify

the range by separating the first and last channels by a colon (for example, x:y). Refer to the *Channel Addressing with the AMUX-64T* section of Chapter 5, *Things You Should Know about Analog Input*, in the *LabVIEW Data Acquisition Basics Manual*, for the syntax you use for AMUX-64T channels or SCXI.

I32 **voltage** contains the voltage value to be written to the specified analog output channel. Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the channel numbers you can use with your DAQ device.

AO Update Channels

Writes voltage values to each of the specified analog output channels.



The AO Update Channels VI updates multiple analog output channels with single voltage values. If an error occurs, a dialog box appears, giving you the option to stop the VI or continue.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **channels** is a list of the analog channels you want to use. If x, y, and z refer to channels, you can specify a list of channels by separating the individual channels with commas (for example, x,y,z). If x refers to the first channel in a consecutive channel range and y refers to the last channel, you can specify the range by separating the first and last channels by a colon (for example, x:y). Refer to the *Channel Addressing with the AMUX-64T* section of Chapter 5, *Things You Should Know about Analog Input*, in the *LabVIEW Data Acquisition Basics Manual*, for the syntax you use for AMUX-64T channels or SCXI.

I32 **voltages** is a 1D array that contains the analog output data in volts. You must supply this data. Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the channel numbers you can use with your DAQ device.

AO Generate Waveform VI

AO Generate Waveform

AO Generate Waveforms VI

AO Generate Waveforms

AO Update Channel VI

AO Update Channel

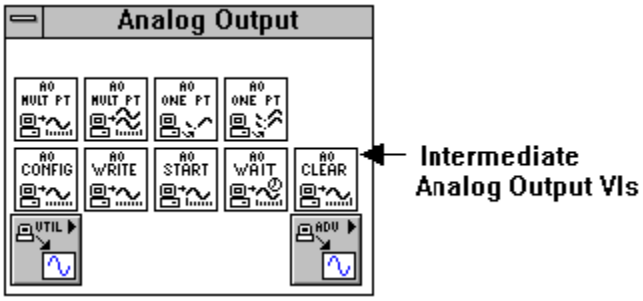
AO Update Channels VI

AO Update Channels

Intermediate Analog Output VIs

The Intermediate Analog Output VIs are built from the fundamental building block layer, called the Advanced Analog Output VIs. These VIs offer almost as much power as the advanced-level VIs, and they conveniently group the advanced-level VIs into a tidy, logical sequence.

To access the Intermediate Analog Output VIs, choose **Functions**»**Data Acquisition**»**Analog Output**. The Intermediate Analog Output VIs are the VIs on the second row of the **Analog Output** palette. Click on an icon in the following illustration to go to a particular Intermediate Analog Output VI.



Click here for special information on Error Handling for Intermediate AO VIs .

For examples of how to use the Intermediate Analog Output VIs, open the example library by opening `examples\daq\anlogout.llb`.

Intermediate VIs

AO Clear
AO Config
AO Start
AO Wait
AO Write

AO Clear

Clears the analog output task associated with **taskID in**.

The AO Clear VI uses the AO Control VI to stop analog output generation associated with **taskID in** and releases associated internal resources, including buffers. Before beginning a new signal generation, call the AO Config VI or the AO Buffer Config VI.

The AO Clear VI always clears the generation regardless of whether **error in** indicates an error.

I32 **taskID** identifies the group and the I/O operation.

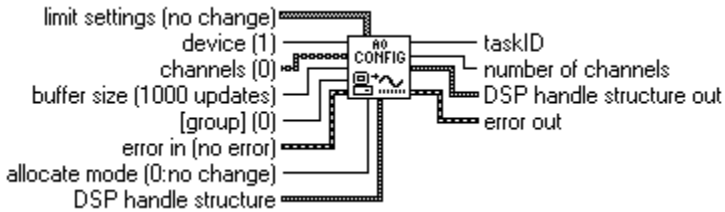
I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

AO Config

Configures the channel list and output limits, and allocates a buffer for analog output operation.



If an error occurs, **taskID** is 0. Otherwise, this VI calls several advanced analog output VIs to set up a task for a buffered analog output. The AO Config VI calls the AO Group Config VI for the specified device, group, and channels to create a taskID. The AO Config VI then calls the AO Hardware Config VI to record information about the hardware configuration, and the AO Buffer Config VI to allocate a buffer for buffered waveform generation.

Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the channel ranges and output limits available with your DAQ device.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **channels** is a list of the analog channels you want to use. If `x`, `y`, and `z` refer to channels, you can specify a list of channels by separating the individual channels with commas (for example, `x,y,z)`. If `x` refers to the first channel in a consecutive channel range and `y` refers to the last channel, you can specify the range by separating the first and last channels by a colon (for example, `x:y)`. Refer to the *Channel Addressing with the AMUX-64T* section of Chapter 5, *Things You Should Know about Analog Input*, in the *LabVIEW Data Acquisition Basics Manual*, for the syntax you use for AMUX-64T channels or SCXI.

I32 **buffer size** is the number of updates you want each buffer to hold. This parameter defaults to an input of 1000 updates.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **limit settings** is an array of clusters, of which each array element specifies limits for the channels in the corresponding element of the **channels** array. If the length of **limit settings** is less than the number of **channels**, the values in the last element of **limit settings** apply to the remaining channels. The default for **limit settings** is an empty array, which means the limits do not change from their default settings. Each cluster contains the following parameters. See the AO Hardware Config VI description for more information.

I32 **upper limit** is equal to your reference voltage and is the maximum voltage the DAC produces.

I32 **lower limit** is either 0.0 V (which implies a unipolar setting) or the negative value of the upper limit (which implies a bipolar setting).

I32 **reference source**.

0: Do not change the reference source setting (default input).
1: Internal (default setting).
2: External.

I32 **group** is the number, from 0 to 15, that you assign to the specified set of channels. The default input and setting for **group** is 0. Usually, you only have one output operation for this device, so you can leave this input unwired and use group 0. Otherwise, you must assign a unique group number to each output operation for a given device.

I32 **allocate mode**.

0: Do not change allocate mode (default input).
3: Allocate memory (default setting).

4: Allocate DSP memory. **DSP handle structure out** contains the resulting handle.
5: Use DSP memory that was previously allocated. The **DSP handle structure** control points to this memory.
6: Use onboard FIFO memory. The entire waveform must fit in the FIFO, and you cannot write after the waveform starts.
(Windows) You can use this mode only with devices that have an analog output FIFO, namely the AT-AO-6/10, AT-MIO-16X, AT-MIO-16E-1, AT-MIO-64F-5, AT-MIO-16E-2, AT-MIO-64E-3, and NEC-MIO-16E-4.

Note: Modes 4, 5, and 6 are not available on the Sun or Macintosh.

I32 **DSP handle structure** contains the following parameters. You use this cluster only when **allocate mode** is 5.

I32 **size** is unused by AO Config. Even when **allocate mode** isÊ5, the buffer size is specified by the **buffer size** control.

I32 **DSP memory handle** specifies a block of DSP memory.

I32 **taskID** identifies the group and the I/O operation.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

I32 **number of channels** is the total number of channels that belong to the group.

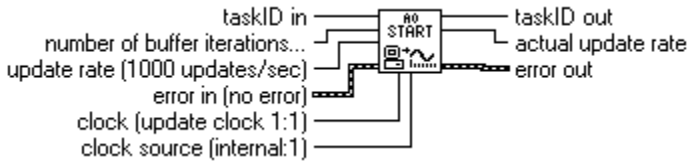
I32 **DSP handle structure out** contains the following output parameters when **allocate mode** is 4 or 5.

I32 **size** is unused by the AO Config VI. Even when **allocate mode** is 5, the buffer size is specified by the **size** control.

I32 **DSP memory handle** specifies a block of DSP memory.

AO Start

Starts a buffered analog output operation. This VI sets the update rate and then starts the generation.



The AO Start VI calls the AO Clock Config VI with the specified **taskID** in to configure the output operation using the specified **clock**, **clock source** (internal by default), and **update rate**. The AO Start VI then calls the AO Control VI to start the generation, using **number of buffer iterations** to determine how many times to generate the buffer contents.

Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the clocks available with your DAQ device.

I32 **taskID in** identifies the group and the I/O operation.

I32 **update rate** is the number of voltage updates to generate per second. The default rate is `1000` updates/s.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **number of buffer iterations** is the number of times LabVIEW should generate the waveform from the output buffer. After generating the buffer the specified number of times, the generation stops. The default value is `1`, which means LabVIEW generates the buffer only once. If you use a value of `0`, LabVIEW generates the buffer continuously, until you stop the operation with the AO Clear VI.

I32 **clock**.
0: Do not change the **clock** setting.
1: Update clock 1 (default setting).
2: Interval clock 1.
3: Update clock 2.

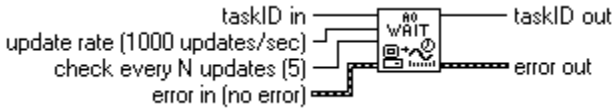
Refer to the AO Clock Config VI description for more information about the **clock** settings.

I32 **clock source**.
0: Do not change the **clock source** setting (default input).
1: Internal (LabVIEW picks one) (default setting).
2: Counter 1. This option is only supported on LabVIEW for Windows.
3: Counter 2. This option is only supported on LabVIEW for Windows.
4: Counter 3. This option is only supported on LabVIEW for Windows.
5: Counter 5. This option is only supported on LabVIEW for Windows.
6: I/O connector.
7: (***Macintosh and Windows***) RTSI connection.

I32 **taskID out** has the same value as **taskID in**.
I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.
I32 **actual update rate** may differ slightly from the requested update rate, depending on the hardware capabilities.

AO Wait

Waits until the waveform generation of the task completes before returning.

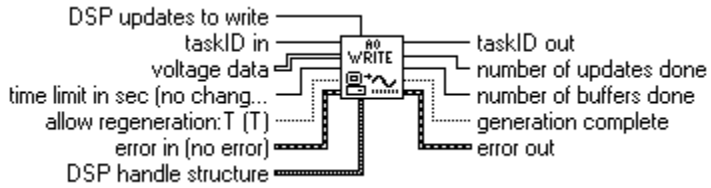


Use the AO Wait VI to wait for a buffered, finite waveform generation to finish before calling the AO Clear VI. The AO Wait VI checks the status of the task at regular intervals by calling the AO Write VI and checking its **generation complete** output. The AO Wait VIwaits asynchronously between intervals to free the processor for other operations. The VI calculates the wait interval by dividing the **check every N updates** input by the update rate. You should not use the AO Wait VI when you generate data continuously, because the generation never finishes. The AO Clear VI stops a continuous waveform generation.

I32 **taskID in** identifies the group and the I/O operation.
I32 **update rate** is the number of voltage updates to generate per second. The default rate is `1000` updates/s.
I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.
I32 **check every N updates** tells the VI how often to check the status of the task to see if generation completes. This parameter default is to check every 5 updates.
I32 **taskID out** has the same value as **taskID in**.
I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

AO Write

Writes data into the buffer for a buffered analog output operation.



This VI calls the AO Buffer Write VI to write data to an analog output buffer.

taskID in identifies the group and the I/O operation.

voltage data is a 2D array that contains analog data. (The AI Read One Scan, AI Single Scan, AO Write One Update, and AO Single Update VIs return only a 1D array in the **voltage data** parameter.) This array is scan-ordered which means that each row contains the data of a single scan or value. If you want to index the array in a loop, you will get a scan on each iteration. Each column holds the data of a single channel. The upper array index selects the scan and the lower index selects the channel. Refer to the Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, in *the LabVIEW Data Acquisition Basics Manual* for more information about the array format.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

DSP updates to write tells LabVIEW how many updates to write when the source data resides in DSP memory.

time limit in sec is the time limit for the output operation. The VI cannot write new data into an area of the buffer currently being generated and must wait until that part of the data has been generated before overwriting it. You can use **time limit in sec** to limit this waiting period. The default input is -1.0, which means LabVIEW calculates the time limit based on how long it takes for the waveform generation to be ready for new data. The resolution of the timeout clock is about 17 ms (Macintosh), 55 ms (Windows), 1 ms (Sun).

allow regeneration. *Regeneration* means generating the same data more than once. Data is fresh until it has been generated. Once the data has been generated, it is stale. If **allow regeneration** is TRUE (default value), when all the data in the buffer has been generated, LabVIEW starts generating again from the beginning of the buffer, regardless of whether the data there is fresh or stale. If **allow regeneration** is FALSE, you must write fresh data to the buffer fast enough and in such quantities that there is always fresh data available to generate. The VI generates an error if fresh data is unavailable. Typically, you will fill the buffer before starting and write additional fresh data in a loop. If there is no room in the buffer, the VI waits to return until there is room. You can write any amount of data at a time.

When DMA is used, LabVIEW examines your waveform for stale data, only at the halfway point and at the end of the buffer. So, when DMA is used, LabVIEW cannot detect when stale data is about to be regenerated and stop immediately. Because LabVIEW can only stop halfway through or at the end of your waveform, some stale data is likely to be regenerated. When interrupts are used, LabVIEW examines every point for stale data and stops immediately when it detects stale data.

If **allow regeneration** is FALSE, your buffer must be at least twice the size of your analog output FIFO. Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual* to see if your devices FIFO size, if it has FIFO.

When the AO Write VI executes, after waveform generation has begun, **allow regeneration** is ignored. The value of **allow regeneration** determines how LabVIEW reacts when it encounters stale data.

DSP handle structure contains the following parameters. You use this cluster when the source of your waveform data resides in DSP memory. In this case, do not wire the **voltage data** input.

size is not used by AO Write.

DSP memory handle specifies the start of a block of DSP memory. **DSP updates to write** tells

LabVIEW how much of the block to write.


taskID out has the same value as **taskID in**.


error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.


number of updates done is the number of updates the VI has generated; that is, the number of updates the VI has actually transferred from the buffer to the DAC.


entire buffer done is the number of times the VI has generated an entire buffer; that is, the number of times the VI has actually transferred all the data in the buffer to the DAC.


generation complete is TRUE when the generation finishes.

AO Clear VI

AO Clear

AO Config VI

AO Config

AO Start VI

AO Start

AO Wait VI

AO Wait

AO Write VI

AO Write

Handling Errors

LabVIEW makes error handling easy with the Intermediate Analog Output VIs. Each intermediate-level VI has an **error in** input cluster and an **error out** output cluster. The clusters contain a Boolean that indicates whether an error occurred, the error code for the error, and the name of the VI that returned the error. If **error in** indicates an error, the VI returns the error information in **error out** and does not continue to run.

Note: The AO Clear VI is an exception to this ruleÐthis VI always clears the acquisition regardless of whether error in indicates an error.

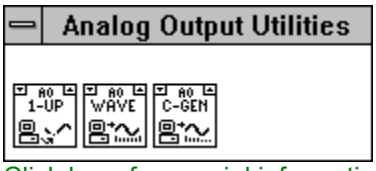
When you use any of the Intermediate Analog Output VIs in a While Loop, you should stop the loop if the **status** in the **error out** cluster reads TRUE. If you wire the error cluster to the General Error Handler VI, the VI deciphers the error information and describes the error to you.

The General Error Handler VI is in **Functions»Utilities** in LabVIEW. For more information on this VI, refer to the topic, General Handler VI.

Analog Output Utility VIs

Analog Output Utility VIs VIs are intermediate-level VIs that rely on advanced-level VIs. Refer to the topic Advanced Analog Output VIs, for additional information on the inputs and outputs and how they work. For more general information about these VIs, see Analog Output Utility VIs (Overview).

To access the **Analog Output Utilities,** palette choose **Functions**»**Data Acquisition**»**Analog Output**»**Analog Output Utilities**. The icon that you must select to access the Analog Output Utility VIs is on the bottom row of the **Analog Output** palette. When you click on the Analog Output Utilities icon, you will see the **Analog Output Utility** palette, as shown in the following illustration. Click on an icon in the following illustration for pop-up help.



Click here for special information on Error Handling for AO Utility VIs.

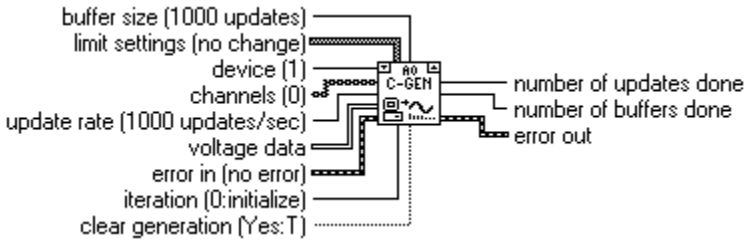
For examples of how to use the Analog Output Utility VIs, open the example library by opening `examples\daq\anlogout.llb`.

Utility VIs

AO Continuous Gen
AO Waveform Gen
AO Write One Update

AO Continuous Gen

Generates a continuous, timed, circular-buffered waveform for the given output channels at the specified update rate. The VI updates the output buffer continuously as it generates the data. If you simply want to generate the same data continuously, use the AO Waveform Gen VI instead.



You use the AO Continuous Gen VI when your waveform data resides on disk and is too large to hold in memory, or when you must create your waveform in real time. Place the VI in a While Loop and wire the iteration terminal to the VI **iteration** input.

Note: If your program iterates more than 2^31-1 times, do not wire this VI iteration terminal to the loop iteration terminal. Instead, set iteration to 0 on the first loop, then to any positive value on all other iterations. The VI reconfigures and restarts if iteration is less than or equal to 0.

Also wire the condition that terminates the loop to the VI's **clear acquisition** input, inverting the signal if necessary so that it is `TRUE` on the last iteration. On iteration 0, the VI calls the AO Config VI to configure the channel group and hardware and to allocate a buffer for the data. It also calls the AO Write VI to write the given voltage data into the buffer, and then the AO Start VI to set the update rate and start the signal generation. On each subsequent iteration, the VI calls the AO Write VI to write the next portion of data

into the buffer at the current write position. On the last iteration (when **clear generation** is TRUE) or if an error occurs, the VI also calls the AO Clear VI to clear any generation in progress. Although it is not normally necessary, you can call the AO Continuous Gen VI outside of a loop (that is, to call it only once). But if you do, leave the **iteration** and **clear generation** inputs unwired.

The first call to the AO Write VI sets **allow regeneration** to TRUE, so that the same data can be generated more than once. If you change **allow regeneration** to FALSE, you must write new data fast enough that new data is always available to be generated. If you do not fill the buffer fast enough, you get a regeneration error. To correct this problem, decrease the **update rate**, increase the **buffer size**, increase the amount of data written each time, or write data more often.

(***Windows***) If you set **allow regeneration** to FALSE, and your device has an analog output FIFO, your **buffer size** must be at least twice as big as your FIFO.

If an error occurs, the VI calls the AO Clear VI to clear any generation in progress, then passes the unmodified error information to **error out**. If an error occurs inside the AO Continuous Gen VI, the AO Clear VI clears any generation in progress and passes its error information out.

Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the channel ranges, output limits, and scanning order available with your DAQ device.

Note: The AO Continuous Gen VI uses an uninitialized shift register as local memory to remember the taskID of the output operation between calls. You normally use this VI in one place on a diagram, but if you use it in more than one place, the multiple instances of the VI share the same taskID. All calls to this VI configure, write data, or clear the same generation. Occasionally, you may want to use this VI in multiple places on the diagram but have each instance refer to a different taskID (for example, when you want to generate waveforms with two devices simultaneously). Save a copy of this VI with a new name (for example, AO Continuous Gen R) and make your new VI reentrant See the topic Reentrant Execution for information on creating a VI reentrant.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **channels** is a list of the analog channels you want to use. If x, y, and z refer to channels, you can specify a list of channels by separating the individual channels with commas (for example, x,y,z). If x refers to the first channel in a consecutive channel range and y refers to the last channel, you can specify the range by separating the first and last channels by a colon (for example, x:y). Refer to the *Channel Addressing with the AMUX-64T* section of Chapter 5, *Things You Should Know about Analog Input*, in the *LabVIEW Data Acquisition Basics Manual*, for the syntax you use for AMUX-64T channels or SCXI.

I32 **update rate** is the number of voltage updates to generate per second. The default rate is 1000 updates/s.

I32 **voltage data** is a 2D array that contains analog data. (The AI Read One Scan, AI Single Scan, AO Write One Update, and AO Single Update VIs return only a 1D array in the **voltage data** parameter.) This array is scan-ordered which means that each row contains the data of a single scan or value. If you want to index the array in a loop, you will get a scan on each iteration. Each column holds the data of a single channel. The upper array index selects the scan and the lower index selects the channel. Refer to the Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, in the *LabVIEW Data Acquisition Basics Manual* for more information about the array format.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **iteration** can be used to optimize operation when you execute this VI in a loop. When **iteration** is

0 (default), LabVIEW calls the AO Config and AO Start VIs to configure the channels. If **iteration** is greater than zero, LabVIEW uses the existing configuration, which improves performance. You usually wire this input to an iteration terminal.

I32 **clear generation**. If you set this parameter to TRUE, the VI calls the AO Clear VI when the generation is cleared.

I32 **buffer size** is the number of scans (with input operations) or updates (with output operations) you want the circular buffer to hold.

I32 **limit settings** is an array of clusters. Each array element specifies the limits for the channel or channels specified by the corresponding element of **channels**. If the length of the **limit settings** array is less than the number of **channels**, the values in the last element of the **limit settings** array apply to the remaining channels. The **limit settings** cluster defaults to an empty array, which means the limit default does not change from their default for any channel. Each cluster contains the following parameters. See the AO Hardware Config VI description in this chapter for more information.

I32 **upper limit** is equal to your reference voltage and is the maximum voltage the DAC can produce.

I32 **lower limit** is either 0.0 V (which implies a unipolar setting) or the negative value of the upper limit (which implies a bipolar setting).

I32 **reference source**.

0: Do not change the **reference source** setting (default input).
1: Internal (default setting).
2: External.

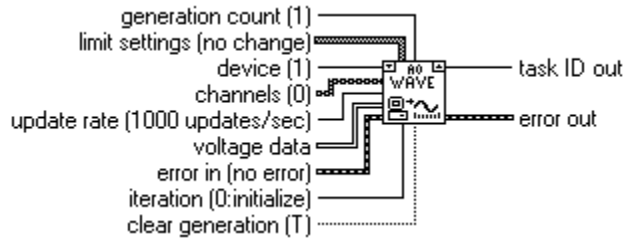
I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

I32 **number of updates done** is the number of updates the VI has generated so farthat is, the number of updates the VI has actually transferred from the buffer to the DAC.

I32 **number of buffers done** is the number of times the VI has generated, or the number of times the VI has actually transferred, all the data in the buffer to the DAC.

AO Waveform Gen

Generates a timed, simple-buffered or circular-buffered waveform for the given output channels at the specified update rate. Unless you perform indefinite generation, the VI returns control to the LabVIEW diagram only when the generation completes.



If you place this VI in a loop to generate multiple waveforms with the same group of channels, wire the iteration terminal to the VI **iteration** input.

Note: If your program iterates more than 2^31-1 times, do not wire this VI iteration terminal to the loop iteration terminal. Instead, set the iteration value to 0 on the first loop, then to any positive value on all other iterations. The VI reconfigures and restarts if iteration is less than or equal to 0.

On iteration 0, the VI calls the AO Config VI to configure the channel group and hardware and to allocate a buffer for the data. On each iteration, the VI calls the AO Write VI to write the data into the buffer, then the AO Start VI to set the update rate and start the generation. If you call the AO Waveform Gen VI only once, you can leave **iteration** unwired. The **iteration** parameter defaults to 0, which tells the VI to

configure the device before starting the waveform generation.

If an error occurs, the VI calls the AO Clear VI to clear any generation in progress, then passes the error information unmodified through **error out**. If an error occurs inside the AO Waveform Gen VI, it clears any generation in progress and passes its error information out.

Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the channel ranges, output limits, and scanning order available with your DAQ device.

Note: The AO Waveform Gen VI uses an uninitialized shift register as local memory to remember the taskID of the output operation between calls. You normally use this VI in one place on your diagram, but if you use it in multiple places, all instances of the VI share the same taskID. All calls to this VI configure, write data, or clear the same generation. Occasionally, you may want to use this VI in multiple places on the diagram, but have each instance refer to a different taskID. Save a copy of this VI with a new name (for example, AO Waveform Gen R) and make the new VI reentrant. See the topic Reentrant Execution for information on creating a VI reentrant.

device is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

channels is a list of the analog channels you want to use. If `x`, `y`, and `z` refer to channels, you can specify a list of channels by separating the individual channels with commas (for example, `x,y,z)`. If `x` refers to the first channel in a consecutive channel range and `y` refers to the last channel, you can specify the range by separating the first and last channels by a colon (for example, `x:y)`. Refer to the *Channel Addressing with the AMUX-64T* section of Chapter 5, *Things You Should Know about Analog Input*, in the *LabVIEW Data Acquisition Basics Manual*, for the syntax you use for AMUX-64T channels or SCXI.

update rate is the number of voltage updates to generate per second. The default rate is `1000` updates/s.

voltage data is a 2D array that contains analog data. (The AI Read One Scan, AI Single Scan, AO Write One Update, and AO Single Update VIs VIs return only a 1D array in the **voltage data** parameter.) This array is scan-ordered which means that each row contains the data of a single scan or value. If you want to index the array in a loop, you will get a scan on each iteration. Each column holds the data of a single channel. The upper array index selects the scan and the lower index selects the channel. Refer to the Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, in *the LabVIEW Data Acquisition Basics Manual* for more information about the array format.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

clear generation. If you set this parameter to TRUE, the VI calls the AO Clear VI when the generation is cleared.

generation count tells LabVIEW how many times to generate the data in the **voltage data** array. The default setting is 1. A value of 0 signifies continuous generation.

limit settings is an array of clusters. Each array element specifies the limits for the channel or channels specified by the corresponding element of **channels**. If the length of the **limit settings** array is less than the length of **channels**, the values in the last element of the **limit settings** array apply to the remaining channels. The default for **limit settings** is an empty array, which means the limits do not change from their default for any channel. Each cluster contains the following parameters. See the AO Hardware Config VI description in this chapter for more information.

upper limit is equal to your reference voltage and is the maximum voltage the DAC can produce.

lower limit is either 0.0 V (which implies a unipolar setting) or the negative value of the upper limit (which implies a bipolar setting).

I32 **reference source**.

0: Do not change the **reference source** setting (default input).
1: Internal (default setting).
2: External.

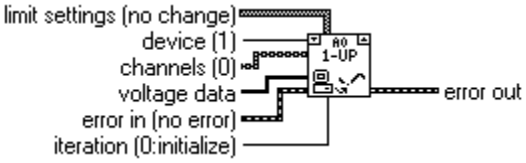
I32 **iteration** can be used to optimize operation when you execute this VI in a loop. When **iteration** is 0 (default), LabVIEW calls the AO Config VI to configure the port. If **iteration** is greater than zero, LabVIEW uses the existing configuration, which improves performance. You usually wire this input to an iteration terminal.

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

AO Write One Update

Writes a single voltage value to each of the specified analog output channels.



The AO Write One Update VI performs an immediate, untimed update of a group of one or more channels. If you place the VI in a loop to write more than one value to the same group of channels, wire the iteration terminal to the VI **iteration** input.

Note: If your program iterates more than 2^31-1 times, do not wire this VI iteration terminal to the loop iteration terminal. Instead, set the iteration value to 0 on the first loop, then to any positive value on all other iterations. The VI reconfigures and restarts if iteration is less than or equal to 0.

On iteration 0, the VI calls the AO Config VI to configure the channel group and hardware, then calls the AO Single Update VI to write the voltage to the output channels. On future iterations, the VI calls only the AO Single Update VI, avoiding unnecessary configuration. If you call the AO Write One Update VI only once to write a single value to each channel, leave the **iteration** input unwired. Its default value of `0` tells the VI to perform the configuration before writing any data.

Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the channel ranges, output limits, and scanning order available with your DAQ device.

Note: The AO Write One Update VI uses an uninitialized shift register as local memory to remember the taskID for the group of channels when calling between VIs. Usually, this VI appears in one place on your diagram. However, if you use it in more than one place, the multiple instances of the VI share the same taskID. All calls to this VI configure or write data to the same group. If you want to use this VI in more than one place on your diagram, and want each instance to refer to a different taskID (for example, to write data with two devices at the same time), you should save a copy of this VI with a new name (for example, AO Write One Update R) and make your new VI reentrant. See the topic Reentrant Execution for information on how to make a VI reentrant.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **channels** is a list of the analog channels you want to use. If `x`, `y`, and `z` refer to channels, you

can specify a list of channels by separating the individual channels with commas (for example, x,y,z). If x refers to the first channel in a consecutive channel range and y refers to the last channel, you can specify the range by separating the first and last channels by a colon(for example, x:y). Refer to the *Channel Addressing with the AMUX-64T* section of Chapter 5, *Things You Should Know about Analog Input*, in the *LabVIEW Data Acquisition Basics Manual*, for the syntax you use for AMUX-64T channels or SCXI.

voltage data is a 2D array that contains analog data. (The AI Read One Scan, AI Single Scan, AO Write One Update, and AO Single Update VIs VIs return only a 1D array in the **voltage data** parameter.) This array is scan-ordered which means that each row contains the data of a single scan or value. If you want to index the array in a loop, you will get a scan on each iteration. Each column holds the data of a single channel. The upper array index selects the scan and the lower index selects the channel. Refer to the Chapter 3, *Basic LabVIEW Data Acquisition Concepts*, in the *LabVIEW Data Acquisition Basics Manual* for more information about the array format.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

limit settings is an array of clusters. Each array element specifies the limits for the channel or channels in the corresponding element of **channels**. If the length of the **limit settings** array is less than the number of **channels**, the values in the last element of the **limit settings** array apply to the remaining channels. The **limit settings** input defaults to an empty array, which means the limits do not change from their default for any channel. Use the configuration utility to establish the default settings. Each cluster contains the following parameters. See the AO Hardware Config VI description in this chapter for more information.

upper limit is equal to your reference voltage and is the maximum voltage the DAC can produce.

lower limit is either 0.0 V (which implies a unipolar setting) or the negative value of the upper limit (which implies a bipolar setting).

reference source.

0: Do not change the **reference source** setting (default input).
1: Internal (default setting).
2: External.

iteration can be used to optimize operation when you execute this VI in a loop. When **iteration** is 0 (default), LabVIEW calls the AO Config VI to configure the port. If **iteration** is greater than zero, LabVIEW uses the existing configuration, which improves performance. You usually wire this input to an iteration terminal.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

AO Continuous Gen VI

AO Continuous Gen

AO Waveform Gen VI

AO Waveform Gen

AO Write One Update VI

AO Write One Update

Error Handling for Analog Output Utility VIs

LabVIEW makes error handling easy with the intermediate-level Analog Output Utility VIs. Each intermediate-level VI has an **error in** input cluster and an **error out** output cluster. The clusters contain a Boolean that indicates whether an error occurred, the error code for the error, and the name of the VI that returned the error. If **error in** indicates an error, the VI returns the error information in **error out** and does not continue to run.

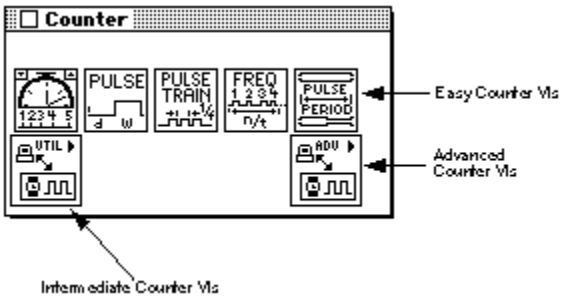
When you use any of the Analog Output Utility VIs in a While Loop, you should stop the loop if the **status** in the **error out** cluster reads `TRUE`. If you wire the error cluster to the General Error Handler VI, the VI deciphers the error information and describes the error to you.

The General Error Handler VI is in **Functions**»**Utilities** in LabVIEW. For more information on this VI, refer to the topic, General Handler VI.

Counter VIs

This section gives general information about the Counter VIs in LabVIEW.

Counter VIs can be found by choosing **Functions**»**Data Acquisition**»**Counter**. When you click on the Counter icon in the **Data Acquisition** palette, the **Counter** palette pops up. Click on an icon in the following illustration to go to a particular VI description, or click on the icon label for basic information about a class of Counter VI.



For examples of how to use the Counter VIs, open the example library by opening `examples\daq\counter.llb`.

Easy Counter VIs (Overview)

The Easy Counter VIs perform simple counting operations. You can run these VIs from the front panel or use them as subVIs in basic applications.

These VIs are stand-alone executables, which means you only need one Easy Counter VI to perform each basic counting operation. Unlike intermediate- and advanced-level VIs, Easy Counter VIs automatically alert you to errors with a dialog box that asks you to stop the execution of the VI or to ignore the error.

The Easy Counter VIs are actually composed of Intermediate Counter VIs, which are in turn composed of Advanced Counter VIs. The Easy Counter VIs provide a basic, convenient interface with only the most commonly used inputs and outputs. For more complex applications, you should use the intermediate- or advanced-level VIs for more functionality and performance.

Refer to Easy Counter VIs, for specific VI information.

Intermediate Counter VIs (Overview)

You can find the Intermediate Counter VIs in the second row of the Counter palette. The Intermediate Counter VIs are in turn built from the fundamental building block layer, called the Advanced Counter VIs. These VIs offer almost as much power as the advanced-level VIs, and they conveniently group the advanced-level VIs into a tidy, logical sequence.

Refer to Intermediate Counter VIs, for specific VI information.

Advanced Counter VIs (Overview)

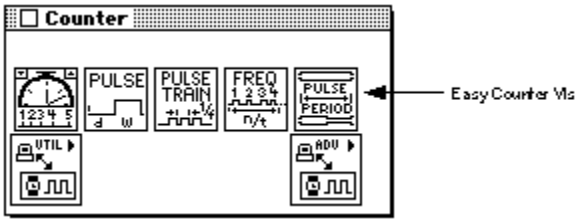
You can access the Advanced Counter palette by choosing the Advanced Counter icon from the Counter palette. These VIs are the interface to the NI-DAQ software and are the foundation of the Easy and Intermediate Counter VIs. Because all these VIs rely on the advanced-level VIs, you can refer to the topic, Advanced Counter VIs, for additional information on the inputs and outputs and how they work.

Refer to Advanced Counter VIs, for specific VI information.

Easy Counter VIs

The Easy Counter VIs perform simple counting operations. You can run these VIs from the front panel or use them as subVIs in basic applications. For general information about this VI, see the Easy Counter VIs Overview.

You can access the Easy Counter VIs by choosing **Functions**»**Data Acquisition**»**Counter**. The Easy Counter VIs are the VIs on the top row of the **Counter** palette. Click on an icon in the following illustration to go to a particular Easy Counter VI.



Easy VIs

Count Events or Time
Generate Delayed Pulse
Generate Pulse Train
Measure Frequency
Measure Pulse Width or Period

This section describes the high-level VIs for programming counters on the MIO, TIO, and other devices with the Am9513 or DAQ-STC counter/timer chips. These VIs call the Intermediate Counter VIs to generate a single delayed TTL pulse, a finite or continuous train of pulses, and to measure the frequency, pulse width, or period of a TTL signal. These VIs do not work with Lab and 1200 Series devices, DAQCards, and other devices that have the 8253 chip. Use the intermediate-level ICTR Control VI for those devices. Refer to Intermediate Counter VIs for more information on the ICTR Control VI.

Some of these VIs use other counters in addition to the one specified. In this case, a logically adjacent counter is chosen, which is referred to as counter+1 when it is the adjacent, logically higher counter and counter -1 when it is the adjacent, logically lower counter.

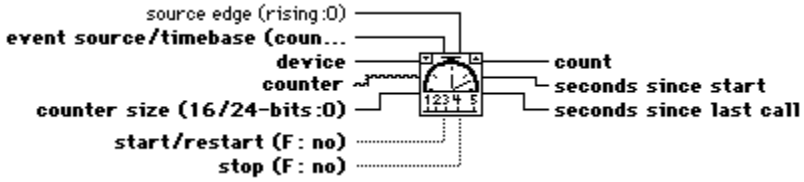
(**Macintosh and Windows**) For a device with the Am9513 chip, if the counter is 1, then counter+1 is counter 2 and counter-1 is counter 5.

See the Adjacent Counters VI described in Intermediate Counter VIs, for more information.

For examples of how to use the Easy Counter VIs, open the example library by opening `examples\daq\counter.llb`.

Count Events or Time

Configures one or two counters to count external events or elapsed time. An external event is a high or low signal transition on the specified SOURCE pin of the counter.



To count events, set the event source/timebase to 0.0 and connect the signal you want to count to the

SOURCE pin of the counter. To count time, set this control to the timebase frequency you want to use.

I32 **event source/timebase** (Hz) is set to the frequency of the internal signal whose cycles are counted, or it is set to # 0.0 (default) to count the signal on the SOURCE pin of the counter.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **counter** identifies the counter(s) the VI configures. The counter(s) specified by counter list must belong to the group identified by **taskID**. If counter list is empty, the VI programs all counters in the group. See the CTR Group Config VI for the counter list element syntax and valid counter numbers.

I32 **counter size** is set to 0 (default) to use the specified 16-bit Am9513 counter or 24-bit DAQ-STC counter. It is set to 1 to use the two Am9513 counters as a 32-bit counter. Leave this input set to 0 for a DAQ-STC counter.

I32 **start/restart** is set TRUE to configure and start the counter(s).

I32 **stop** is set TRUE to stop the counter(s).

I32 **source edge** is the edge of the counter clock signal on which it decrements.

0: Count on low to high transition.

1: Count on high to low transition.

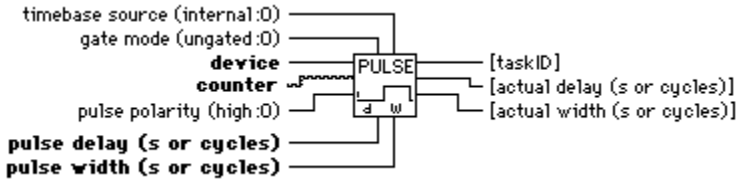
I32 **count** is the value of the counter at the time it is read. If there are two Am9513 counters assigned to the **taskID**, the value of the higher order counter is multiplied by 10000 hex, shifting it to the left 16 bits. The higher order counter is then added to the value of the lower counter. The count will be incorrect if the **taskID** is for two concatenated DAQ-STC counters.

I32 **seconds since start** is the amount of time that has elapsed since counting began. This indicator is not used for event counting.

I32 **seconds till overflow** is the amount the time remaining until the counter reaches TC and overflows. This indicator is not used for event counting.

Generate Delayed Pulse

Configures and starts a counter to generate a single pulse with the specified delay and pulse width on the counters OUT pin. A single pulse consists of a delay phase (phase 1), followed by a pulse phase (phase 2), and than a return to the phase 1 level. If an internal timebase is chosen, the VI selects the highest resolution timebase for the counter to achieve the desired characteristics. If an external timebase signal is chosen, the user indicates the delay and width as cycles of that signal. Execute the Counter Start VI with this VIs taskID to generate another pulse. You can optionally gate or trigger the pulse with a signal on the counters GATE pin.



I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **counter** identifies the counter(s) the VI configures. The counter(s) specified by counter list must belong to the group identified by **taskID**. If counter list is empty, the VI programs all counters in the group. See the CTR Group Config VI for the counter list element syntax and valid counter numbers.

I32 **pulse delay (s or cycles)** is the desired duration of the first phase of the pulse, phase 1. The unit

is seconds if **timebase source** is `0` (internal) and cycles if **timebase source** is `1` (external). If **pulse delay** is `0.0` and **timebase source** is `0`, the VI selects a minimum delay of three cycles of the timebase used.

I32 **pulse width (s or cycles)** is the desired duration of the second phase of the pulse, phase 2. The unit is seconds if **timebase source** is `0` (internal) and cycles if **timebase source** is `1` (external). If **pulse width** is `0.0` and **timebase source** is `0`, the VI selects a minimum width of three cycles of the timebase used.

I32 **timebase source** is the signal that counts the counter.

I32 **gate mode** specifies how the counters GATE signal is used.

0: Ungated/software start: ignore the gate source and start when the VI is called (default).
1: Count while the gate signal is TTL high.
2: Count while the gate signal is TTL low.
3: Start counting on the rising edge of the TTL gate signal.
4: Start counting on the falling edge of the TTL gate signal.
5: Restart counting on each rising edge of the TTL gate signal.
6: Restart counting on each falling edge of the TTL gate signal.

Use **gate mode** 3 or 4 to generate one delayed pulse on the first gate edge after starting. Use **gate mode** 5 or 6 to generate a delayed pulse for each gate edge (i.e., retriggerable one-shot behavior).

I32 **pulse polarity** is the polarity of the second phase (phase 2) of the pulse.

0: High pulse: phase 1 (the delay) is a low TTL level and phase 2 is a high level (default).
1: Low pulse: phase 1 is a high TTL level and phase 2 is a low level.

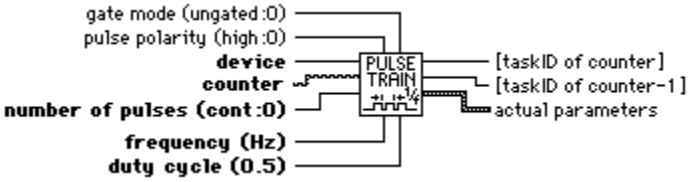
I32 **taskID** is the reference number assigned for the device and counter(s) reserved for this operation.

I32 **actual delay (s or cycles)** is the achieved delay. It may differ from the desired delay because the hardware has limited resolution and range.

I32 **actual width (s or cycles)** is the achieved pulse width. It may differ from the desired width because the hardware has limited resolution and range.

Generate Pulse Train

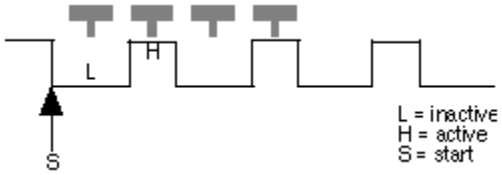
Configures the specified counter to generate a continuous pulse train on the counter's OUT pin, or to generate a finite-length pulse train using the specified counter and an adjacent counter. The signal has the prescribed frequency, duty cycle, and polarity. Each cycle of the pulse train consist of a delay phase (phase 1) followed by a pulse phase (phase 2).



This VI uses only the specified **counter** to generates a continuous pulse. For a finite-length pulse, the VI also uses counter-1 to generate a minimum-delayed pulse to gate **counter**. To generate another pulse train, execute the intermediate Counter Start VI with the **taskID**s supplied by this VI. To stop a continuous pulse train, execute the intermediate Counter Stop VI or execute this counter again to generate one, short pulse. You must externally wire counter-1's OUT pin to **counter's** GATE pin for a finite-length pulse train. You can optionally gate or trigger the start of the train with a signal on the counter-1's GATE pin.

Note: A pulse train consists of a series of delayed pulses, where phase 1 or the first phase of each pulse is the inactive state of the output (low for a high pulse) and the phase 2 of the second phase is the pulse itself. Refer to the following illustration of a high polarity pulse

train.



I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **counter** identifies the counter(s) the VI configures. The counter(s) specified by counter list must belong to the group identified by **taskID**. If counter list is empty, the VI programs all counters in the group. See the CTR Group Config VI for the counter list element syntax and valid counter numbers.

I32 **number of pulses** is the number of pulses you want in the pulse train. If the value is #0 (default), the VI generates a continuous pulse train.

I32 **frequency** (Hz) is the desired repetition rate of the pulse train.

I32 **duty cycle** is the desired ratio of the durations of phase 2 (phase two) of the pulse to the period of one cycle (1/frequency); default is 0.5. **If duty cycle** is `0.0` or `1.0`, the VI computes the closest achievable duty cycle using a minimum period of three timebase cycles. A **duty cycle** very close to `0.0` or `1.0` may not be possible.

I32 **gate mode** specifies how the signal on the counter's GATE pin is used for a continuous pulse train or how the signal on counter-1's GATE pin is used for a finite pulse train.

0: ungated/software start: ignore the gate source and start when the VI is called. (default)
1: count while the gate signal is TTL high.
2: count while the gate signal is TTL low.
3: start one pulse train on the rising edge of the TTL gate signal.
4: start one pulse train on the falling edge of the TTL gate signal.

If **number of pulses** # 0 (continuous pulse train), **gate mode** 3 or 4 generates one pulse per gate edge, which is the behavior of a retriggerable one shot. If **number of pulses** > 0 and you want to retriggered the pulse train, you must restart **counter** between the end of one train and the next edge of the gate signal.

I32 **pulse polarity** is the polarity of the pulse phase (phase 2) of each pulse in the train.

0: High pulse: phase 1 (the delay) is a low TTL level and phase 2 is a high level (default).
1: Low pulse: phase 1 is a high TTL level and phase 2 is a low level.

I32 **taskID of gate counter** is the taskID of counter-1, the one used to gate the continuous pulses.

I32 **taskID of continuous pulse counter** is the taskID of **counter**, the one used to generate the continuous pulse.

I32 **actual parameters** is a cluster of lesser parameters. These parameters may differ from the desired parameters because the hardware has limited resolution and range.

I32 **frequency** (Hz) is the achieved frequency.

I32 **duty cycle** is the achieved duty cycled.

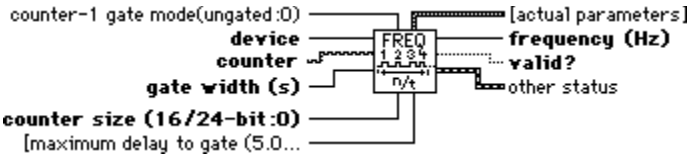
I32 **pulse delay** is the achieved minimum delay to the gating pulse.

I32 **pulse widt**h is the achieved width of the gating pulse.

Measure Frequency

Measures the frequency of a TTL signal on the specified counters SOURCE pin by counting positive edges of the signal during a specified period of time. In addition to this connection, you must wire the

counters GATE pin to the OUT pin of counter-1. This VI is useful for relatively high frequency signals, when many cycles of the signal occur during the timing period. Use the Measure Pulse Width or Period VI for relatively low frequency signals. Keep in mind that period(s) = 1/frequency (Hz).



This VI configures the specified **counter** and counter+1 (optional) as event counters to count rising edges of the signal on counter's SOURCE pin. The VI also configures counter-1 to generate a minimum-delayed pulse to gate the event counter, starts the event counter and then the gate counter, waits the expected gate period, and then reads the gate counter until its output state is low. Next the VI reads the event counter and computes the signal frequency (**number of events/actual gate pulse width**) and stops the counters. You can optionally gate or trigger the operation with a signal on counter-1's GATE pin.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **counter** identifies the counter(s) the VI configures. The counter(s) specified by counter list must belong to the group identified by **taskID**. If counter list is empty, the VI programs all counters in the group. See the CTR Group Config VI for the counter list element syntax and valid counter numbers.

I32 **gate width (s)** is the desired length of the pulse used to gate the signal. The lower the signal frequency, the longer the width must be.

I32 **counter size** is set to 0 (default) to use the specified 16-bit Am9513 counter or 24-bit DAQ-STC counter. It is set to 1 to use the two Am9513 counters as a 32-bit counter. Leave this input set to 0 for a DAQ-STC

I32 **frequency** (Hz) is the frequency of the signal, computed as **number of events/actual gate pulse width**.

I32 **valid?** is TRUE if the measurement completes without a counter overflow.

I32 **counter-1 gate** mode specifies how the counter-1s GATE signal is used.

0: Ungated/software start: ignore the gate source and start when the VI is called. (default)
1: Count while the gate signal is TTL high.
2: Count while the gate signal is TTL low.
3: Start counting on the rising edge of the TTL gate signal.
4: Start counting on the falling edge of the TTL gate signal.

I32 **maximum delay** to gate is the maximum expected delay between the time the VI is called and the start of the gating pulse. If **counter-1 gate mode**>0 and the gate does not start in this time, a timeout occurs.

I32 **actual parameters** is a cluster of lesser parameters. These parameters may differ from the desired parameters because the hardware has limited resolution and range.

I32 **number of rising edges** is the number of rising edges of the signal counted during the gate.

I32 **gate pulse (s) width** is the length of the gating pulse that is used.

I32 **other status** is a cluster of additional status information.

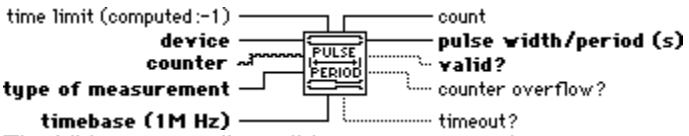
I32 **counter overflow?** is TRUE if the event counter reaches TC. Overflows do not produce an error.

I32 **timeout?** is TRUE if the time limit expires during the call. A timeout and valid measurement may occur at the same time. A timeout does not produce an error.

Measure Pulse Width or Period

Measures the pulse width (length of time a signal is high or low) or period (length of time between adjacent rising or falling edges) of a TTL signal connected to the counters GATE pin. The method used

gates an internal timebase clock with the signal being measured. This VI is useful in measuring the period or frequency (1/period) of relatively low frequency signals, when many timebase cycles occur during the gate. Use the Measure Frequency VI to measure the period or frequency of relatively high frequency signals.

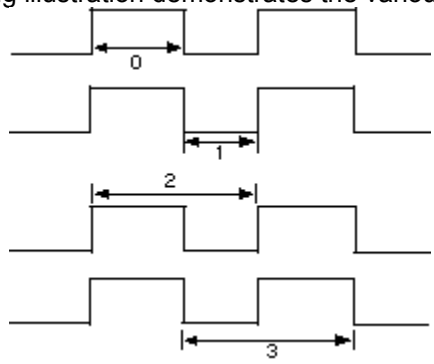


The VI iterates until a valid measurement, timeout, or counter overflow occurs. A valid measurement exists when **count** 4 without a counter overflow. If counter overflow occurs, lower the timebase. If you start a pulse width measurement during the phase you want to measure, you get an incorrect low measurement. Therefore, make sure the pulse does not occur until after the counter is started. This restriction does not apply to period measurements.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **counter** identifies the counter(s) the VI configures. The counter(s) specified by counter list must belong to the group identified by **taskID**. If counter list is empty, the VI programs all counters in the group. See the CTR Group Config VI for the counter list element syntax and valid counter numbers.

I32 **type of measurement** identifies the type of pulse width or period measurement to make. The following illustration demonstrates the various values for **type of measurement**.



0: Measure high pulse width from rising to falling edge.
1: Measure low pulse width from falling to rising edge.
2: Measure period between adjacent rising edges. (default)
3: Measure period between adjacent falling edges.

I32 **timebase** is the internal clock signal to use (default 1 MHz). If the counter overflows because **timebase** is too high, lower it until a valid reading occurs or until the lowest timebase is used and a timeout occurs.

I32 **pulse width/period (s)** is the measured pulse width or period; it equals **count/timebase** and may be valid or invalid.

I32 **valid?** is TRUE if counter has not underflowed (if count 4) or overflowed.

I32 **time limit** is the period to wait for a valid measurement. If **time limit** is `-1.0` (default), the time limit is set to five seconds or four times the range of the counter at the selected **timebase** (4*65,536)/**timebase** in seconds for the Am9513 and (4*16,777,216)/**timebase** for the DAQ-STC), whichever is larger.

I32 **count** is the value of the counter at the time it is read. For best accuracy, choose a timebase frequency that maximizes the count without overflowing it. If there are two Am9513 counters assigned to the taskID, the value of the higher order counter is multiplied by 10000 hex, shifting it to the left 16 bits.

The higher order counter is then added to the value of the lower counter. The count will be incorrect if the **taskID** is for two concatenated DAQ-STC counters.

counter overflow? is TRUE if counter reaches TC. Overflow does not produce an error.

timeout is TRUE if a valid reading is not within the prescribed or computed time limit. The **timeout** parameter does not produce an error.

Count Events or Time VI

Count Events or Time

Generate Delayed Pulse VI

Generate Delayed Pulse

Generate Pulse Train VI

Generate Pulse Train

Measure Frequency VI

Measure Frequency

Measure Pulse Width or Period VI

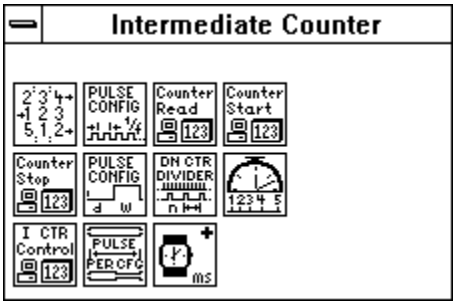
Measure Pulse Width or Period

Intermediate Counter VIs

This section describes the Intermediate Counter VIs you can use to program counters on MIO, TIO, and other devices with the Am9513 or DAQ-STC counter chips. These VIs call the Advanced Counter VIs to configure the counters for common operations and to start, read, and stop the counters. You can configure these VIs to generate single pulses and continuous pulse trains, to count events or elapsed time, to divide-down a signal, and to measure pulse width or period. The Easy Counter VIs call these Intermediate VIs for several pulse generation, counting, and measurement operations.

(Macintosh and Windows) This topic also describes the ICTR Control VI that you use with Lab and 1200 Series devices, which contain the 8253 counter chip.

You can access the Intermediate Counter VIs by choosing **Functions**»**Data Acquisition**»**Counter**. The Intermediate Counter VIs are the VIs on the second row of the **Counter** palette. Click on an icon in the following illustration to go to a particular Intermediate Counter VI.



Click here for information on Error Handling for Counter VIs. For general information about these VIs, see the Intermediate Counter VIs Overview.

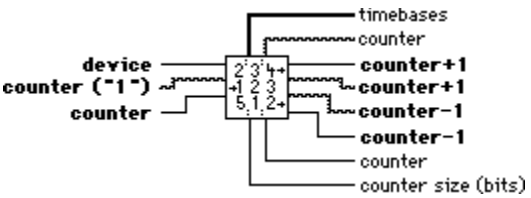
For examples of how to use the Intermediate Counter VIs, open the example library by opening `examples\daq\counter.llb`.

Intermediate VIs

Adjacent Counters
Continuous Pulse Generator Config
Counter Read
Counter Start
Counter Stop
Delayed Pulse Generator Config
Down Counter or Divider Config
Event or Time Counter Config
ICTR Control
Pulse Width or Period Meas Config
Wait+ (ms)

Adjacent Counters

This VI identifies the counters logically adjacent to a specified counter of an MIO or TIO device. It also returns the counter size (number of bits) and the timebases.



Devices with the Am9513 chip have one or two sets of five, 16-bit counters (1-5, 6-10) that can be connected in a circular fashion. For example, the next higher counter to counter 1 (called counter+1) is 2 and the next lower one (called counter-1) is 5.

Devices with the DAQ-STC chip have two 24-bit counters (0 and 1). The table below identifies the adjacent counters.

Adjacent Counters for Counter Chips

Device Type	Next Lower Counter	Counter	Next Higher Counter
Am9513 (Macintosh and Windows)	5	1	2
	1	2	3
	2	3	4
	3	4	5
	4	5	1
	10	6	7
	6	7	8
	7	8	9
	8	9	10
	9	10	6
DAQ-STC (Windows and Sun)	1	0	1
	0	1	0

Many counter operations require two or more counters, and in some cases, you can use a logically adjacent counter without external wiring. The Adjacent Counters VI lets you build VIs that have only one counter parameter and identifies any other counters it uses.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **counter** identifies the counter(s) the VI configures. The counter(s) specified by counter list must belong to the group identified by **taskID**. If counter list is empty, the VI programs all counters in the group. See the CTR Group Config VI for the counter list element syntax and valid counter numbers.

I32 **counter** is the number of the counter to use for the operation. Use this input when you want to specify the counter by number rather than by string. This input is used if the string input is empty.

I32 **counter+1** is the number of the next higher counter.

I32 **counter+1** is the number of the next higher counter returned as a string.

I32 **counter-1** is the number of the next lower counter returned as a string.

I32 **counter-1** is the number of the next lower counter.

I32 **timebases** is an array of the counters internal timebase frequencies. The Am9513 counters have timebases of 1 MHz, 100 kHz, 10 kHz, 1 kHz, and 100 Hz; the DAQ-STC has timebases of 20 MHz and 100 kHz.

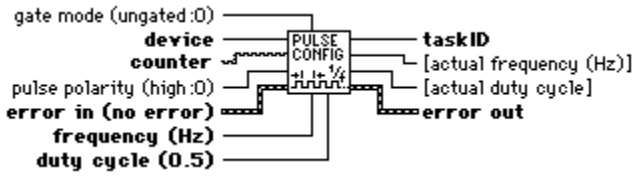
I32 **counter** is the number of the counter, returned as a string, to use for the operation. This output is convenient when you specify the counter by ASCII string but need the numeric form as well. This input is used if the string input is empty.

I32 **counter** is the number of the counter to use for the operation. This output is convenient when you specify the counter by number but need the form as well. This input is used if the string input is empty.

I32 **counter size** is the size of the counter in bits. The Am9513 has 16-bit counters, while the DAQ-STC has 24-bit counters. Use this information when you need to determine the range of the counter.

Continuous Pulse Generator Config

Configures a counter to generate a continuous TTL pulse train on its OUT pin.



The signal is created by repeatedly decrementing the counter twice, first for the delay to the pulse (phase 1), then for the pulse itself (phase two). The VI selects the highest resolution timebase to achieve the desired characteristics. You can optionally gate or trigger the operation with a signal on the counter's GATE pin. Call the Counter Start VI to start the pulse train or to enable it to be gated.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **counter** identifies the counter(s) the VI configures. The counter(s) specified by counter list must belong to the group identified by **taskID**. If counter list is empty, the VI programs all counters in the group. See the CTR Group Config VI for the counter list element syntax and valid counter numbers.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **frequency** (Hz) is the desired repetition rate of the continuous pulse train.

I32 **duty cycle** is the desired ratio of the duration of the pulse phase (phaseÊ2) of the pulse to the period of one cycle (1/frequency). The default of 0.5 generates a square wave. If **duty cycle** is 0.0, the VI computes the closest achievable duty cycle using a minimum pulse phase (phase 2) of three timebase cycles. If duty cycle is 1.0, the VI computes the achievable duty cycle using a minimum delay phase (phase 1) of three timebase cycles. A duty cycle very close to 0.0 or 1.0 may not be possible.

I32 **gate mode** specifies how the counters GATE signal is used.

0: Ungated/software start: ignore the gate source and start when the Counter Start VI is called (default).
1: Count while the gate signal is TTL high after the Counter Start VI is called.
2: Count while the gate signal is TTL low after the Counter Start VI is called.
3: Start counting on the rising edge of the TTL gate signal after the Counter Start VI is called.
4: Start counting on the falling edge of the TTL gate signal after the Counter Start VI is called.

If **gate mode** is 3 or 4, the counter generates a single pulse on each edge.

I32 **pulse polarity** is the polarity of phase 2 of each cycle in the pulse train.
0: High pulse: the delay (phase 1) is a low TTL level and phase 2 is a high level (default).
1: Low pulse: the delay (phase 1) is a high TTL level and phase 2 is a low level.

I32 **taskID out** has the same value as **taskID.**

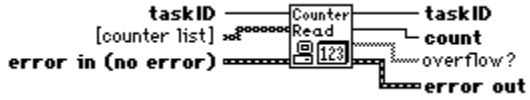
I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

I32 **actual frequency** (Hz) is the achieved frequency. It may differ from the desired frequency because the hardware has limited resolution and range.

I32 **actual duty cycle** is the achieved duty cycled. It may differ from the desired duty cycle because the hardware has limited resolution and range.

Counter Read

Reads the counter or counters identified by **taskID**.



The VI is designed to read one counter or two concatenated counters of an Am9513 counter chip or to read one counter of a DAQ-STC counter chip.

I32 **taskID** identifies the group and the I/O operation.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **counter list** is the set of counters to read. Use this array only to read a subset of counters identified by taskID; otherwise, leave it empty.

I32 **taskID out** has the same value as **taskID**.

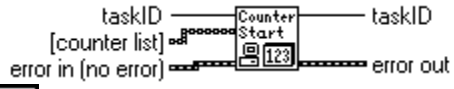
I32 **count** is the value of the counter at the time it is read. The **count** is incorrect if the taskID is for two cascaded DAQ-STC counters.
If there are two counters assigned to the **taskID**, the value of the higher order counter is shifted to 16 bits to scale it, and then it is added to the value of the lower counter.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI..

TF **overflow?** is an array of overflow indicators, one for each counter in counter list, or if the list is empty, in the group identified by **taskID**. A value of TRUE means the counter stopped when it reached TC and the reading is inaccurate.

Counter Start

Starts the counters identified by **taskID**.



I32 **taskID** identifies the group and the I/O operation.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **counter list** is the set of counters to start. Use this array only to start a subset of the counters

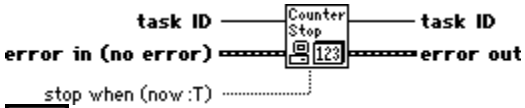
identified by taskID; otherwise, leave it empty.

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Counter Stop

Stops a count operation immediately or conditionally on an input error.



I32 **taskID** identifies the group and the I/O operation.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

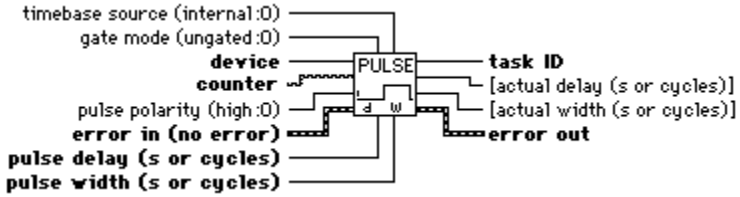
I32 **stop when** is set TRUE to stop immediately (default) and FALSE to stop on an input error. Use this input when you call the VI in a loop and want to stop on the last iteration or on an error.

I32 **taskID out** has the same value as **taskID**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Delayed Pulse Generator Config

Configures a counter to generate a single, delayed TTL pulse on its OUT pin.



The signal is created by decrementing the counter twice, first for the delay to the pulse (called phase 1), then for the pulse itself (phase 2). If an internal timebase is chosen, the VI selects the highest resolution timebase for the counter to achieve the desired characteristics. If an external timebase signal is chosen, the user designates the delay and width as cycles of that signal. You can optionally gate or trigger the operation with a signal on the counter's GATE pin. Call the Counter Start VI to start the pulse or enable it to be gated.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **counter** is the ASCII number of the counter to use for the operation.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **pulse delay (s or cycles)** is the desired duration of the first phase of the pulse, phase 1. The unit is seconds if **timebase source** is 0 (internal) and cycles if **timebase source** is 1 (external). If **pulse delay** is 0.0 and **timebase source** is internal, the VI selects a minimum delay of three cycles of the timebase used.

I32 **pulse width (s or cycles)** is the desired duration of the second phase of the pulse, phase 2. The unit is seconds if **timebase source** is 0 (internal) and cycles if **timebase source** is 1 (external). If **pulse width** is 0.0 and **timebase source** is internal, the VI selects a minimum width of three cycles of the

timebase used.

I32 **timebase source** is the signal that counts the counter.

0: Internal & calculated (default). An internal timebase is selected based on the pulse delay and width, in units of seconds.

1: Counter's SOURCE. The signal on the counter's SOURCE pin is used and the units of pulse delay and width are cycles of that signal.

I32 **gate mode** specifies how the counter's GATE signal is used.

0: Ungated/software start: ignore the gate source and start when the VI is called (default).

1: Count while the gate signal is TTL high.

2: Count while the gate signal is TTL low.

3: Start counting on the rising edge of the TTL gate signal.

4: Start counting on the falling edge of the TTL gate signal.

Use **gate mode** 3 or 4 to generate one delayed pulse on the first gate edge after starting.

I32 **pulse polarity** is the polarity of phase 2 of the pulse.

0: High pulse: the delay (phase 1) is a low TTL level and phase 2 is a high level (default).

1: Low pulse: the delay (phase 1) is a high TTL level and phase 2 is a high level.

I32 **taskID out** has the same value as **taskID in**.

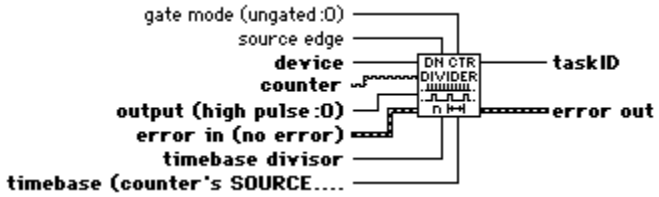
I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

I32 **actual delay (s or cycles)** is the achieved delay. This may differ from the desired delay because the hardware has limited resolution and range.

I32 **actual width (s or cycles)** is the achieved pulse width. This may differ from the desired width because the hardware has limited resolution and range.

Down Counter or Divider Config

Configures the specified counter to count down or divide a signal on the counters SOURCE pin or on an internal timebase signal using a count value called the **timebase divisor**. The result is that the signal on the counters OUT pin is equal to the frequency of the input signal/**timebase divisor**.



You can use this VI to generate finite pulse trains by enabling a continuous pulse generator until the desired number of pulses has occurred. You can also use it in place of the Continuous Pulse Generator Config VI to generate a train of strobe or trigger signals.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **counter** identifies the counter(s) the VI configures. The counter(s) specified by counter list must belong to the group identified by **taskID**. If counter list is empty, the VI programs all counters in the group. See the CTR Group Config VI for the counter list element syntax and valid counter numbers.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **timebase divisor** is the count down or divide value. For example, if the input frequency is 1000 Hz, **timebase divisor** is 10, and the output is pulsed, the frequency of the counter's OUT signal is 100 Hz. If the output is toggled, the frequency is 50 Hz.

I32 **timebase** is set to a value # 0.0 (default) to count down or divide the signal on the counter's SOURCE pin, and it is set to a valid frequency to count down or divide an internal timebase signal.

I32 **gate mode** specifies how the counter's GATE signal is used.

0: Ungated/software start: ignore the gate source and start when the VI is called (default).
1: Count while the gate signal is TTL high.
2: Count while the gate signal is TTL low.
3: Start counting on the rising edge of the TTL gate signal.
4: Start counting on the falling edge of the TTL gate signal.

I32 **source edge** is the edge of the counter clock signal on which it decrements

0: Count on low to high transition.
1: Count on high to low transition.

I32 **output (high pulse:0)** is the behavior of the output signal when **counter** reaches TC.

0: High pulse lasting one cycle of the source or timebase signal.
1: Low pulse lasting one cycle of the source or timebase signal.
2: High toggle lasting until the next TC.
3: Low toggle lasting until the next TC.

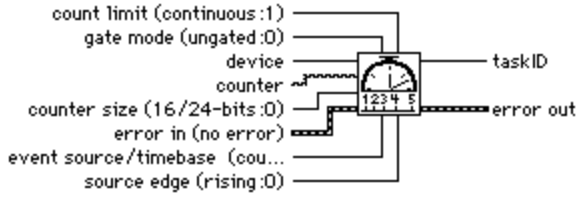
The effect of **output** modes 0 and 1 is to divide-down the source of timebase frequency by the **timebase divisor**. The effect of **output** modes 2 and 3 is to divide the frequency by twice the **timebase divisor**.

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Event or Time Counter Config

Configures one or two counters to count edges in the signal on the specified counter's SOURCE pin or the number of cycles of a specified internal timebase signal.



When the internal timebase is used, this VI works like the Tick Count (ms) function but uses a hardware counter on the DAQ device with programmable resolution. You can optionally gate or trigger the operation with a signal on the counter's GATE pin. Call the Counter Start VI to start the operation or enable it to be gated.

I32 **count limit** is the action to take when the counter reaches TC.

0: Count until TC, and set the overflow status when it is reached.
1: Count continuously and do not set the overflow status.

I32 **gate mode** specifies how the signal on the counter's GATE pin is used.

0: Ungated/software start: ignore the gate source and start when Counter Start VI is called (default).
1: Count while the gate signal is TTL high after the Counter Start VI is called.

2: Count while the gate signal is TTL low after the Counter Start VI is called.
3: Start counting on the rising edge of the TTL gate signal after Counter Start VI is called.
4: Start counting on the falling edge of the TTL gate signal after the Counter Start VI is called.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **counter** identifies the counter(s) the VI configures. The counter(s) specified by counter list must belong to the group identified by **taskID**. If counter list is empty, the VI programs all counters in the group. See the CTR Group Config VI for the counter list element syntax and valid counter numbers.

I32 **counter size** is set to 0 (default), to use only a 16-bit Am9513 counter or 24-bit DAQ-STC counter, or is set to 1 to use two Am9513 counters as a 32-bit counter. Leave this input set to 0 for a DAQ-STC counter.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **source edge** specifies whether the counter counts on the rising or falling edge of the event source/timebase.
0: Count on rising edge.
1: Count on falling edge.

I32 **event source/timebase** (Hz) is set to the frequency of the internal signal whose cycles are counted, or is set to # 0.0 (default) to count the rising edges of the signal on the counters SOURCE pin.

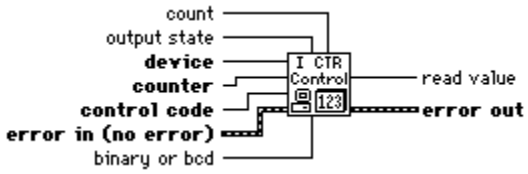
I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

ICTR Control

Controls counters the following devices that use the 8253 chip.

- (**Macintosh and Windows**) Lab and 1200 Series devices, DAQCard-500, and DAQCard 700
- (**Windows**) PC-LPM-16



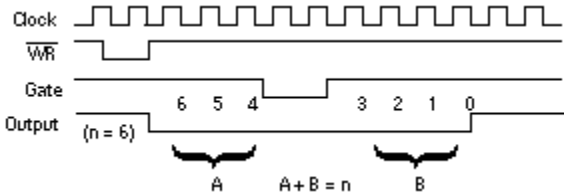
I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **counter** is the counter this VI controls, ranging from 0 through 2.

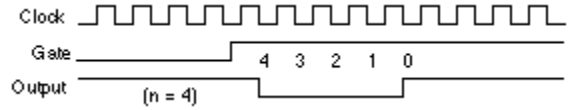
I32 **control code** determines the counters operating mode.
0: Setup mode 0--Toggle output from low to high on TC (default).
1: Setup mode 1--Programmable one-shot.
2: Setup mode 2--Rate generator.
3: Setup mode 3--Square wave rate generator.
4: Setup mode 4--Software-triggered strobe.
5: Setup mode 5--Hardware-triggered strobe.
6: Read.

7: Reset.

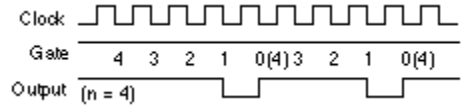
In setup mode 0, as shown in the figure below, the output becomes low after the mode set operation, and the counter begins to count down while the gate input is high. The output becomes high when counter reaches the TC (that is, when the counter decreases to 0) and stays high until you set the selected counter to a different mode.



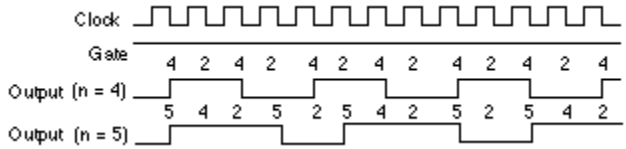
In setup mode 1, as shown in the figure below, the output becomes low on the count following the leading edge of the gate input and becomes high on TC.



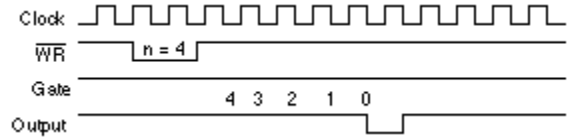
In setup mode 2, as shown in the figure below, the output becomes low for one period of the clock input. The **count** indicates the period between output pulses.



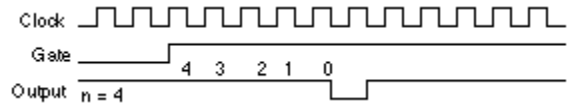
In setup mode 3, the output stays high for one-half of the **count** clock pulses and stays low for the other half. Refer to the figure below.



In setup mode 4, as in the figure below, the output is initially high, and the counter begins to count down while the gate input is high. On TC, the output becomes low for one clock pulse, then becomes high again.



Setup mode 5 is similar to mode 4, except that the gate input triggers the count to start. See the figure below for an illustration of mode 5.



See the 8253 Programmable Interval Timer data sheet in your Lab device user manual for details on these modes and their associated timing diagrams.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

I32 **count** is the period between output pulses. If c**ontrol code** is 0, 1, 4, or 5, **count** can be 0 through 65,535 in binary counter operation and 0 through 9,999 in binary-coded decimal (BCD) counter operation. If c**ontrol code** is 2 or 3, **count** can be 2 through 65,535 and 0 in binary counter operation and 2 through 9,999 and 0 in BCD counter operation.

Note: Use 0 to specify a count of 65,536 in binary counter operation and 10,000 in BCD counter operation.

I32 **output state** is only valid when **control code** is 7 (reset).

0: Low (default input).
1: High.

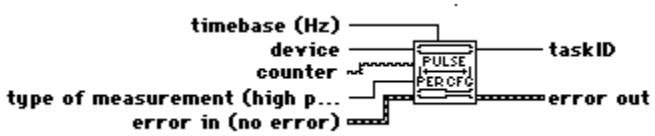
I32 **binary or BCD** controls whether the counter operates as a 16-bit binary counter or as a 4-decade BCD counter.

0: 4-decade BCD counter.
1: 16-bit binary counter.

I32 **read value**. When you set **control code** to 6 (read) **read value** returns the value the VI read from the counter.

Pulse Width or Period Meas Config

Configures the specified counter to measure the pulse width or period of a TTL signal connected to its GATE pin.



The measurement is done by counting the number of cycles of the specified timebase between the appropriate starting and ending events. To accurately measure pulse width, the pulse must occur after the counter is started. Call the Counter Start VI to start the operation. You can also use this VI to measure the frequency of low frequency signals. For more accurate measurements, use a faster timebase.

I32 **timebase** is the source of the signal that is counted to measure the pulse width or period. If **timebase** is # 0.0 (default) the counter's SOURCE signal is used; otherwise, it is the specified timebase frequency.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **counter** identifies the counter(s) the VI configures. The counter(s) specified by counter list must belong to the group identified by **taskID**. If counter list is empty, the VI programs all counters in the group. See the CTR Group Config VI for the counter list element syntax and valid counter numbers.

I32 **type of measurement** identifies the type of pulse width or period measurement to make.

0: Measure high pulse width from rising to falling edge.
1: Measure low pulse width from falling to rising edge.
2: Measure period between adjacent rising edges (default).
3: Measure period between adjacent falling edges.

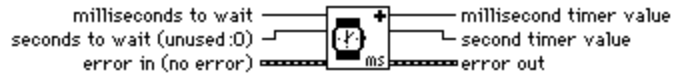
I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

taskID out has the same value as **taskID in**.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Wait+ (ms)

Calls the Wait (ms) function only if no input error exists.



This VI is useful when you want to wait between calls to I/O subVIs that use the error I/O mechanism; without it you need to use a Sequence Structure to control the execution order.

milliseconds to wait.

seconds to wait.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

millisecond timer value returns the value of the millisecond timer when the wait expires.

second timer value returns the value of the second timer when the wait expires.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Adjacent Counters VI

Adjacent Counters

Continuous Pulse Generator Config VI

Continuous Pulse Generator Config

Counter Read VI

Counter Read

Counter Start VI

Counter Start

Counter Stop VI

Counter Stop

Delayed Pulse Generator Config VI

Delayed Pulse Generator Config

Down Counter or Divider Config VI

Down Counter or Divider Config

Event or Time Counter Config VI

Event or Time Counter Config

ICTR Control VI

ICTR Control

Pulse Width or Period Meas Config VI

Pulse Width or Period Meas Config

Wait+(ms) VI

Wait+ (ms)

Error Handling for Counter VIs

LabVIEW makes error handling easy with the Intermediate Counter VIs. Each intermediate-level VI has an **error in** input cluster and an **error out** output cluster. The clusters contain a Boolean that indicates whether an error occurred, the error code for the error, and the name of the VI that returned the error. If **error in** indicates an error, the VI returns the error information in **error out** and does not continue to run.

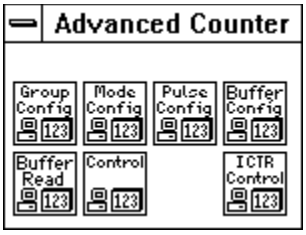
When you use any of the Intermediate Counter VIs in a While Loop, you should stop the loop if the **status** in the **error out** cluster reads `TRUE`. If you wire the error cluster to the General Error Handler VI, the VI deciphers the error information and describes the error to you.

The General Error Handler VI is in **Functions**»**Utilities** in LabVIEW. For more information on this VI, refer to the topic, General Handler VI.

Advanced Counter VIs

The Advanced Counter VIs configure and control hardware counters. You can use these VIs to generate variable duty cycle square waves, to count events, and to measure periods and frequencies. For general information about this VI, see the Advanced Counter VIs Overview.

You access the **Advanced Counter** palette by choosing **Functions»Data Acquisition»Counter»Advanced Analog Input**. The icon that you must select to access the Advanced Counter VIs is on the bottom row of the **Counter** palette. Click on an icon in the following illustration for pop-up help



For examples of how to use the Advanced Counter VIs, open the example library by opening `examples\daq\counter.llb`.

Advanced VIs

CTR Buffer Config
CTR Buffer Read
CTR Group Config
CTR Mode Config
CTR Pulse Config
CTR Control
ICTR Control

The following show the type of counter chips that your device must have to work with your version of LabVIEW.

- (**Macintosh and Windows**) Am9513, 8253, or DAQ-STC Counter Chip
- (**Sun**) DAQ-STC Counter Chip

(**Macintosh and Windows**) The ICTRControl VI works with devices that contain the 8253 counter chip.

Refer to the table below for the counter chips used with the various devices.

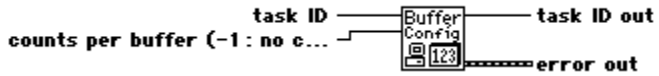
Counter Chips and Their Available DAQ Devices

Counter Chip	DAQ Device
Am9513	AT-MIO-16, AT-MIO-16D, AT-MIO-16F-5, AT-MIO-16X, AT-MIO-64F-5, PC-TIO-10, All AO-2DC Devices, EISA-A2000, NB-MIO-16, NB-MIO-16X, NB-DMA-8-G, NB-DMA2800, NB-TIO-10, NB-A2000
DAQ-STC	All E Series Devices
8253	All Lab and 1200 Series Devices,

DAQCard-500,
DAQCard-700, PC-LPM-16

CTR Buffer Config (Macintosh and Windows)

Allocates memory where LabVIEW stores counter data. The CTR Buffer Config VI also configures the specified group to perform buffered counter operations instead of the normal single point operations.



{bmc uint32c.BMP} **taskID** identifies the group and the I/O operation.

{bmc uint32c.BMP} **counts per buffer** specifies the number of counts the buffer can hold. A non-zero value configures the specified group for buffered counter operations. A value of 0 configures the group for single point counter operations. The default input is -1, which tells LabVIEW not to change the **counts per buffer** parameter. The default setting is 0.

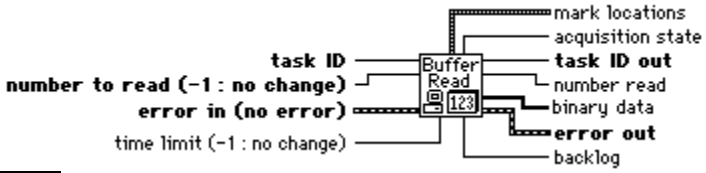
I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

CTR Buffer Read (Macintosh and Windows)

Returns data from the buffer allocated by CTR Buffer Config.



I32 **taskID** identifies the group and the I/O operation.

I32 **number to read** is the number of count values to read from the buffer. The default input is -1, which tells LabVIEW not to change **number to read**. The default setting is the CTR Buffer Config VI parameter **counts per buffer**. A value of 0 returns the status of the counter operation without reading any data.

Note: **Incremental reading from the count buffer is not supported at this time. Therefore, you must allow the buffer to fill before you read from it and then you must read all of it. Until incremental reading and circular use of the buffer are implemented, leave number to read unwired (with a value of -1) or set it to the value of counts per buffer.**

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **time limit** is the maximum length of time in seconds that this VI waits for the data. If the time expires before the data is available, the VI returns a timeout error in the **status** parameter. The default input is -1.0, which tells LabVIEW not to change the **time limit**. The default setting is 0.

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

I32 **mark locations** identifies the current location of the read and end of data marks following

execution of the VI. The marks are one-based indices, meaning the first count value within the buffer is at index 1.

I32 **read mark** identifies the next value to be read.

I32 **end of data** is the most recently acquired value.

I32 **reserved**.

I32 **reserved**.

I32 **acquisition state**

0: Running.
1: Finished with backlog.
2: Finished with no backlog.
3: Paused.
4: No acquisition.

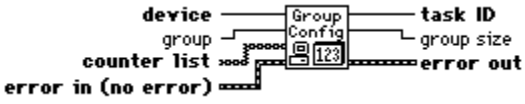
I32 **number read** is the number of counts returned in the **binary data** array.

I32 **binary data** is an array containing the counter data.

I32 **backlog** is the amount of unread data in the buffer. If the **backlog** continues to increase, the VI is not reading data fast enough, data is eventually lost, and an overwrite error occurs.

CTR Group Config

Collects one or more counters into a group. You can use counter groups containing more than one counter to start, stop, or read multiple counters simultaneously. DAQ-STC devices do not currently support multiple counter groups.



I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **counter list** is an array of strings specifying the counters the VI assigns to the group. Use the following syntax for the **counter list** strings. Use a comma to separate counter numbers from one another in a list and a colon between two counter numbers to show a range of counters. The table below gives examples of **counter list** values and their corresponding counters.

Counter List Values and Their Corresponding Counters

counter list	Counters Specified
""	removes all counter assignments in the group
1	counter 1
1, 5	counters 1 and 5
1:5	counters 1, 2, 3, 4, and 5

The table below contains valid counter numbers for devices supported by this VI.

Valid Counter Numbers for CTR Group Config Devices

Device Type	Valid Numbers
DAQ-STC Devices	0 and 1
Am9513 MIO Devices, NB-MIO-16, NB-MIO-16X	1, 2, and 5
NB-DMA-8-G, NB-DMA2800	1 through 5
PC-TIO-10, PC-AO-2DC, DAQCard-AO-2DC, NB-TIO-10	1 through 10
EISA-A2000, NB-A2000	2

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **group** ranges from 0 to 15, inclusive. This parameter defaults to a **group** 0 input and setting, which contains the maximum number of available counters. Refer to TableÊ19-3 for valid counter numbers for your device.

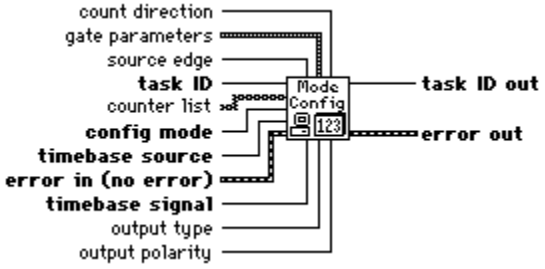
I32 **taskID** identifies the group and the I/O operation.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

I32 **group size** indicates the number of counters in the group. For example, if **counter list** contains counters 1 and 2, **group size** is 2.

CTR Mode Config

Configures one or more counters for a designated counter operation and selects the source signal, gating mode, and output behavior on terminal count (TC).



This VI does not start the counters. Use CTR Control VI with **control code** 1 (Start) to start the counters. If you are using a counter for pulse generation, you do not have to call this VI unless you want to change the gate mode or output behavior.

I32 **taskID** identifies the group and the I/O operation.

I32 **config mode** identifies the counter operation to configure for the signal selected with **timebase signal** and **timebase source**.

0: Do not change the **config mode** setting (default input).
1: Reset to the default settings.
2: Count until first TC (default setting for Am9513 devices). *LabVIEW for Sun does not currently support this option*.
3: Count continuously (default setting for DAQ-STC devices).
4: Measure period or pulse width.

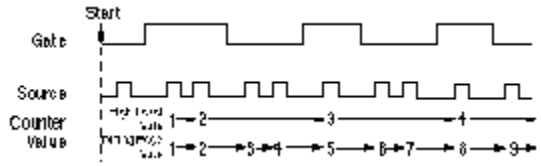
5: Generate pulse.
6: Measure pulse width (DAQ-STC devices only).
7: Measure buffered semi-periods (DAQ-STC devices only). *LabVIEW for Sun does not currently support this option.*

Modes 3, 4, and 6 can be used with or without buffered counting. Mode 7 must be used with buffered counting. With buffered counting, call the CTR Buffer Config VI before or after the CTR Mode Config VI and before the CTR Control VI to start the operation, then call the CTR Buffer Read VI to read the buffered count values. With buffered or unbuffered operations, call the CTR Control VI to read the most recently acquired, unbuffered count value.

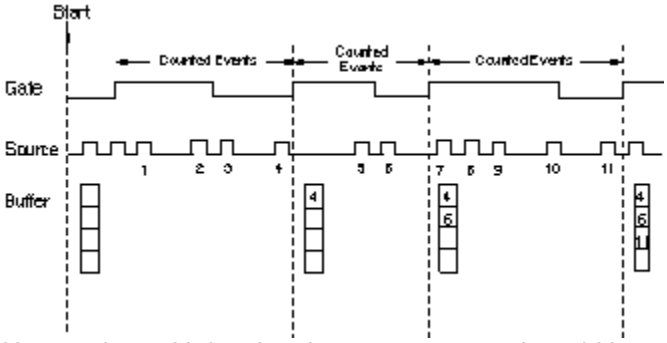
Unless otherwise stated, the following figures show timing and counter values for operations in which the **gate mode** is set to high-level or rising-edge and the **source edge** is set to rising-edge.

Use mode 1 to reset all the CTR Mode Config VI parameters to their default settings. This mode overrides any conflicting parameter settings.

Use mode 2 to count transitions of the selected signal and to stop at the first TC. The overflow status bit is set at TC. Use the CTR Control VI to read the overflow status. This mode is available only with Am9513 devices. Mode 2 counting is unbuffered. The figure below shows the count values you would read with this mode using three **gate mode** settings (gating off; high-level gating; and rising-edge gating).



Use mode 3 to count transitions of the selected signal continuously, rolling over at TC and then continuing on. The figure below shows unbuffered mode 3 counting. The following figure illustrates a buffered mode 3 operation with rising-edge gating. This buffered operation is available only with DAQ-STC devices. With buffered mode 3 operation, LabVIEW stores the current count value into the buffer on each selected edge of the source signal.

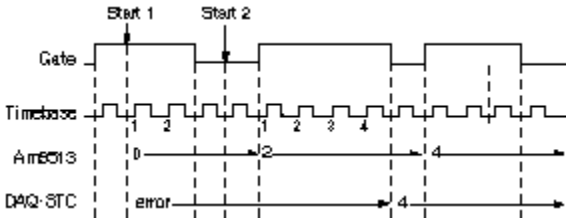


Use mode 4 with level gating to measure pulse width and with edge gating to measure the period of the selected gate signal.

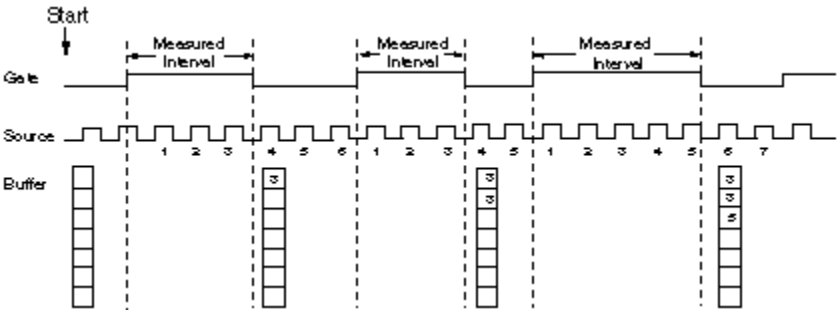
Note: For the following descriptions of pulse width measurements (modes 4, 6, and 7), a high pulse is defined simply as the high-level phase of a signal when gate mode is set to high-level gating. This definition differs from that of a high pulse using pulse generation (mode 5), which consists of a low level delay phase followed by a high level pulse phase. (Low pulses are similarly defined by switching the words high and low.)

To measure pulse width, set the **gate mode** to high or low level. The figure below shows unbuffered mode 4 pulse width measurements. You can start an Am9513 counter at any time, and it will measure pulses until you stop it. If you start it in the middle of the pulse you want to measure (for example, during a high pulse for high-level gating), LabVIEW returns a short count for that measurement. You must start a DAQ-

STC counter only when the signal is in the opposite polarity from the selected gate level (for example, a low-level phase for high-level gating). Otherwise, the VI returns error number -10890. With unbuffered counting, the DAQ-STC stops counting after one measurement. Mode 5 configures the counter for pulse generation. Use the CTR Pulse Config VI to specify the pulse you want to generate.

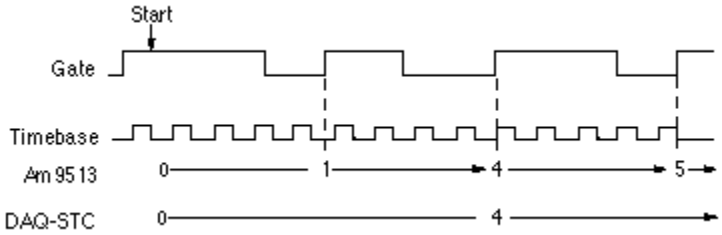


(Macintosh and Windows) The figure below shows the buffered mode 4 pulse width measurement, which is available only with DAQ-STC devices. The measured value is stored into the buffer at the end of each pulse. See mode 6 for another way to measure pulse width with a DAQ-STC device.

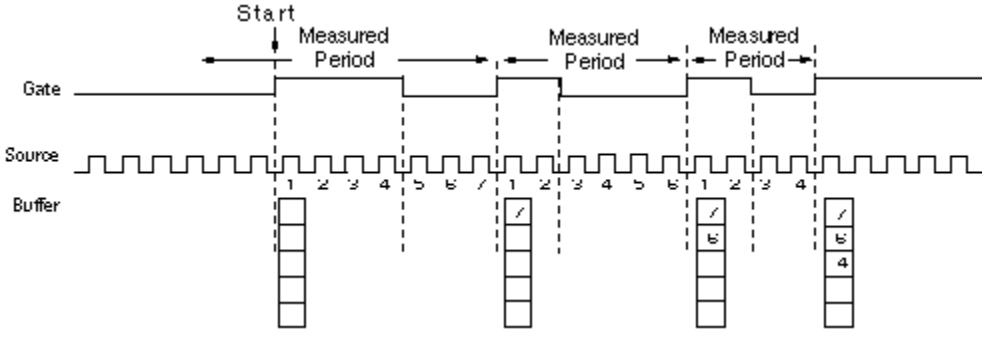


To measure period, set the **gate mode** to rising or falling edge. The figure below shows unbuffered mode 4 pulse width measurement.

You may start either an Am9513 or a DAQ-STC counter at any time. The counter begins counting at the start of the next period. The Am9513 counter measures periods continuously. With unbuffered counting, the DAQ-STC stops counting after one measurement.

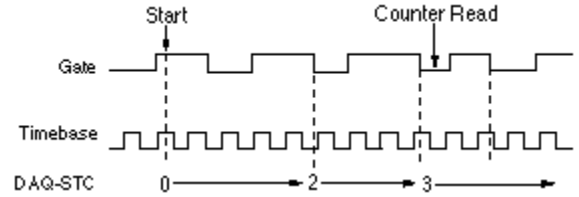


(Macintosh and Windows) The figure below shows buffered mode 4 period measurement, which is available only with DAQ-STC devices. The measured value is stored into the buffer at the end of each period.

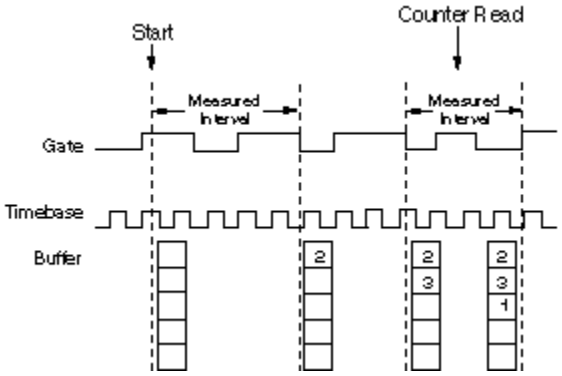


Use mode 5 to configure for pulse generation when you also need to configure the **gate mode**, **output type**, or **output polarity** to non-default values. Otherwise, avoid calling the CTR Mode Config VI and use only the CTR Pulse Config VI for pulse generation. See the CTR Pulse Config VI more additional information about this operation.

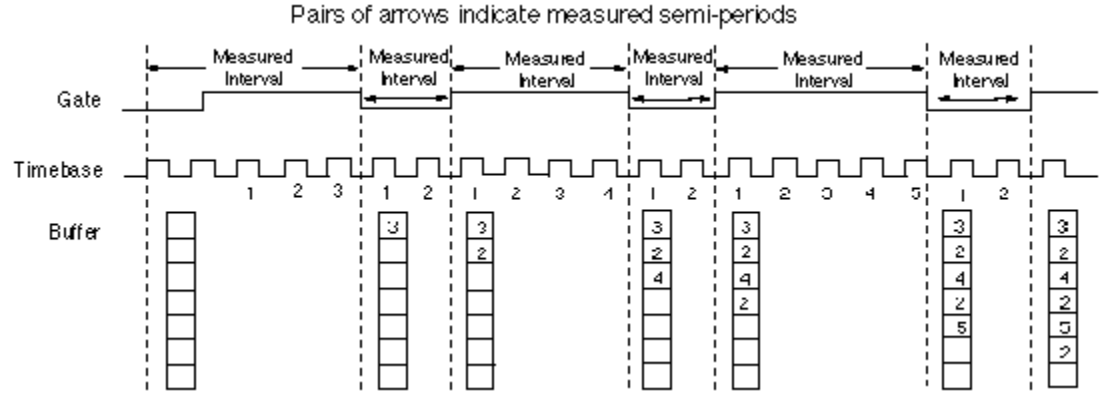
Use mode 6 with level gating to measure the pulse width of the selected signal. This mode is available only with DAQ-STC devices. Mode 6 differs from mode 4 in that the measurement of a high (low) pulse does not begin until the first falling (rising) edge of the signal after you start the counter. If you use unbuffered counting, the counter continues to measure pulses until you call the CTR Control VI to read the most recently measured value, at which time the counter stops. Unbuffered mode 6 counting is illustrated in the figure below.



(***Macintosh and Windows***) With buffered mode 6 counting, the measured value is stored into the buffer at the end of each pulse, as illustrated with the figure below. Call the CTR Buffer Read VI to read the values.



Use mode 7 to measure every phase of the selected signal using buffered counting. This mode is not available on the Sun and is available only with DAQ-STC devices. The count value is stored in the buffer on each low-to-high and high-to-low transition. Use the CTR Buffer Read VI to read the values. To measure period with this mode, sum successive pairs of signals. To measure phase, use every other value. LabVIEW ignores the value of **gate mode** with mode 7, which means that you cannot tell whether the first measurement starts at a rising or falling edge.



I32 **timebase source**.

0: Do not change the **timebase source** (default input).
1: Internal frequency in hertz (default setting).
2: Source n pin.
3: Gate n pin.
4: TC of counter n-1.
5: PFI pin (DAQ-STC devices only).
6: RTSI pin (DAQ-STC devices only).
7: ATCOut (DAQ-STC devices only). *LabVIEW for Sun does not currently support this option.*

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **timebase signal** works with **timebase source** to fully specify the timebase. If **timebase source** is `1`, **timebase signal** is the internal frequency in hertz. If **timebase source** is `2`, `3`, or `4`, **timebase signal** is the number of the counter associated with the source, gate, or TC signal. If **timebase source** is `5` or `6`, **timebase signal** is the pin number.
Click here to see the legal values and default settings for timebase signal. A value of `-1` tells LabVIEW to use the default settings. When the table says counter, it refers to the counter being configured. If there are multiple counters, LabVIEW configures each counter successively.

I32 **count direction** determines the initial counting direction (when software controls the direction) or whether a digital line controls the direction.
0: Do not change the **count direction** setting (default input).
1: Count up.
2: Count down.
3: Digital control (DAQ-STC devices only).

This parameter defaults to 1 (count up) for counting and period measurement. For pulse generation, this parameter defaults to 2 (count down). If count direction is 3, a digital line controls the direction (DIO6 for counter 0 and DIO7 for counter 1). The counter counts up when the digital line is high, and down when it is low.

I32 **gate parameters** contains the following parameters.

I32 **gate source**.
0: Do not change the **gate source** setting (default input).
1: Gate of the counter (default setting).
2: Gate of the next higher consecutive counter.
3: Gate of next lower consecutive counter.
4: TC of next lower consecutive counter.
5: PFI Pin (DAQ-STC devices only).
6: RTSI Pin (DAQ-STC devices only). *LabVIEW for Sun does not currently support this option.*
7: AI Start Trigger 1 (DAQ-STC devices only).
8: AI Stop Trigger 2 (DAQ-STC devices only).
9: ATCOut (DAQ-STC devices only). *LabVIEW for Sun does not currently support this option.*

Click here to see what is the next higher or lower consecutive counter for your counter chip.

I32 **gate signal**.
-1: Do not change the **gate signal** setting (default input).
n: PFI or RTSI pin numbers.

The default setting for **gate signal** is undefined. When **gate source** is 5 or 6, **gate signal** specifies the pin number.

I32 **gate mode** characterizes how the gate signal affects a counter.

0: Do not change the **gate mode** setting (default input).
1: Gating off (default setting).
2: High-level. The counter counts when the gate is high, but not when it is low.
3: Low-level. The counter counts when the gate is low, but not when it is high.
4: Rising-edge. The counter counts when the gate goes from low to high and continues according to the **config mode**.
5: Falling-edge. The counter counts when the gate goes from high to low and continues according to the **config mode**.

I32 **reserved**. Not currently used by LabVIEW.

I32 **source edge** specifies whether the counter counts on the rising or the falling edge of **timebase source**.

0: Do not change the **source edge** setting (default input).
1: Count on rising edge (default setting).
2: Count on falling edge.

I32 **counter list** identifies the counter(s) the VI configures. The counter(s) specified by counter list must belong to the group identified by **taskID**. If counter list is empty, the VI programs all counters in the group. See the CTR Group Config VI description in this chapter for the counter list element syntax and valid counter numbers.

I32 **output type** specifies whether a counter toggles or pulses its output when it reaches TC.

0: Do not change the **output type** setting (default input).
1: Toggled (default setting).
2: Pulsed.

I32 **output polarity** indirectly selects the initial state of a counter output line.

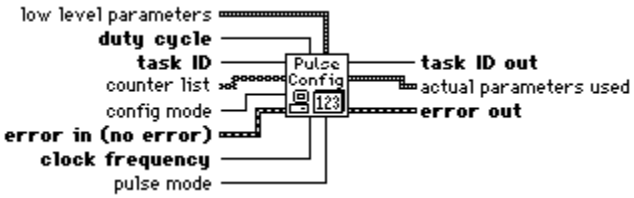
0: Do not change the **output polarity** setting (default input).
1: PositiveÐinitial state low (default setting).
2: NegativeÐinitial state high.

I32 **taskID** identifies the group and the I/O operation.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

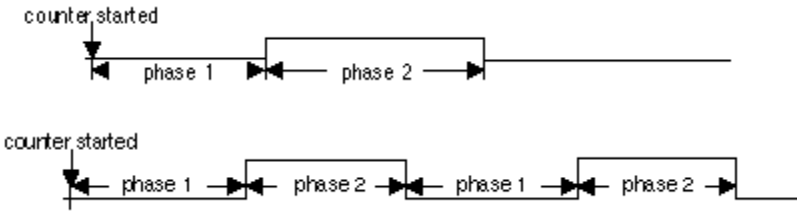
CTR Pulse Config

Specifies the parameters for pulse generation. This VI configures the counters but does not start them. Use the CTR Control VI with control code 1 (Start) to produce the pulse.



Use this VI to specify the characteristics of your pulses. You can also use the CTR Mode Config VI to set your desired gate modes, output polarity, and output type. Use the CTR Pulse Config VI to specify **timebase source** and **timebase signal** for pulse generation, because LabVIEW ignores these values specified in the CTR Mode Config VI.

I32 **duty cycle** is equal to counter **phase 2** divided by the sum of **phase 1** and **phase 2**. The figures below illustrate how LabVIEW uses phase one and phase two to construct a pulse. The range of **duty cycle** is greater than 0.0 and less than 1.0 (values of exactly 0.0 and exactly 1.0 are illegal).



I32 **taskID** identifies the group and the I/O operation.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **clock frequency** is expressed in cycles per second. When **clock frequency** is greater than zero, then it and **duty cycle** determine the pulse train characteristics. Set **clock frequency** to 0.0 to turn off the clock. The **clock frequency** defaults toÊ-1.0, which means LabVIEW does not use the **clock frequency** input, it uses the low-level parameters instead. The default setting for **clock frequency** is undefined.

I32 **low-level parameters** contains the following parameters.

I32 **clock period**. If **clock period** (expressed in seconds per cycle) is not zero, then it and **duty cycle** determine the pulse shape. The VI checks **clock frequency** first, then **clock period**. Set **clock period** to 0.0 to turn off the clock. The **clock period** defaults to -1.0, which means LabVIEW does not use the **clock period** input. The default setting for **clock period** is undefined.

If both **clock frequency** and **clock period** are -1.0, the **timebase source**, **timebase signal**, **period one**, and **period two** inputs must all be valid. These four inputs specify the pulse.

I32 **timebase source**.

0: Do not change the **timebase source** (default input).
1: Internal frequency in hertz (default setting).
2: Source *n*.
3: Gate *n*.
4: TC of counter *n*.
5: PFI Pin (DAQ-STC devices only).
6: RTSI Pin (DAQ-STC devices only). *LabVIEW for Sun does not currently support this option.*

I32 **timebase signal** works with **timebase source** to fully specify the timebase. If **timebase source** is 1, **timebase signal** is the internal frequency in hertz. If **timebase source** is 2, 3, or 4, **timebase signal** is the number of the counter associated with the source, gate, or TC signal. If **timebase source** is 5 or 6, **timebase signal** is the pin number.

Table 12-4 shows the default settings for **timebase signal**. A value of -1 tells LabVIEW to use the default settings. When the table says counter, it refers to the counter being configured. If there are multiple counters, LabVIEW configures each counter successively.

I32 **phase 1** is the number of **timebase source** edges the counter counts, to produce the first phase of a pulse. The default input for **phase 1** is -1, which means LabVIEW leaves the pulse unchanged. The default setting is undefined. For Am9513 devices the minimum and maximum values for **phase 1** are 3 and 65,535. For DAQ-STC devices, the minimum and maximum values are 2 and 16,777,215. See FiguresÊ19-10 and 19-11 for more information.

I32 **pulse 2** is the number of **timebase source** edges the counter counts to produce the second phase of a pulse. For Am9513 devices the minimum and maximum values for **phase 2** are 3 and 65,535. For DAQ-STC devices, the value ranges from 2 to 16,777,215.

I32 **auto-increment count**. This parameter applies to DAQ-STC devices only; it is ignored for all other devices. The default input and value for **auto-increment count** is 0, which means LabVIEW does

not perform auto-incrementing. This value specifies how much **phase 1** increments each time it is loaded into the counter. The values range from 0 to 255. For example, on the Macintosh or in Windows, auto-incrementing is useful if you want to use a general-purpose counter to generate timing for *equivalent-time sampling* (ETS).

I32 **counter list** identifies the counter(s) the VI configures. The counter(s) specified by counter list must belong to the group identified by **taskID**. If counter list is empty, the VI programs all counters in the group. See the CTR Group Config VI description in this chapter for the counter list element syntax and valid counter numbers.

I32 **config mode**.

0: Do not change the **config mode** setting (default input).
1: Change pulse configuration immediately (default setting).
2: Change pulse configuration at end of cycle. *LabVIEW does not currently support this option.*
3: Translate only.

If config mode is 1 and this VI is called while the counter is generating a pulse, the effect on the current cycle or period is unknown.

I32 **pulse mode**.

0: Do not change the **pulse mode** setting (default input).
1: Single pulse generation.
2: Continuous pulse generation (default setting).

I32 **taskID out** has the same value as **taskID in**.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

I32 **actual parameters used** contains the actual settings used.

I32 **frequency used**

I32 **duty cycle used**

I32 **clock period used**

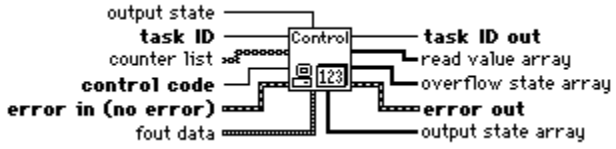
I32 **timebase signal used**

I32 **phase 1 used**

I32 **phase 2 used**

CTR Control

Controls and reads groups of counters. Control operations include starting, stopping, and setting the output state.



I32 **taskID** identifies the group and the I/O operation.

I32 **control code**.

0: Stop and reprogram (default input and setting).
1: Start.
2: Pause.
3: Resume.
4: Read.
5: Read output state.
6: Set output state.
7: Enable FOUT.

8: Disable FOUT.
9: Change count direction (DAQ-STC devices only).
10: Change parameter.

Set **control code** to 0 to stop and reprogram the counter for another start operation. If you configure the counter for counting or period measurement, all the **control code** settings are valid. If you configure the counter for pulse generation, the **control code** 4 operation is invalid.

When **control code** is 4, the VI returns the values, the output states (low or high), and the overflow states of the counters in arrays. When **control code** is 5, the VI returns only the **output state array**. The **control code** 6 parameter (set output state) is valid if you stop or pause the counter. In this case, **output state** determines the output state setting.

The **control code** 7 parameter uses the values in **fout data** to enable the FOUT output (FREQ_OUT on DAQ-STC devices). The **control code** 8 parameter disables the FOUT output (FREQ_OUT on DAQ-STC devices). The **control code** 9 parameter changes the counter direction. The **control code** 10 parameter changes the parameter specified in **change parameter data**.


error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.


output state.
0: Low-logic.
1: High-logic.
2: High impedance (default input and setting).


counter list identifies the counter(s) the VI configures. The counter(s) specified by counter list must belong to the group identified by **taskID**. If counter list is empty, the VI programs all counters in the group. See the CTR Group Config VI description in this chapter for the counter list element syntax and valid counter numbers.


fout data. If the **control code** parameter is 7 (Enable FOUT) this cluster describes the FOUT signal (FREQ_OUT for DAQ-STC devices). The signal starts immediately.


FOUT port is the frequency output port to use.
0: Do not change the **FOUT port** setting (default input).
1: FOUT port 1 (default setting).
2: FOUT port 2.


FOUT timebase source identifies the source of the timebase for the FOUT output.
0: Do not change the **FOUT timebase source** setting (default input).
1: Internal frequency in hertz (default setting).
2: Source of counter n (Am 9513 only). LabVIEW for Sun does not currently support this option.
3: Gate of counter n (Am 9513 only). LabVIEW for Sun does not currently support this option.


FOUT timebase signal works with **FOUT timebase source** to fully specify the timebase. If the **timebase source** is 1, the **timebase signal** is that frequency in hertz. For Am9513 devices, the valid values are 1,000,000.0, 100,000.0, 10,000.0, 1,000.0, and 100.0. For DAQ-STC devices the valid values are 10,000,000.0 and 100,000.0. If **timebase source** is 2 or 3, **timebase signal** is the counter number. The default input for **FOUT timebase signal** is -1.0, which means LabVIEW leaves the timebase signal unchanged.


FOUT divisor is the divide-down factor for clock generation. The **clock frequency** equals the frequency of **FOUT timebase** divided by **FOUT divisor**. The **FOUT divisor** can be 1 through 16.

0: Do not change the **FOUT divisor** setting (default input).
1: Default setting.

taskID out has the same value as **taskID in**.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

read value array contains counter values the VI reads. The readings appear in either the order of the counters in **counter list**, or if **counter list** is empty, the order of the counters in **group**.

overflow state array shows a value of 1 if a counter overflows. Otherwise, the **overflow state array** shows a value of 0. The values in the array appear in either the order of the counters in **counter list**, or if **counter list** is empty, the order of the counters in **group**.

output state array contains a value of 0 when a counter has a low output state, and a value of 1 when a counter has a high output state. The values in the array appear in either the order of the counters in **counter list**, or if **counter list** is empty, the order of the counters in **group**.

ICTR Control (Macintosh and Windows)

Controls counters on devices that use the 8253 chip (Lab and 1200 Series devices, PC-LPM-16, DAQCard-500, and DAQCard 700).

device is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

counter is the counter this VI controls, ranging from 0 through 2.

control code determines the counters operating mode.
0: Setup mode 0 -- Toggle output from low to high on TC (default).
1: Setup mode 1 -- Programmable one-shot.
2: Setup mode 2 -- Rate generator.
3: Setup mode 3 -- Square wave rate generator.
4: Setup mode 4 -- Software-triggered strobe.
5: Setup mode 5 -- Hardware-triggered strobe.
6: Read.
7: Reset.

In setup mode 0, as shown in the figure below, the output becomes low after the mode set operation, and the counter begins to count down while the gate input is high. The output becomes high when counter reaches the TC (that is, when the counter decreases to 0) and stays high until you set the selected counter to a different mode.

In setup mode 1, as shown in the figure below, the output becomes low on the count following the leading edge of the gate input and becomes high on TC.

In setup mode 2, as shown in the figure below, the output becomes low for one period of the clock input. The **count** indicates the period between output pulses.

In setup mode 3, the output stays high for one-half of the **count** clock pulses and stays low for the other half. Refer to the following figure.

In setup mode 4, as in the figure below, the output is initially high, and the counter begins to count down

while the gate input is high. On terminal count, the output becomes low for one clock pulse, then becomes high again.

Setup mode 5 is similar to mode 4, except that the gate input triggers the count to start. See the figure below for an illustration of mode 5.

See the 8253 Programmable Interval Timer data sheet in your Lab Series device user manual for details on these modes and their associated timing diagrams.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

count is the period between output pulses. If c**ontrol code** is 0, 1, 4, or 5, **count** can be 0 through 65,535 in binary counter operation and 0 through 9,999 in binary-coded decimal (BCD) counter operation. If c**ontrol code** is 2 or 3, **count** can be 2 through 65,535 and 0 in binary counter operation and 2 through 9,999 and 0 in BCD counter operation.

Note: Use 0 to equal 65,536 in binary counter operations and 10,000 in BCD counter operations.

output state is only valid when **control code** = 7 (reset).

0: Low (default input).
1: High.

binary or BCD controls whether the counter operates as a 16-bit binary counter or as a 4-decade BCD counter.

0: 4-decade BCD counter.
1: 16-bit binary counter.

read value. When you set **control code** to 6 (read) **read value** returns the value the VI read from the counter.

CTR Buffer Config VI

CTR Buffer Config

CTR Buffer Read VI

CTR Buffer Read

CTR Group Config VI

CTR Group Config

CTR Mode Config VI

CTR Mode Config

CTR Pulse Config VI

CTR Pulse Config

CTR Control VI

CTR Control

ICTR Control VI

ICTR Control

CTR Mode Config VI timebase signal Default Settings and Legal Values

timebase source	Device Type	Legal Values	Default
1	Am9513	1,000,000, 100,000, 10,000, 1,000, 100	1,000,000
	DAQÐSTC	20,000,000, 100,000	20,000,000
2, 3	Am9513	1 through 5 if counter is 1 through 5; 6 through 10 if counter is 6 through 10	counter
	DAQÐSTC	0, 1	counter
4	Am9513, DAQ-STC	Refer to the Next Lower Counter column of Table 12-5, Consecutive Counters for Devices.	N/A
5	DAQ-STC	0 through 9	no default
6	DAQ-STC	0 through 6	no default
7	DAQ-STC	N/A	N/A

Adjacent Counters for Counter Chips

Device Type	Next Lower Counter	Counter	Next Higher Counter
Am9513	5	1	2
	1	2	3
	2	3	4
	3	4	5
	4	5	1
	10	6	7
	6	7	8
	7	8	9

	8	9	10
	9	10	6
DAQ-STC	1	0	1
	0	1	0

Data Acquistion VI Error Codes

Note: All error codes and descriptions are also included in the configuration utility help panels.

Code	Name	Description
-10001	syntaxErr	An error was detected in the input string; the arrangement or ordering of the characters in the string is not consistent with the expected ordering.
-10002	semanticsErr	An error was detected in the input string; the syntax of the string is correct, but certain values specified in the string are inconsistent with other values specified in the string.
-10003	invalidValueErr	The value of a numeric parameter is invalid.
-10004	valueConflictErr	The value of a numeric parameter is inconsistent with another parameter, and the combination is therefore invalid.
-10005	badDeviceErr	The device parameter is invalid.
-10006	badLineErr	The line parameter is invalid.
-10007	badChanErr	A channel is out of range for the board type or input configuration, the combination of channels is not allowed, or you must reverse the scan order so that channel 0 is last.
-10008	badGroupErr	The group parameter is invalid.
-10009	badCounterErr	The counter parameter is invalid.
-10010	badCountErr	The count parameter is too small or too large for the specified counter.
-10011	badIntervalErr	The interval parameter is too small or too large for the associated counter or I/O channel.
-10012	badRangeErr	The analog input or analog output voltage range is invalid for the specified channel.
-10013	badErrorCodeErr	The driver returned an unrecognized or unlisted error code.
-10014	groupTooLargeErr	The group size is too large for the board.
-10015	badTimeLimitErr	The time limit parameter is invalid.
-10016	badReadCountErr	The read count parameter is invalid.
-10017	badReadModeErr	The read mode parameter is invalid.
-10018	badReadOffsetErr	The offset is unreachable.
-10019	badClkFrequencyErr	The frequency parameter is invalid.
-10020	badTimebaseErr	The timebase parameter is invalid.
-10021	badLimitsErr	The limits are beyond the range of the board.
-10022	badWriteCountErr	Your data array contains an incomplete update or you are trying to write past the end of the internal buffer or your output operation is continuous and the length of your array is not a multiple of one half of the internal buffer size.
-10023	badWriteModeErr	The write mode is out of range or is disallowed.
-10024	badWriteOffsetErr	The write offset plus the write mark is greater than the internal buffer size or it must be set to 0.
-10025	limitsOutOfRangeErr	The voltage limits are out of range for this

		board in the current configuration. Alternate limits were selected.
-10026	badInputBufferSpecification	The input buffer specification is invalid. This error results if, for example, you try to configure a multiple-buffer acquisition for a board that does not support multiple-buffer acquisition.
-10027	badDAQEventErr	For DAQEvents 0 and 1, general value A must be greater than 0 and less than the internal buffer size. If DMA is used for DAQEvent 1, general value A must divide the internal buffer evenly. If the TIO-10 is used for DAQEvent 4, general value A must be 1 or 2.
-10028	badFilterCutoffErr	The cutoff frequency specified is not valid for this device.
-10080	badChanErr	The gain parameter is invalid.
-10081	badPretrigCountErr	The pretrigger sample count is invalid.
-10082	badPosttrigCountErr	The posttrigger sample count is invalid.
-10083	badTrigModeErr	The trigger mode is invalid.
-10084	badTrigCountErr	The trigger count is invalid.
-10085	badTrigRangeErr	The trigger range or trigger hysteresis window is invalid.
-10086	badExtRefErr	The external reference value is invalid.
-10087	badTrigTypeErr	The trigger type parameter is invalid.
-10088	badTrigLevelErr	The trigger level parameter is invalid.
-10089	badTotalCountErr	The total count specified is inconsistent with the buffer configuration and pretrigger scan count or with the board type.
-10090	badRPGErr	The individual range, polarity, and gain settings are valid but the combination specified is not allowed for this board.
-10091	badIterationsErr	The analog output buffer iterations count is not allowed. It must be 0 (for indefinite iterations) or 1.
-10100	badPortWidthErr	The requested digital port width is not a multiple of the hardware port width.
-10200	EEPROMMreadError	Unable to read data from EEPROM.
-10201	EEPROMwriteError	Unable to write data to EEPROM.
-10240	noDriverErr	The driver interface could not locate or open the driver.
-10241	oldDriverErr	The driver is out-of-date.
-10242	functionNotFoundErr	The specified function is not located in the driver.
-10243	configFileErr	The driver could not locate or open the configuration file, or the format of the configuration file is not compatible with the currently installed driver.
-10244	deviceInitErr	The driver encountered a hardware-initialization error while attempting to configure the specified device.
-10245	osInitErr	The driver encountered an operating system error while attempting to perform an operation, or the operating system does not support an operation performed by the

		driver.
-10246	communicationsErr	The driver is unable to communicate with the specified external device.
-10247	cmosConfigErr	The CMOS configuration memory for the computer is empty or invalid, or the configuration specified does not agree with the current configuration of the computer.
-10248	dupAddressErr	The base addresses for two or more devices are the same; consequently, the driver is unable to access the specified device.
-10249	intConfigErr	The interrupt configuration is incorrect given the capabilities of the computer or device.
-10250	dupIntErr	The interrupt levels for two or more devices are the same.
-10251	dmaConfigErr	The DMA configuration is incorrect given the capabilities of the computer/DMA controller or device.
-10252	dupDMAErr	The DMA channels for two or more devices are the same.
-10253	jumperlessBoardErr	NI-DAQ was unable to find one or more jumperless boards you have configured using WDAQCONF.
-10254	DAQCardConfigErr	Cannot configure the DaqCard because: 1. The correct version of card and socket services software is not installed. 2. The card in the PCMCIA socket is not a DAQCard. 3. The base address and/or interrupt level requested are not available according to the card and socket services resource manager. Try different settings or use AutoAssign in the configuration utility.
-10340	noConnectErr	No RTSI signal/line is connected, or the specified signal and the specified line are not connected.
-10341	badConnectErr	The RTSI signal/line cannot be connected as specified.
-10342	multConnectErr	The specified RTSI signal is already being driven by a RTSI line, or the specified RTSI line is already being driven by a RTSI signal.
-10343	SCXIConfigErr	The specified SCXI configuration parameters are invalid, or the function cannot be executed given the current SCXI configuration.
-10360	DSPInitErr	The DSP driver was unable to load the kernel for its operating system.
-10370	badScanListError	The scan list is invalid. This error can result if, for example, you mix AMUX-64T channels and onboard channels, or if you scan multiplexed SCXI channels out of order.
-10400	userOwnedRsrcErr	The specified resource is owned by the user and cannot be accessed or modified by the driver.

-10401	unknownDeviceErr	The specified device is not a National Instruments product, or the driver does not support the device (for example, the driver was released before the device was supported).
-10402	deviceNotFoundErr	No device is located in the specified slot or at the specified address.
-10403	deviceSupportErr	The specified device does not support the requested action (the driver recognizes the device, but the action is inappropriate for the device).
-10404	noLineAvailErr	No line is available.
-10405	noChanAvailErr	No channel is available.
-10406	noGroupAvailErr	No group is available.
-10407	lineBusyErr	The specified line is in use.
-10408	chanBusyErr	The specified channel is in use.
-10409	groupBusyErr	The specified group is in use.
-10410	relatedLCGBusyErr	A related line, channel, or group is in use; if the driver configures the specified line, channel, or group, the configuration, data, or handshaking lines for the related line, channel, or group will be disturbed.
-10411	counterBusyErr	The specified counter is in use.
-10412	noGroupAssignErr	No group is assigned, or the specified line or channel cannot be assigned to a group.
-10413	groupAssignErr	A group is already assigned, or the specified line or channel is already assigned to a group.
-10414	reservedPinErr	Selected signal indicates a pin reserved by NI-DAQ. You cannot configure this pin yourself.
-10440	sysOwnedRsrcErr	The specified resource is owned by the driver and cannot be accessed or modified by the user.
-10441	memConfigErr	No memory is configured to support the current data transfer mode, or the configured memory does not support the current data transfer mode. (If block transfers are in use, the memory must be capable of performing block transfers.)
-10442	memDisabledErr	The specified memory is disabled or is unavailable given the current addressing mode.
-10443	memAlignmentErr	The transfer buffer is not aligned properly for the current data transfer mode. For example, the memory buffer is at an odd address, is not aligned to a 32-bit boundary, is not aligned to a 512-bit boundary, and so on. Alternatively, the driver is unable to align the buffer because the buffer is too small.
-10444	memFullErr	No more system memory is available on the heap, or no more memory is available on the device.
-10445	memLockErr	The transfer buffer cannot be locked into physical memory.

-10446	memPageErr	The transfer buffer contains a page break; system resources may require reprogramming when the page break is encountered.
-10447	memPageLockErr	The operating environment is unable to grant a page lock.
-10448	stackMemErr	The driver is unable to continue parsing a string input due to stack limitations.
-10449	cacheMemErr	A cache-related error occurred, or caching is not supported in the current mode.
-10450	physicalMemErr	A hardware error occurred in physical memory, or no memory is located at the specified address.
-10451	virtualMemErr	The driver is unable to make the transfer buffer contiguous in virtual memory and therefore cannot lock the buffer into physical memory; thus, you cannot use the buffer for DMA transfers.
-10452	noIntAvailErr	No interrupt level is available for use.
-10453	intInUseErr	The specified interrupt level is already in use by another device.
-10454	noDMACErr	No DMA controller is available in the system.
-10455	noDMAAvailErr	No DMA channel is available for use.
-10456	DMAInUseErr	The specified DMA channel is already in use by another device.
-10457	badDMAGroupErr	DMA cannot be configured for the specified group because it is too small, too large, or misaligned. Consult the user manual for the device in question to determine group ramifications with respect to DMA.
-10459	DLLInterfaceErr	The DLL could not be called due to an interface error.
-10460	interfaceInteractionErr	You have attempted to mix LabVIEW 2.2 VIs and LabVIEW 3.0 VIs. You may run an application consisting only of 2.2 VIs, then run the 2.2 Board Reset VI, before you can run any 3.0 VIs. You may run an application consisting of only 3.0 VIs, then run the 3.0 Device Reset VI, before you can run any 2.2 VIs.
-10560	invalidDSPhandleError	The DSP handle input to the VI is not a valid handle.
-10600	noSetupErr	No setup operation has been performed for the specified resources.
-10601	multSetupErr	The specified resources have already been configured by a setup operation.
-10602	noWriteErr	No output data has been written into the transfer buffer.
-10603	groupWriteErr	The output data associated with a group must be for a single channel or must be for consecutive channels.
-10604	activeWriteErr	Once data generation has started, only the transfer buffers originally written to can be updated. If DMA is active and a single transfer buffer contains interleaved channel-

		data, new data must be provided for all output channels currently using the DMA channel.
-10605	endWriteErr	No data was written to the transfer buffer because the final data block has already been loaded.
-10606	notArmedErr	The specified resource is not armed.
-10607	armedErr	The specified resource is already armed.
-10608	noTransferInProgErr	No transfer is in progress for the specified resource.
-10609	transferInProgErr	A transfer is already in progress for the specified resource.
-10610	transferPauseErr	A single output channel in a group cannot be paused if the output data for the group is interleaved.
-10611	badDirOnSomeLinesErr	Some of the lines in the specified channel are not configured for the transfer direction specified. For a write transfer, some lines were configured for input. For a read transfer, some lines were configured for output.
-10612	badLineDirErr	The specified line does not support the specified transfer direction.
-10613	badChanDirErr	The specified channel does not support the specified transfer direction.
-10614	badGroupDirErr	The specified group does not support the specified transfer direction.
-10615	masterClkErr	The clock configuration for the clock master is invalid.
-10616	slaveClkErr	The clock configuration for the clock slave is invalid.
-10617	noClkSrcErr	No source signal has been assigned to the clock resource.
-10618	badClkSrcErr	The specified source signal cannot be assigned to the clock resource.
-10619	multClkSrcErr	A source signal has already been assigned to the clock resource.
-10620	noTrigErr	No trigger signal has been assigned to the trigger resource.
-10621	badTrigErr	The specified trigger signal cannot be assigned to the trigger resource.
-10622	preTrigErr	The pretrigger mode is not supported or is not available in the current configuration, or no pretrigger source has been assigned.
-10623	postTrigErr	No posttrigger source has been assigned.
-10624	delayTrigErr	The delayed trigger mode is not supported or is not available in the current configuration, or no delay source has been assigned.
-10625	masterTrigErr	The trigger configuration for the trigger master is invalid.
-10626	slaveTrigErr	The trigger configuration for the trigger slave is invalid.
-10627	noTrigDrvErr	No signal has been assigned to the trigger resource.
-10628	multTrigDrvErr	A signal has already been assigned to the

		trigger resource.
-10629	invalidOpModeErr	The specified operating mode is invalid, or the resources have not been configured for the specified operating mode.
-10630	invalidReadErr	An attempt was made to read 0 bytes from the transfer buffer, or an attempt was made to read past the end of the transfer buffer.
-10631	noInfiniteModeErr	Continuous input or output transfers are not allowed in the current operating mode.
-10632	someInputsIgnoredErr	Certain inputs were ignored because they are not relevant in the current operating mode.
-10633	invalidRegenModeError	This board does not support the specified analog output regeneration mode.
-10680	badChanGainErr	All channels must have an identical setting for this board.
-10681	badChanRangeErr	All channels of this board must have the same range.
-10682	badChanPolarityErr	All channels of this board must have the same polarity.
-10683	badChanCouplingErr	All channels of this board must have the same coupling.
-10684	badChanInputModeErr	All channels of this board must have the same input range.
-10685	clkExceedsBrdsMaxConvRate	The clock rate selected exceeds the recommended maximum rate for this board.
-10686	scanListInvalidErr	A configuration change has invalidated the scan list.
-10687	bufferInvalidErr	A configuration change has invalidated the allocated buffer.
-10688	noTrigEnabledErr	The total number of scans and pretrigger scans implies that a trigger start is intended, but no trigger is enabled.
-10689	digitalTrigBErr	Digital trigger B is illegal for the total scans and pretrigger scans specified.
-10690	digitalTrigAandBErr	This board does not allow digital triggers A and B to be enabled at the same time.
-10691	extConvRestrictionErr	This board does not allow an external sample clock with an external scan clock, start trigger, or stop trigger.
-10692	chanClockDisabledErr	Cannot start the acquisition because the channel clock is disabled.
-10693	extScanClockError	Cannot use an external scan clock when performing a single scan of a single channel.
-10694	unsafeSamplingFreqError	The sampling frequency exceeds the safe maximum rate for the ADC, gains, and filters you are using.
-10695	DMANotAllowedErr	You must use interrupts. DMA is not allowed.
-10696	multiRateModeErr	Multi-rate scanning can not be used with AMUX-64, SCXI, or pre-triggered acquisitions.
-10697	rateNotSupportedErr	NI-DAQ was unable to convert your timebase/interval pair to match the actual hardware capabilities of the specified board.
-10698	timebaseConflictErr	You cannot use this combination of scan and

		sample clock timebases for the specified board.
-10699	polarityConflictErr	You cannot use this combination of scan and sample clock source polarities for this operation, for the specified board.
-10700	signalConflictErr	You cannot use this combination of scan and convert clock signal sources for this operation, for the specified board.
-10740	SCXITrackHoldErr	A signal has already been assigned to the SCXI track-and-hold trigger line, or a control call was inappropriate because the specified module is not configured for one-channel operation.
-10780	sc2040InputModeErr	When you have an SC2040 attached to your device, all analog input channels must be configured for differential input mode.
-10781	outputTypeMustBeVoltageErr or	The polarity of the output channel cannot be bipolar when outputting currents.
-10800	timeOutErr	The operation could not complete within the time limit.
-10801	calibrationErr	An error occurred during the calibration process.
-10802	dataNotAvailErr	The requested amount of data has not yet been acquired, or the acquisition has completed and no more data is available to read.
-10803	transferStoppedErr	The transfer has been stopped to prevent regeneration of output data.
-10804	earlyStopErr	The transfer stopped prior to reaching the end of the transfer buffer.
-10805	overRunErr	The clock source for the input transfer is faster than the maximum input-clock rate; the integrity of the data has been compromised. Alternatively, the clock source for the output transfer is faster than the maximum output-clock rate; a data point was generated more than once since the update occurred before new data was available.
-10806	noTrigFoundErr	No trigger value was found in the input transfer buffer.
-10807	earlyTrigErr	The trigger occurred before sufficient pretrigger data was acquired.
-10808	LPTCommunicationErr	An error occurred in the parallel port communication with the SCXI-1200.
-10809	gateSignalError	Attempted to start a pulse width measurement with the pulse in the active state.
-10840	softwareErr	The contents or the location of the driver file was changed between accesses to the driver.
-10841	firmwareErr	The firmware does not support the specified operation, or the firmware operation could not complete due to a data-integrity problem.
-10842	hardwareErr	The hardware is not responding to the specified operation, or the response from the

		hardware is not consistent with the functionality of the hardware.
-10843	underFlowErr	The update rate exceeds your system's capacity to supply data to the output channel.
-10844	underWriteErr	At the time of the update for the device-resident memory, insufficient data was present in the output transfer buffer to complete the update.
-10845	overFlowErr	At the time of the update clock for the input channel, the device-resident memory was unable to accept additional data–one or more data points may have been lost.
-10846	overWriteErr	New data was written into the input transfer buffer before the old data was retrieved.
-10847	dmaChainingErr	New buffer information was not available at the time of the DMA chaining interrupt; DMA transfers will terminate at the end of the currently active transfer buffer.
-10848	noDMACountAvailErr	The driver could not obtain a valid reading from the transfer-count register in the DMA controller.
-10849	openFileError	Unable to open a file.
-10850	closeFileError	Unable to close a file.
-10851	fileSeekError	Unable to seek within a file.
-10852	readFileError	Unable to read from a file.
-10853	writeFileError	Unable to write to a file.
-10854	miscFile Error	An error occurred accessing a file.
-10880	updateRateChangeError	A change to the update rate is not possible at this time because: 1. When waveform generation is in progress, you cannot change the interval timebase. 2. When you make several changes in a row, you must give each change enough time to take effect before requesting further changes.
-10881	partialTransferCompleteError	You cannot do another transfer after a successful partial transfer.
-10920	gpctrDataLossError	One or more data points may have been lost during buffered GPCTR operations due to the speed limitations of your system.

DAQ Configuration Utility Error Codes

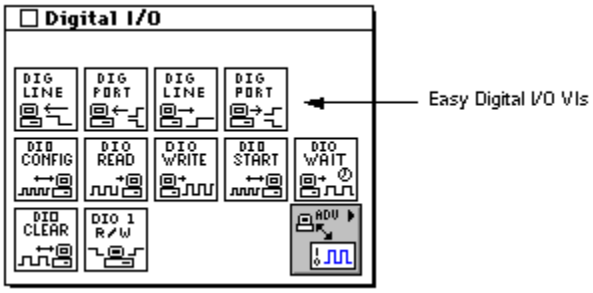
Error Code	Error Name	Description
-60	notOurBrdErr	The board in the specified slot is not an MC Series, AT Series, EISA Series, or Lab-PC board.
-61	badBrdNumErr	The board parameter is out of range.
-62	badGainErr	The gain parameter is out of range.
-63	badChanErr	The channel parameter is out of range.
-64	noSupportErr	Function cannot be executed by the specified board.
-65	badPortErr	The port parameter is out of range or the port is busy.
-66	badOutPortErr	The specified port has not been configured as an output port.
-67	noLatchModeErr	The specified port has not been configured for handshaking.
-69	badInputValErr	One or more input parameters are out of range.
-70	timeOutErr	A/D conversion did not complete or timeout period has expired.
-71	outOfRangeErr	Scaled input value is out of range.
-72	daqInProgErr	Data acquisition is in progress; therefore, call was not executed.
-75	overFlowErr	A/D FIFO memory has overflowed as a result of a DAQ or SCAN operation.
-76	overRunErr	Minimum sample interval has been exceeded as a result of a DAQ or SCAN operation.
-81	portAssignToGrp	The specified port is currently assigned to a group and can be accessed only through digital group calls until unassigned.
-197	incompatibleVISRDErr	Incorrect version of NIVISRD.386 is installed.

Easy Digital I/O VIs

The Easy Digital I/O VIs, which perform simple digital I/O operations. You can run these VIs from the front panel or use them as subVIs in basic applications.

Access the Easy Digital I/O VIs by choosing **Functions**»**Data Acquisition**»**Digital I/O**. The Easy Digital I/O VIs are the VIs on the top row of the **Digital I/O** palette. For general information about these VIs, see the Easy Digital I/O VIs (Overview).

Click on one of the icons below to go to that particular VIs description.



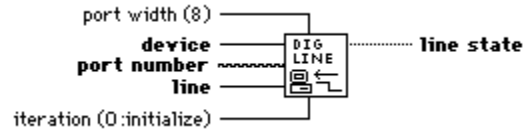
For examples of how to use the Easy Digital I/O VIs, open the example library by opening `examples\daq\digital.llb`.

Easy VIs

Read from Digital Line
Read from Digital Port
Write to Digital Line
Write to Digital Port

Read from Digital Line

Reads the logical state of a digital line on a port that you configure.



If an error occurs, a dialog box appears, giving you the option to stop the VI or continue.

Note: When you call this VI on a digital I/O port that is part of an 8255 PPI when your iteration terminal is left at 0, the 8255 PPI goes through a configuration phase, where all the ports within the same PPI chip get reset to logic low, regardless of the data direction. The data direction on other ports, however, is maintained. To avoid this effect, connect a value other than 0 to the iteration terminal once you have configured the desired ports.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **port number** is the port this VI configures. If you use a digital SCXI module, use the `SCx!MDy!0` syntax, where `x` is the chassis ID and `y` is the module number, to specify the port on a module.

I32 **line** is the individual port bit or line to be used for I/O.

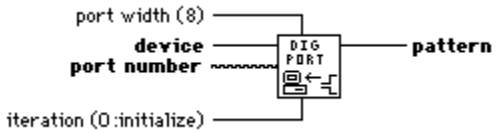
I32 **line state** is TRUE for high logic, and FALSE for low logic.

I32 **port width** is the total width or the number of lines of the port in bits. For example, you can combine two 4-bit ports into an 8-bit port on an MIO device by setting **port width** to `8`.

I32 **iteration** can be used to optimize operation when you execute this VI in a loop. When iteration is 0 (default), LabVIEW calls the DIO Port Config VI to configure the port. If iteration is greater than zero, LabVIEW uses the existing configuration, which improves performance. You usually wire this input to an iteration terminal.

Read from Digital Port

Reads a digital port that you configure.



If an error occurs, a dialog box appears, giving you the option to stop the VI or continue.

Note: When you call this VI on a digital I/O port that is part of an 8255 PPI when your iteration terminal is left at 0, the 8255 PPI goes through a configuration phase, where all the ports within the same PPI chip get reset to logic low, regardless of the data direction. The data direction on other ports, however, is maintained. To avoid this effect, connect a value other than 0 to the iteration terminal once you have configured the desired ports.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **port number** is the port this VI configures. If you use a digital SCXI module, use the `SCx!MDy!0` syntax, where `x` is the chassis ID and `y` is the module number, to specify the port on a module.

I32 **pattern** is the data the VI reads from the port.

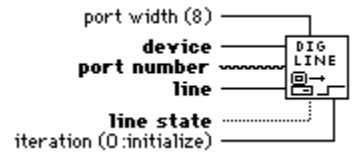
Note: If you want to display pattern in hex, octal, or binary, see the Digital Controls and Indicators section in the Numeric Controls and Indicators topic.

I32 **port width** is the total width or the number of lines of the port in bits. For example, you can combine two 4-bit ports into an 8-bit port on an MIO device by setting **port width** to `8`.

I32 **iteration** can be used to optimize operation when you execute this VI in a loop. When iteration is 0 (default), LabVIEW calls the DIO Port Config VI to configure the port. If iteration is greater than zero, LabVIEW uses the existing configuration, which improves performance. You usually wire this input to an iteration terminal.

Write to Digital Line

Sets the output logic state of a digital line to high or low on a digital port that you specify.



If an error occurs, a dialog box appears, giving you the option to stop the VI or continue.

Note: When you call this VI on a digital I/O port that is part of an 8255 PPI when your iteration terminal is left at 0, the 8255 PPI goes through a configuration phase, where all the ports within the same PPI chip get reset to logic low, regardless of the data direction. The data

direction on other ports, however, is maintained. To avoid this effect, connect a value other than 0 to the iteration terminal once you have configured the desired ports.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **port number** is the port this VI configures. If you use a digital SCXI module, use the `SCx!MDy!0` syntax, where `x` is the chassis ID and `y` is the module number, to specify the port on a module.

I32 **line** is the individual port bit or line to be used for I/O.

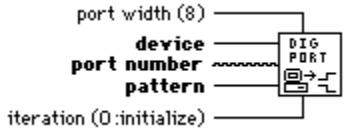
I32 **line state** is TRUE for high logic, and FALSE for low logic.

I32 **port width** is the total width or the number of lines of the port in bits. For example, you can combine two 4-bit ports into an 8-bit port on an MIO device by setting **port width** to `8`.

I32 **iteration** can be used to optimize operation when you execute this VI in a loop. When iteration is 0 (default), LabVIEW calls the DIO Port Config VI to configure the port. If iteration is greater than zero, LabVIEW uses the existing configuration, which improves performance. You usually wire this input to an iteration terminal.

Write to Digital Port

Outputs a decimal pattern to a digital port that you specify.



If an error occurs, a dialog box appears, giving you the option to stop the VI or continue.

Note: When you call this VI on a digital I/O port that is part of an 8255 PPI when your iteration terminal is left at 0, the 8255 PPI goes through a configuration phase, where all the ports within the same PPI chip get reset to logic low, regardless of the data direction. The data direction on other ports, however, is maintained. To avoid this effect, connect a value other than 0 to the iteration terminal once you have configured the desired ports.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **port number** is the port this VI configures. If you use a digital SCXI module, use the `SCx!MDy!0` syntax, where `x` is the chassis ID and `y` is the module number, to specify the port on a module.

I32 **pattern** is the data the VI writes to the port.

Note: If you want to enter pattern in hex, octal, or binary, see the Digital Controls and Indicators section.

I32 **port width** is the total width or the number of lines of the port in bits. For example, you can combine two 4-bit ports into an 8-bit port on an MIO device by setting **port width** to `8`.

I32 **iteration** can be used to optimize operation when you execute this VI in a loop. When iteration is 0 (default), LabVIEW calls the DIO Port Config VI to configure the port. If iteration is greater than zero, LabVIEW uses the existing configuration, which improves performance. You usually wire this input to an iteration terminal.

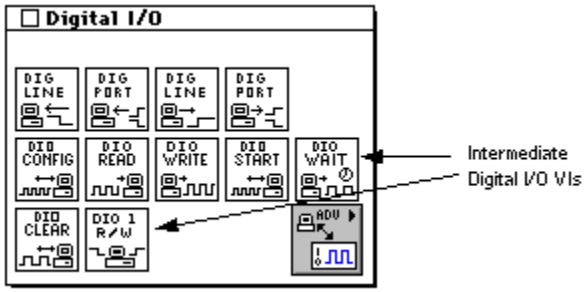
Intermediate Digital I/O VIs

Digital I/O VIs are single VI solutions to common digital problems.

Access the Intermediate Digital I/O VIs by choosing **Functions**»**Data Acquisition**»**Digital I/O**. The Intermediate Digital I/O VIs are the VIs on the second and third rows of the **Digital I/O** palette. For general information about these VIs, see the Intermediate Digital I/O VIs (Overview).

For examples of how to use the Intermediate Digital I/O VIs, open the example library by opening `examples\daq\digital.llb`.

Click on any icon in the following illustration to go to that particular VI description.



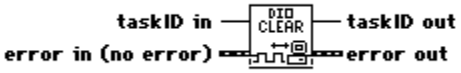
Click here for information on Error Handling for Digital I/O VIs.

Intermediate VIs

DIO Clear
DIO Config
DIO Read
DIO Single Read/Write
DIO Start
DIO Wait
DIO Write

DIO Clear (Macintosh and Windows)

Calls the Digital Group Buffer Control VI to halt a transfer and clear the group.



I32 **taskID in** identifies the group and the I/O operation.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

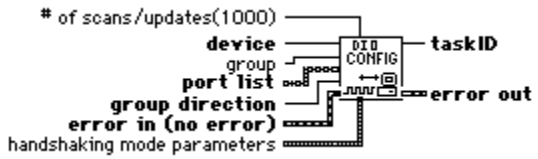
I32 **taskID out** has the same value as **taskID in.**

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI..

DIO Config (Macintosh and Windows)

The DIO Config VI calls the advanced Digital Group Config VI to assign a list of ports to the group, establish the groups direction, and produce the **taskID**. The VI then calls the Digital Mode Config VI to establish the handshake parameters, which only affect the operation of the DIO-32F devices. Finally, the

VI calls the Digital Buffer Config VI to allocate a buffer to hold the scans as they are read or the updates to be written.



Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the ports and directions available with your DAQ device.

I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **port list** is a list of digital ports, each of which is 8 lines wide. If element *n* of **port list** contains the value 1, port 1 is a member of the group. The order of **port list** determines how data that is read from or written to the group maps to the ports.

(***Macintosh and Windows***) The last port in the list controls the handshaking lines if the device is a Lab and 1200 Series, DIO-24, or DIO-96 device.

If the device is a DIO-32F, the following rules apply. If you combine ports 0 and 1 or use them separately, the REQ1 and ACK1 signals control handshaking. If you combine ports 2 and 3 or use them separately, the REQ2 and ACK2 signals control handshaking. If you combine ports 0, 1, 2, and 3, the REQ1 and ACK1 signals control handshaking. Because the two sets of handshaking signals (REQ1/ACK1 and REQ2/ACK2) are independent, you can run two groups simultaneously on a DIO-32F device.

The syntax for **port list** is `port[:port|,port]`. Items in brackets are optional. A vertical bar character (`|`) separating items means you must choose one option or the other, but not both.

The following table gives examples of the meanings of **port list** elements.

Value of port list	Meaning
port list[0] = `0`	Port 0
port list[0] = `0` **port list**[1] = 1	Ports 0 and 1
port list[0] = `0:3`	Ports 0, 1, 2, and 3
port list[0] = `0,1,3,4`	Ports 0, 1, 3, and 4

I32 **group direction** sets the direction for the group.

0: Do not change the **group direction** setting (default input).
1: Input (default setting).
2: Output.
3: Bidirectional

(Macintosh and Windows) DIO-24 and Lab devicesport 0 only
(Macintosh and Windows) DIO-96ports 0, 3, 6, 9 only
(Windows) AT-MIO-16D and AT-MIO-16DE-10 port 2 only

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **# of scans/updates** tells LabVIEW how much memory to allocate for the buffer. If the group direction is input, a single read from each port in the group is a scan. If the direction is output, a single write to all ports in the group is an update. The size of a scan or update in bytes is equal to the number of ports in the group.

(Macintosh) If you are using DMA, the buffer size cannot be greater than 2^24- 1, which is 16,777,215 bytes.

The default input for **# of scans/updates** is -1, which means LabVIEW leaves the current setting for **# of scans/updates** unchanged. The default setting for **# of scans/updates** is 1000.

I32 **group** is the number the VI assigns to the set of ports, ranging from 0 to 15. The default input and setting for **group** is 0. The default **group** 0 contains ports 0 through `n`, where `n` is the maximum number of ports minus 1. The maximum number of ports varies with device type. If the device supports handshaking, the default group is the largest group where you can do handshaking.

I32 **handshaking mode parameters**. These parameters affect only the handshaking operation of DIO-32F devices.

I32 **signal mode** specifies level or edge handshaking signals.

0: Do not change the **signal mode** setting (default input).
1: Level (default setting).
2: Edge.

I32 **edge mode** is valid only if **signal mode** specifies edge signals. **edge mode** specifies whether LabVIEW uses the leading or trailing edges of the handshaking signal.

0: Do not change the **edge mode** setting (default input).
1: Leading edges (default setting).
2: Trailing edges.

I32 **request polarity** specifies active high or low handshaking request signals.

0: Do not change the **request polarity** setting (default input).
1: Active low requests (default setting).
2: Active high requests.

I32 **acknowledge polarity** specifies active high or low handshaking acknowledge signals.

0: Do not change the **acknowledge polarity** setting (default input).
1: Active low acknowledges (default setting).
2: Active high acknowledges.

I32 **acknowledge modify mode** specifies whether LabVIEW delays issuing the acknowledge pulse or changes the duration of the pulse. You can change the duration of the pulse only when **signal mode** specifies edge handshaking signals.

0: Do not change the **acknowledge modify mode** setting (default input).
1: No modification (default setting).
2: Delay the acknowledge pulse.
3: Change the duration of the acknowledge pulse.

I32 **acknowledge modify amount** specifies the length of delay or the pulse width in nanoseconds, depending on the setting of **acknowledge modify mode**. Only the following values are valid for **acknowledge modify amount**.

0 ns
100 ns (default input and setting)

200 ns
300 ns
400 ns
500 ns
600 ns
700 ns

I32 **hardware double buffer mode** turns hardware double buffering on and off. When the direction is input and hardware double buffering is on, LabVIEW latches data into the internal buffer when the handshake signal becomes active. When the direction is output and hardware double buffering is on, data is not written to a group at the output port until the handshake signal becomes active. You typically use output hardware double buffering when a clock produces the handshake signals (that is, in a pattern generation operation).

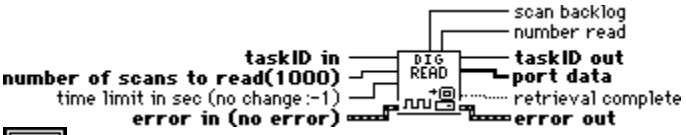
0: Do not change the **hardware double buffer mode** setting (default input).
1: Off (default setting for output groups).
2: On (default setting for input groups).

I32 **taskID** identifies the group and the I/O operation.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

DIO Read (Macintosh and Windows)

Calls the Digital Buffer Read VI to read data from the internal transfer buffer and returns the data read in **pattern**.



I32 **taskID in** identifies the group and the I/O operation.

I32 **number of scans to read** is the number of scans you want the VI to retrieve before returning. This parameter defaults to -1, which means LabVIEW leaves the **number of scans to read** setting unchanged. The default setting is equal to the number of scans in the buffer, which you set when you call the DIO Config VI. If **number of scans to read** is 0, you can check the scan backlog to determine how many scans have accumulated. If the operation is continuous, the **number of scans to read** must be a multiple of one half of the number of scans in the buffer. The VI waits until the data is available or the time limit expires.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **time limit in sec**. You express **time limit in sec** in seconds. The default input is -1.0, which tells LabVIEW to leave the **time limit in sec** setting unchanged if the transfer is handshaked and the clock rate is known. The default setting is 1 s for handshaked transfers. The resolution of the timeout clock is about 17 ms ***(Macintosh)***, 55 ms ***(Windows)***.

I32 **taskID out** has the same value as **taskID in.**

I32 **port data** is a 1D array containing the digital data that the VI obtained from the internal buffer. Each element in this array is an 8-bit unsigned integer that represents a single ports data. The total number of elements in this array equals the number of ports in the group multiplied by the number of scans to read. For example, if your group contains two ports, then elements 0 and 1 contain the data for the first scan.

I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

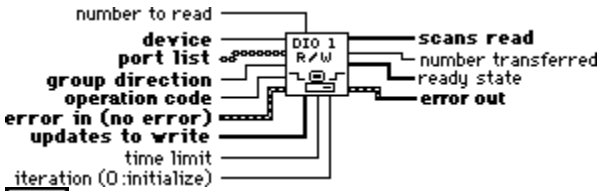
I32 **scan backlog** is the amount of data in the buffer that remains unread after this VI completes.

I32 **number read** is the number of scans returned in this call.

I32 **retrieval complete** is `TRUE` when the total number of the scans you specified in the DIO Start VI has been read.

DIO Single Read/Write

Reads or writes digital data to the ports specified in the port list. This single VI configures and transfers data. When you use this VI in a loop, wire the **iteration** counter to the **iteration** input so that port configuration takes place only once.



I32 **device** is the device number you assigned to the DAQ device during configuration (in Windows) or the number assigned by NI-DAQ during the boot sequence (on the Macintosh). For information about configuration, refer to Chapter 2, *Installing and Configuring Your Data Acquisition Hardware,* in the *LabVIEW Data Acquisition Basics Manual*.

I32 **port list** is a list of digital ports, each of which is 8 lines wide. If element *n* of **port list** contains the value 1, port 1 is a member of the group. The order of **port list** determines how data that is read from or written to the group maps to the ports.

(***Macintosh and Windows***) The last port in the list controls the handshaking lines if the device is a Lab and 1200 Series, DIO-24, or DIO-96 device.

If the device is a DIO-32F, the following rules apply. If you combine ports 0 and 1 or use them separately, the REQ1 and ACK1 signals control handshaking. If you combine ports 2 and 3 or use them separately, the REQ2 and ACK2 signals control handshaking. If you combine ports 0, 1, 2, and 3, the REQ1 and ACK1 signals control handshaking. Because the two sets of handshaking signals (REQ1/ACK1 and REQ2/ACK2) are independent, you can run two groups simultaneously on a DIO-32F device.

The syntax for **port list** is `port[:port|,port]`. Items in brackets are optional. A vertical bar character (`|`) separating items means you must choose one option or the other, but not both.

The following table gives examples of the meanings of **port list** elements.

Value of port list	**Meaning**
port list[0] = `0`	Port 0
port list[0] = `0` **port list**[1] = 1	Ports 0 and 1
port list[0] = `0:3`	Ports 0, 1, 2, and 3
port list[0] = `0,1,3,4`	Ports 0, 1, 3, and 4

I32 **group direction** sets the direction for the group.

0: Reserved
1: Input (default setting)
2: Output

3: Reserved

I32 **operation code** can have the following values.
0: Do not change the **operation code** setting (default input).
1: Read or write when data is ready for transfers (default setting).
2: Check **ready state** only.
3: Read or write data immediately, regardless of **ready state**.

If handshaking works with the configured group, the state of the handshaking lines determines whether the device is ready to transfer (read or write) each scan or update. When **operation code** is 1, the VI waits until the group is ready or a timeout occurs. If the group is ready before the timeout occurs, the VI transfers the data. When **operation code** is 3, the VI transfers the data immediately, without waiting for the group to be ready. When handshaking works with the group, **ready state** reflects the handshaking state of the group.

I32 **error in** describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

I32 **updates to write** is a 1D array containing digital data for output. Each element in this array is an 8-bit unsigned integer representing a ports data. The total number of elements in this array equals the number of ports in the group times the number of updates to write. For example, if your group contains two ports, then elements 0 and 1 contain the data for the first update.

I32 **number to read** is the number of scans you want the VI to read before returning.

I32 **time limit** is expressed in seconds. This parameter defaults to -1.0, which tells LabVIEW to leave the **time limit** setting unchanged if the transfer is handshaked or to calculate a time limit if the transfer is clocked and the clock rate is known. The default setting is 1 s for handshaked transfers. The resolution of the timeout clock is about 17 ms ***(Macintosh)*** , 55 ms ***(Windows)*** , 1 ms ***(Sun)*** .

I32 **iteration** can be used to optimize operation when you execute this VI in a loop. When **iteration** is 0 (default), LabVIEW calls the DIO Group Config VI to configure the port. If **iteration** is greater than zero, LabVIEW uses the existing configuration, which improves performance. You usually wire this input to an iteration terminal.

I32 **scans read** is a 1D array containing the digital data that the VI obtained from the internal buffer. Each element in this array is an 8-bit unsigned integer representing a ports data. The total number of elements in this array equals the number of ports in the group times the value of the **number transferred**. For example, if your group contains two ports, then elements 0 and 1 contain the data for the first scan.

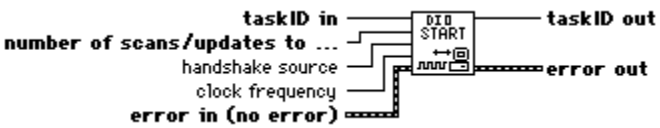
I32 **error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

I32 **number transferred** shows how many scans the VI read or updates the VI wrote before returning.

I32 **ready state**. If **ready state** is 0, the group is not ready to transfer data. If **ready state** is 1, the group is ready. If you can not use handshaking with this group, **ready state** is always 1.

DIO Start (Macintosh and Windows)

Starts a buffered digital I/O operation. This VI calls the Digital Clock Config VI to set the clock rate if the internal clock produces the handshake signals, and then starts the data transfer by calling the Digital Buffer Control VI.



taskID in identifies the group and the I/O operation.

number of scans/updates to acquire or generate is the total number of scans that you want the VI to acquire or generate before the operation completes. This VI defaults to -1, which tells LabVIEW to leave the **number of scans/updates to acquire or generate** setting unchanged. You specify the default setting (the number of scans in the buffer) by calling the DIO Config VI.

(Windows) If **number of scans/updates to acquire or generate** is 0, LabVIEW acquires or generates data continuously or until you clear the operation.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

handshake source can have the following values.

0: Do not change the **handshake source** setting (default input).
1: Internal clock. ***(Macintosh)*** This mode is not valid for the NB-DIO-32F device, unless a National Instruments DMA device is associated with the DIO device, and the two devices share a common RTSI bus.
2: I/O connector (default setting).
3: RTSI connection.

When **handshake source** is 1, the **clock frequency** control determines the clock rate.

When **handshake source** is 2, you must connect the handshake signal to the proper line on the I/O connector.

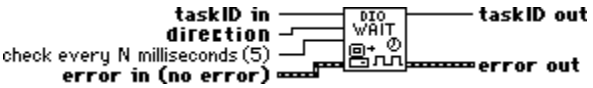
clock frequency is the rate to which you want to handshake the data. This parameter is expressed in scans/s or updates/s, depending on the input or output direction of the group. This parameter defaults to -1.0. The default setting is undefined.

taskID out has the same value as **taskID in.**

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

DIO Wait (Macintosh and Windows)

Waits until the digital buffered input or output operation completes before returning. For input, the VI detects completion when the acquisition state returned by the Digital Buffer Read VI finishes with or without backlog. For output, the VI detects completion when the **generation complete** indicator of the DIO Write VI is `TRUE`.



Refer to Appendix B, *Hardware Capabilities*, in the *LabVIEW Data Acquisition VI Reference Manual*, for the handshake modes available with your DAQ device.

taskID in identifies the group and the I/O operation.

direction tells the VI what type of operation to wait for completion.

0: Reserved.
1: Input.
2: Output.
3: Reserved.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

check every N milliseconds tells the VI how often to check the task status to see if generation is

complete. This parameter defaults to 5 ms. If this VI checks too often, it can prevent the execution of other VIs.

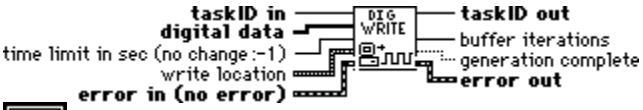
taskID out has the same value as **taskID in.**

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

DIO Write (Macintosh and Windows)

Calls the Digital Buffer Write VI to write to the internal transfer buffer.

(Macintosh) You must fill the buffer with data before you use the DIO Start VI to begin the digital output operation. You can call the DIO Write VI after the transfer begins to retrieve status information.



taskID in identifies the group and the I/O operation.

digital data is a 1D array containing digital output data. Each element in this array is an 8-bit unsigned integer that represents a single ports data. The total number of elements in this array equals the number of ports in the group multiplied by the number of updates to write. For example, if your group contains two ports, then elements 0 and 1 contain the data for the first update. If the output operation is continuous, the number of updates in this array must be a multiple of one half of the number of updates the DIO Config VI allocates in the buffer.

error in describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**. Click here to see the **error in** cluster parameters.

time limit in sec. You express **time limit in sec** in seconds. The default input is -1.0, which tells LabVIEW to leave the **time limit in sec** setting unchanged if the transfer is handshaked or to calculate a time limit if the transfer is clocked and the clock rate is known. The default setting is 1 s for handshaked transfers. The resolution of the timeout clock is about 17 ms ***(Macintosh)*** , 55 ms ***(Windows)***.

write location contains the following parameters.

write offset. The VI adds the value of **write offset** to the write mark to determine where the write begins. The default input is -1, which tells LabVIEW to leave the **write offset** setting unchanged. This parameter defaults to a setting of 0.

write mode has the following values.

0: Do not change the **write mode** setting (default input).
1: Write at the write mark plus the write offset (default setting).
2: Write at the beginning of the buffer plus the write offset.

taskID out has the same value as **taskID in**.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

buffer iterations indicates the VIs current number of complete iterations of the buffer.

generation complete is TRUE when the number of updates to generate, specified by DIO Start, has finished.

Read from Digital Line VI

Read from Digital Line

Read from Digital Line VI

Read from Digital Line

Read from Digital Port VI

Read from Digital Port

Write to Digital Line VI

Write to Digital Line

Write to Digital Port VI

Write to Digital Port

DIO Clear VI (Macintosh and Windows)

DIO Clear

DIO Config VI (Macintosh and Windows)

DIO Config

DIO Read VI (Macintosh and Windows)

DIO Read

DIO Single Read/Write VI

DIO Single Read/Write

DIO Start VI (Macintosh and Windows)

DIO Start

DIO Wait VI (Macintosh and Windows)

DIO Wait

DIO Write VI (Macintosh and Windows)

DIO Write

Error Handling

LabVIEW makes error handling easy with the Intermediate Digital I/O VIs. Each intermediate-level VI has an **error in** input cluster and an **error out** output cluster. The clusters contain a Boolean that indicates whether an error occurred, the error code for the error, and the name of the VI that returned the error. If **error in** indicates an error, the VI returns the error information in **error out** and does not continue to run.

Note: The DIO Clear VI is an exception to this rulethis VI always clears the acquisition regardless of whether error in indicates an error.

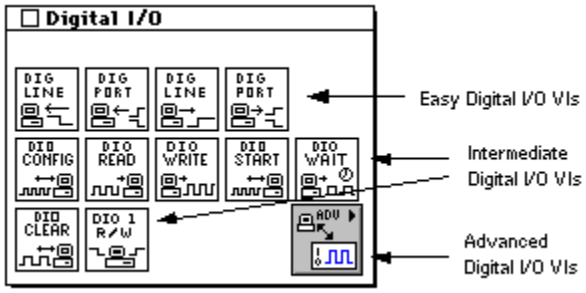
When you use any of the Intermediate Digital I/O VIs in a While Loop, you should stop the loop if the **status** in the **error out** cluster reads TRUE. If you wire the error cluster to the General Error Handler VI, the VI deciphers the error information and describes the error to you.

The General Error Handler VI is in **Functions**»**Time & Dialog** in LabVIEW. For more information on this VI, refer to the topic, General Handler VI.

Digital I/O VIs

This section gives general information about Digital I/O VIs in LabVIEW. The Digital I/O VIs can be found by choosing **Functions**»**Data Acquisition**»**Digital I/O**. When you click on the Digital I/O icon in the **Data Acquisition** palette, the **Digital I/O** palette pops up.

Click on an icon in the following illustration to go to that particular Digital I/O VI. Click on a label in the illustration to get a brief overview of the class of Digital I/O VI.



For examples of how to use the Digital I/O VIs, see the examples in `examples\daq\digital.llb`.

Easy Digital I/O VIs (Overview)

The Easy Digital I/O VIs perform simple digital I/O operations. You can run these VIs from the front panel or use them as subVIs in basic applications.

These VIs are stand-alone executables, which means you only need one Easy Digital I/O VI to perform each basic analog input operation. Unlike intermediate- and advanced-level VIs, Easy Digital I/O VIs automatically alert you to errors with a dialog box that asks you to stop the execution of the VI or to ignore the error.

The Easy Digital I/O VIs are actually composed of Intermediate Digital I/O VIs, which are in turn composed of Advanced Digital I/O VIs. The Easy Digital I/O VIs provide a basic, convenient interface with only the most commonly used inputs and outputs. For more complex applications, you should use the intermediate- or advanced-level VIs for more functionality and performance.

Click here to go to the Easy Digital I/O VIs.

Intermediate Digital I/O VIs (Overview)

Digital I/O VIs are single VI solutions to common digital problems. For example, the DIO Single Read/Write VI is a single VI solution for non-buffered reads and writes to the ports in your group. The DIO Single Read/Write VI works with any device with digital ports.

You combine the other VIs–DIO Config, DIO Start, DIO Read, DIO Write, DIO Wait, and DIO Clear–to build more demanding applications using buffered digital reads and writes. Your device must support handshaking to use this group of VIs, with the exception of the DIO Single Read/Write VI.

All the Intermediate Digital I/O VIs are built from the fundamental building block layer, the advanced-level VIs. Because all the Intermediate Digital I/O VIs rely on the advanced layer, refer to the topic Advanced Digital I/O VIs, for additional information on the inputs and outputs and how they work.

Click here to go to the Intermediate Digital I/O VIs for specific VI information.

Advanced Digital I/O VIs (Overview)

You can access the **Advanced Digital I/O** palette by choosing the Advanced Digital I/O icon from the **Digital I/O** palette. These VIs are the interface to the NI-DAQ software and are the foundation of the Easy, Utility, and Intermediate Digital I/O VIs. Because all these VIs rely on the advanced-level VIs, you can refer to Advanced Digital I/O VIs, for additional information on the inputs and outputs and how they work.

Click here to go to the Advanced Digital I/O VIs for specific VI information.

Read from Digital Line VI

Read from Digital Line

Read from Digital Port VI

Read from Digital Port

DIO Wait VI (Macintosh and Windows)

DIO Wait