

Learn 10 Functions in NI-DAQmx and Handle 80 Percent of Your Data Acquisition Applications

Publish Date: May 20, 2016

Table of Contents

1. NI-DAQmx: Reduce Development Time and Improve Performance
2. DAQ Assistant
3. NI-DAQmx Create Virtual Channel
4. NI-DAQmx Trigger
5. NI-DAQmx Timing
6. NI-DAQmx Start Task
7. NI-DAQmx Read
8. NI-DAQmx Write
9. NI-DAQmx Wait Until Done
10. NI-DAQmx Clear Task
11. NI-DAQmx Properties
12. Conclusion

(<http://zone.ni.com/devzone/cda/tut/p/id/5434>) The National Instruments Getting Started with NI-DAQmx Series is aimed at helping you learn NI-DAQmx programming fundamentals. Through video and text tutorials, this series will take you from verifying your device's operation in Measurement & Automation Explorer (MAX) to programming data acquisition applications using LabVIEW. It is intended for both the beginner who wants to learn how to use the DAQ Assistant, as well as the experienced user who wishes to take advantage of advanced NI-DAQmx functionality.

This document is part of the
**Getting Started with
NI-DAQmx Series**

1. NI-DAQmx: Reduce Development Time and Improve Performance

Since the release of NI-DAQmx, users of National Instruments data acquisition (DAQ) hardware have been taking full advantage of its many features designed to both save development time and improve the performance of their data acquisition applications.

One feature that saves a considerable amount of development time is the NI-DAQmx Application Programming Interface (API), which is the same across both device functionality and device families. This means that all of the functionality of a multifunction device is programmed with the same set of functions (analog input, analog output, digital I/O, and counters). Furthermore, both a digital I/O device and an analog output device are programmed using this same set of functions. In LabVIEW, this is possible because of polymorphism. A polymorphic VI accepts multiple data types for one or more input and/or output terminals. The NI-DAQmx API is also consistent across all of its applicable programming environments. You need to learn how to use only a single set of functions to be able to program most National Instruments data acquisition hardware in multiple programming environments.

Another feature of NI-DAQmx that improves your development experience is DAQ Assistant. This tool helps you create your applications without programming through a graphical interface for configuring both simple and complex data acquisition tasks. Moreover, synchronization, a process that is usually difficult to implement because trigger and/or clock signals must be manually routed, is effortless with NI-DAQmx, which automatically performs signal routing between the different functional areas of a single device and between multiple devices.

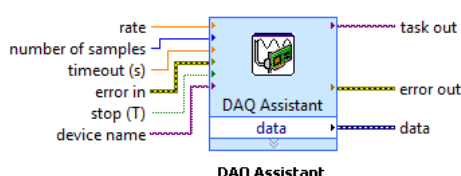
The data acquisition applications you build using NI-DAQmx benefit from an architecture designed to maximize performance. This begins with an efficient state model that eliminates unnecessary reconfiguration. With this overhead removed, both configuration and acquisition are optimized. In addition, you can achieve single-point I/O rates of greater than 50 kS/s. This level of performance is possible because of memory-mapped registers.

Another significant feature of the NI-DAQmx architecture is Measurement Multithreading (<http://zone.ni.com/devzone/cda/tut/p/id/4525>). Because NI-DAQmx is multithreaded, multiple data acquisition operations can occur simultaneously, significantly improving the performance of your applications that contain multiple operations. It also greatly simplifies programming such applications.

To begin taking advantage of these benefits, you only need to learn a few functions. In fact, 10 NI-DAQmx functions provide the functionality to solve 80% of data acquisition applications. These functions are described in detail to help you understand both their functionality and the types of applications in which they are used.

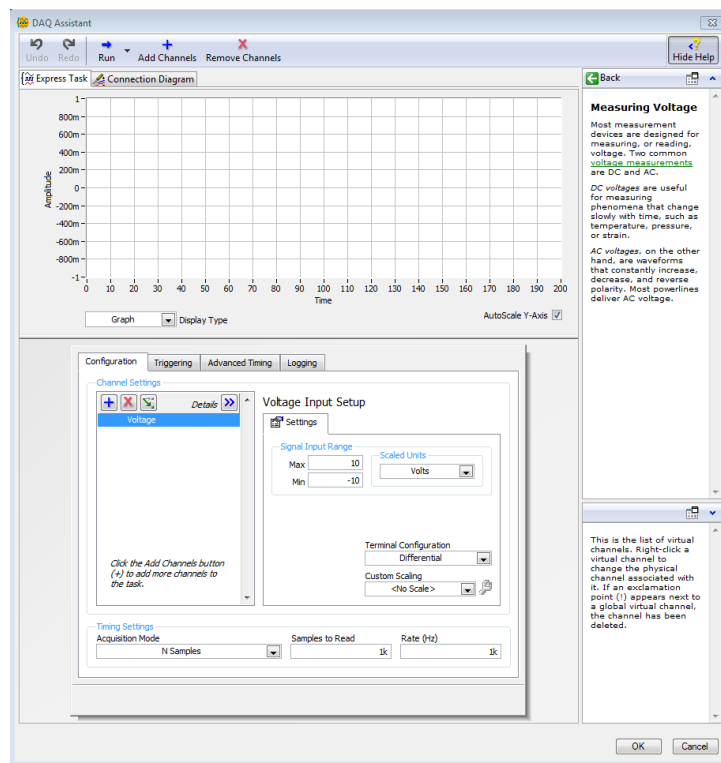
NOTE: The examples referenced throughout this document in the locations referenced by this KnowledgeBase article (<http://digital.ni.com/public.nsf/allkb/E3BAF6FC4017960B8625755A00525D37?OpenDocument>).

2. DAQ Assistant



**Tools»Create/Edit DAQmx Tasks [CVI]
Project»Add New Item»DAQmx Task [.NET]**

DAQ Assistant is a graphical interface for interactively creating, editing, and running NI-DAQmx virtual channels and tasks. An NI-DAQmx virtual channel consists of a physical channel on a DAQ device and the configuration information for this physical channel, such as input range and custom scaling. An NI-DAQmx task is a collection of virtual channels, timing and triggering information, and other properties regarding the acquisition or generation. In the following figure, DAQ Assistant is configured to perform a finite strain measurement.



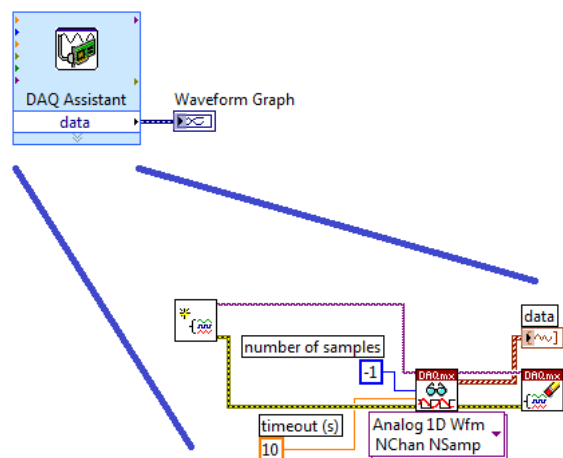
The following documents describe the use of DAQ Assistant in LabVIEW, LabWindows/CVI, and .NET:

NI-DAQmx Express VI Tutorial (<http://zone.ni.com/devzone/cda/tut/p/id/2744>)

Using the DAQ Assistant in LabWindows/CVI (<http://zone.ni.com/devzone/cda/tut/p/id/4650>)

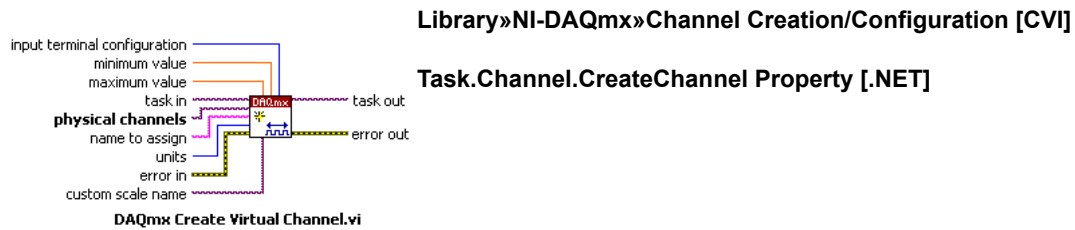
Using the DAQ Assistant in Measurement Studio (<http://zone.ni.com/devzone/cda/tut/p/id/4652>)

DAQ Assistant can also generate code to configure and/or perform the specified acquisition or generation. This procedure in LabVIEW is described in the *DAQ Assistant Help* and in the Using DAQ Assistant to Automatically Generate LabVIEW Code (<http://zone.ni.com/devzone/cda/tut/p/id/4656>) document. The following figure displays an instance of DAQ Assistant and the resulting automatically generated configuration and example LabVIEW code.



Certain data acquisition applications require more flexibility and/or performance than the DAQ Assistant provides. These applications require the simple, yet powerful, NI-DAQmx functions described below.

3. NI-DAQmx Create Virtual Channel



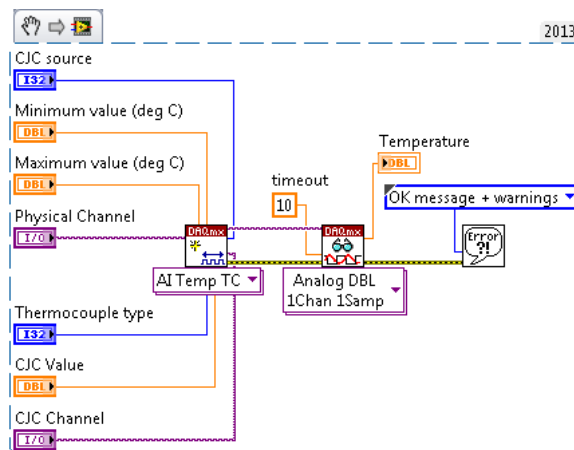
The NI-DAQmx Create Virtual Channel function creates a virtual channel and adds it to a task. It can also be used to create multiple virtual channels and add all of them to a task. When a task is not specified, the function creates a task. The NI-DAQmx Create Virtual Channel function has numerous instances. These instances correspond to the specific type of measurement or generation the virtual channel(s) perform.

Creating a Channel in LabVIEW

The following figure shows four examples of different instances of the NI-DAQmx Create Virtual Channel VI.



The inputs to the NI-DAQmx Create Virtual Channel function differ for each instance of the function. However, certain inputs are common to most, if not all, of the function's instances. For example, an input is required to specify the **physical channels** (analog input and analog output), **lines** (digital), or **counter** that the virtual channel(s) will use. Additionally, analog input, analog output, and counter operations use **minimum value** and **maximum value** inputs to configure and optimize the measurements and generations based on the minimum and maximum expected values of the signals. Furthermore, a **custom scale** can be applied to many types of virtual channels. In the following LabVIEW block diagram, the NI-DAQmx Create Virtual Channel VI is used to create a thermocouple virtual channel.



Creating a Channel in .NET

Most of the classes in the NI-DAQmx .NET library cannot be instantiated directly. These classes are used as subobjects of the **Task** class.

These classes contain properties that are specific to a particular type of channel. For example, properties such as **Count** are only applicable to counters and are only available on the **CICChannel** and **COChannel** classes. You can associate the following types of channels with a Task in the NI-DAQmx .NET class library:

Analog input channels—**AICChannel** class

Analog output channels—**AOChannel** class

Digital input channels—**DICChannel** class

Digital output channels—**DOChannel** class

Counter input channels—**CICChannel** class

Counter output channels—**COChannel** class

The **Task** class has a channel collection property for each of six types of channels— AICchannels, AOChannels, DICchannels, DOChannels, CICchannels, and COChannels. To create a channel, you can use one of the many create channel methods on the channel collections.

After instantiating a new Task object, creating and assigning an AICChannel object is done by calling the appropriate member function in the Channel class. The following snippet creates a simple analog input voltage channel:

```
analogInTask = new Task();

AIChannel myChannel;

myChannel = analogInTask.AIChannels.CreateVoltageChannel(

    "dev1/ai1", //The physical name of the channel

    "myChannel", //The name to associate with this channel

    AITerminalConfiguration.Differential, //Differential wiring

    -10, //-10v minimum

    10, //10v maximum

    AIVoltageUnits.Volts //Use volts
);
```

Creating a Task in C/C++

Use the DAQmxCreateTask() function to create a task and DAQmxCreateAIVoltageChan() to create an analog input voltage channel. For information on creating channels for different measurement types, see the NI-DAQmx C Reference Help. Below is a snippet of how to use the two functions:

```
TaskHandle taskHandle=0;
char chan[256] = "Dev1/ai0";
float64 min = -10, max = 10;

DAQmxCreateTask("",&taskHandle);


DAQmxCreateAIVoltageChan(taskHandle ,chan ,"", DAQmx_Val_Cfg_Default, min, max, DAQmx_Val_Volts, NULL);
```

Examples to Examine:

LabVIEW	Digital - Finite Output.vi
	Thermocouple - Continuous Input.vi
CVI	Write Dig Port
	Cont Thrmcpl Samples-Int Clk
.NET	WriteDigPort
	ContAcqThermocoupleSamples_IntClk


4. NI-DAQmx Trigger

task/channels in



task out

error in



error out

DAQmx Trigger.vi

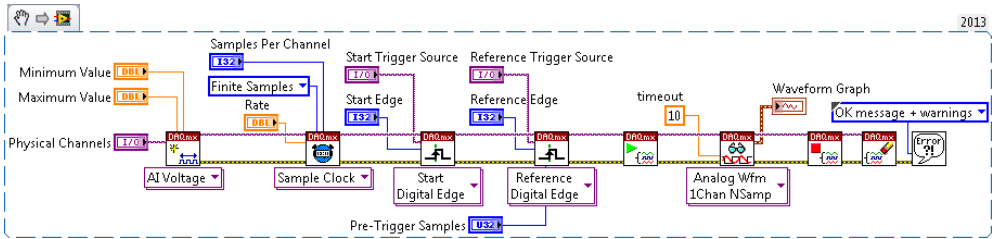
Library»NI-DAQmx»Triggering [CVI]

Task.Triggers Property [.NET]

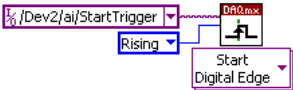
The NI-DAQmx Trigger function configures a trigger to perform a specific action. The most commonly used actions are a **start trigger** and a **reference trigger**. A start trigger initiates an acquisition or generation. A reference trigger establishes the location in a set of acquired samples where pretrigger data ends and posttrigger data begins. Both of these triggers can be configured to occur on a digital edge, an analog edge, or when an analog signal enters or leaves a window.

Creating a Trigger in LabVIEW

In the following LabVIEW block diagram, both a start trigger and a reference trigger are configured, using the NI-DAQmx Trigger VI, to occur on digital edges for an analog input operation.



Many data acquisition applications require synchronization of different functional areas of a single device (e.g. analog output and counters). Others require multiple devices to be synchronized. To achieve this synchronization, trigger signals must be routed between the different functional areas of a single device and between multiple devices. NI-DAQmx automatically



Creating a Trigger in .NET

After creating a Task object and a Channel, you can add a trigger to the task by calling the methods within the Task.Triggers collection. The following code snippet creates a Digital Edge Start Trigger:

```
analogInTask = new Task();

DigitalEdgeStartTriggerEdge triggerEdge = DigitalEdgeStartTriggerEdge.Rising;
analogInTask.AIChannels.CreateVoltageChannel( ... );
analogInTask.Triggers.StartTrigger.ConfigureDigitalEdgeTrigger("PFI0", triggerEdge);
```

Based on the parameters sent to the ConfigureDigitalEdgeTrigger function the device will look to in an internal or external line for a rising or falling digital edge before it begins to acquire data. The snippet above configures the device to look on PFI 0 for a rising digital edge trigger.

Creating a Trigger in C/C++

Use the DAQmxCfgDigEdgeStartTrig() function to create a digital edge start trigger. For information on creating different triggers, see the NI-DAQmx C Reference Help. Below is a snippet that shows how to use the function for a digital start trigger coming in on PFI0 and looking for a rising edge on that line.

```
char      startTrigger[256] = "Dev1/PFI0";
int       startEdge = 0; // rising edge
DAQmxCreateTask("", &taskHandle));

DAQmxCreateAIVoltageChan(taskHandle, chan, "", DAQmx_Val_Cfg_Default, min, max, DAQmx_Val_Volts, NULL);

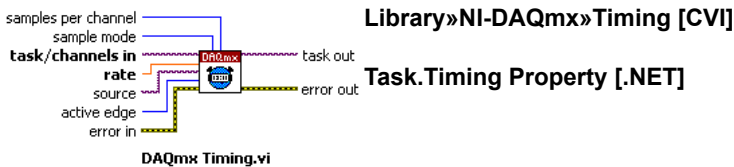
DAQmxCfgDigEdgeStartTrig(taskHandle, startTrigger, startEdge);
```

The Timing and Synchronization Features of NI-DAQmx (<http://zone.ni.com/devzone/cda/tut/p/id/4322>) document contains additional information concerning the use of the NI-DAQmx Trigger function to perform synchronization with NI-DAQmx.

Examples to Examine:

LabVIEW	Voltage - Finite Input.vi
	Voltage - Continuous Input.vi
CVI	Acq-Int Clk-Dig Start&Ref
	Cont Acq-Int Clk-Anlg Start
.NET	AcqVoltageSamples_IntClkDigStartAndRef
	ContAcqVoltageSamples_IntClkAnalogStart

5. NI-DAQmx Timing

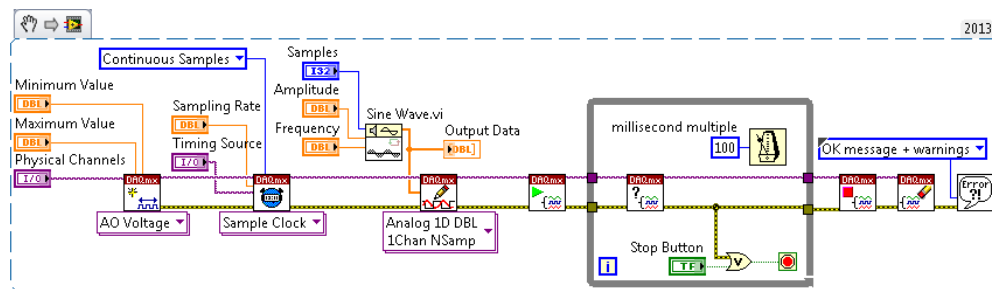


The NI-DAQmx Timing function configures the timing for hardware-timed data acquisition operations. This includes specifying whether the operation will be continuous or finite, selecting the number of samples to acquire or generate for finite operations, and creating a buffer when needed.

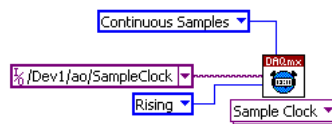
For operations that require sample timing (analog input, analog output, and counter), the **Sample Clock** instance of the NI-DAQmx Timing function sets both the source of the sample clock, which could be an internal or external source, and its rate. The sample clock controls the rate at which samples are acquired or generated. Each clock pulse initiates the acquisition or generation of one sample for each virtual channel included in the task.

Configuring Task Timing in LabVIEW

The following LabVIEW block diagram demonstrates the use of the Sample Clock instance of the NI-DAQmx Timing VI to configure a continuous analog output generation with an external sample clock.

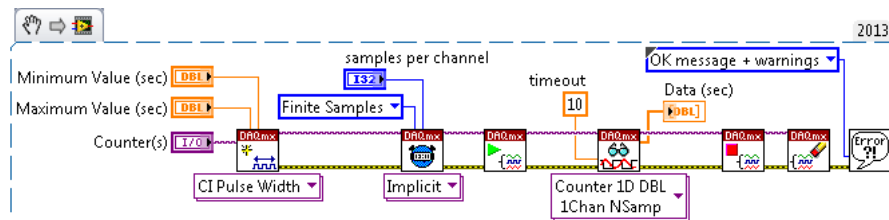


To achieve synchronization in data acquisition applications, just as trigger signals must be routed between the different functional areas of a single device and between multiple devices, timing signals must also be routed in the same manner. NI-DAQmx also automatically performs this routing. All valid timing signals are available as the **source** input to the NI-DAQmx Timing function. For example, in the following DAQmx Timing VI, the analog output sample clock signal of the device is available as the source of the sample clock for the analog input channels, without any explicit routing being performed.



The Timing and Synchronization Features of NI-DAQmx (<http://zone.ni.com/devzone/cda/tut/p/id/4322>) document contains additional information concerning the use of the NI-DAQmx Timing function to perform synchronization with NI-DAQmx.

Most counter operations do not require sample timing because the signal being measured provides the timing. The **Implicit** instance of the NI-DAQmx Timing function should be used for these applications. In the following LabVIEW block diagram, the Implicit instance of the NI-DAQmx Timing VI is used to configure a buffered pulse width acquisition to be finite with a specified number of samples.



Configuring Task Timing in .NET

To set the source of the sample clock, the rate of the sample clock, and the number of samples to acquire or generate use the Channel.Timing.ConfigureSampleClock() function. The following code snippet creates a continuous acquisition based on an external sample clock connected to PFI 0 on the device.

```
analogInTask = new Task();
analogInTask.AIChannels.CreateVoltageChannel( ... );
analogInTask.Timing.ConfigureSampleClock(
    "/Dev1/PFI0", // external clock source line or use "" for internal clock
    10000, // expected rate of external clock or actual rate of internal clock
    SampleClockActiveEdge.Rising, // acquire on rising or falling edge of ticks
    SampleQuantityMode.ContinuousSamples, // continuous or finite samples
    1000 // number of finite samples to acquire or used for buffer size if continuous
);
```

Configuring Task Timing in C/C++

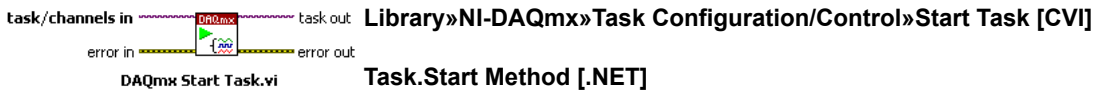
Use the DAQmxCfgSampClkTiming function to configure task timing in C or C++. The following code snippet shows its use for an internal clock running at 10000 hz.

```
char    clockSource[256] = "";
float64 rate = 10000;
DAQmxCfgSampClkTiming(taskHandle, clockSource, rate, DAQmx_Val_Rising, DAQmx_Val_ContSamps, 1000);
```

Examples to Examine:

LabVIEW	Voltage - Continuous Output.vi
	Counter - Read Pulse Width and Frequency (Finite).vi
CVI	Cont Gen Volt Wfm-Ext Clk
	Buff Semi-Period-Finite
.NET	ContGenVoltageWfm_ExtClk
	MeasBuffered_SemiPeriodFinite

6. NI-DAQmx Start Task



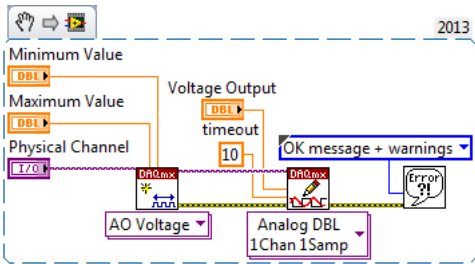
As mentioned in the Introduction, the state model used by NI-DAQmx eliminates unnecessary reconfiguring to obtain a high level of efficiency and maximize performance. This state model consists of five states in which a task can reside. Detailed information concerning each of these states is available in the **NI-DAQmx Help** under **NI-DAQmx Key Concepts»Channels and Tasks in NI-DAQmx»Tasks in NI-DAQmx»Task State Model**.

The NI-DAQmx Start Task function explicitly transitions a task to the running state. In the running state, the task performs the specified acquisition or generation. A task will be implicitly transitioned to the running state and automatically started if the NI-DAQmx Start Task function is not used when the NI-DAQmx Read function executes. This implicit transition also occurs if the NI-DAQmx Start Task function is not used and the NI-DAQmx Write function executes with its **auto start** input specified accordingly.

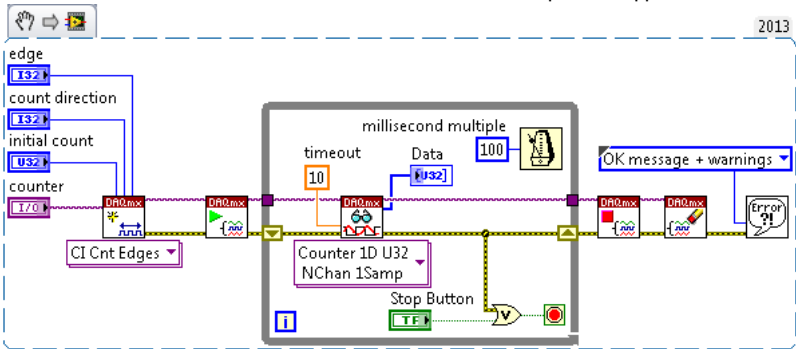
Although it is not always required, using the NI-DAQmx Start Task function to explicitly start a task involving a hardware-timed acquisition or generation is preferred. Furthermore, if the NI-DAQmx Read function or the NI-DAQmx Write function will be executed multiple times, such as in a loop, the NI-DAQmx Start Task function should also be used. Otherwise, the task performance will be reduced because it will be repeatedly started and stopped. Additional information describing when to use the NI-DAQmx Start Task function is available in the When to Use the DAQmx Start Task and DAQmx Stop Task VIs (<http://digital.ni.com/public.nsf/allkb/0622DCB1E143EB7F86256D270078B07F?OpenDocument>) article.

Starting a Task in LabVIEW

The following LabVIEW block diagram demonstrates a situation where the NI-DAQmx Start function does not need to be used because the analog output generation only consists of a single, software-timed sample.



Conversely, the following LabVIEW block diagram demonstrates a situation where the NI-DAQmx Start function should be used because the NI-DAQmx Read function is executed multiple times to read from the counter.



Starting a Task in .NET

To start a task use the Start() function in the Task object. The following code snippet shows the use of the Start() function.

```
analogInTask = new Task();
analogInTask.AIChannels.CreateVoltageChannel( ... );
analogInTask.Timing.ConfigureSampleClock( ... );
analogInTask.Start();
```

Starting a Task in C/C++

To start a task use the DAQmxStartTask() function. A code snippet of its use follows.

```
DAQmxStartTask(taskHandle);
```

Examples to Examine:

LabVIEW	Counter - Continuous Output.vi
	Current - Continuous Input.vi
CVI	Dig Pulse Train-Cont
	Cont 0-20mA Samps-Int Clk
.NET	GenDigPulseTrain_Continuous
	ContAcq0_20mACurrentSamples_IntClk

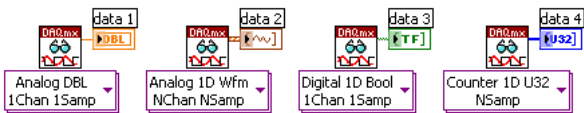
7. NI-DAQmx Read



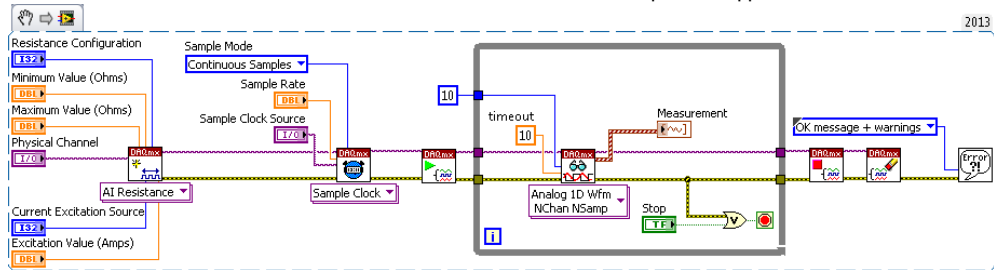
The NI-DAQmx Read function reads samples from the specified acquisition task. The different instances of the function allow for the type of acquisition (analog, digital, or counter), the number of virtual channels, the number of samples, and the data type to be selected. After the specified number of samples have been transferred from the FIFO on the DAQ board to the PC Buffer in RAM, the NI-DAQmx Read function moves samples from the PC Buffer to the Application Development Environment (ADE) Memory.

Reading Data in LabVIEW

The following figures shows four examples of the different instances of the NI-DAQmx Read VI.



Instances of the NI-DAQmx Read function that are capable of reading multiple samples include an input to specify the **number of samples per channel** to read when the function executes. For finite acquisitions, by specifying a **number of samples per channel** of -1, the function waits for all of the requested samples to be acquired and then reads these samples. Specifying a **number of samples per channel** of -1 for a continuous acquisition results in the function reading all of the samples that are currently available in the buffer when the function executes. In the following LabVIEW block diagram, the NI-DAQmx Read VI has been configured to read multiple samples from multiple analog input virtual channels and return the data as waveforms. Furthermore, since the **number of samples per channel** input has been wired to a constant value of 10, each time the VI executes it will read 10 samples from each virtual channel.



Reading Data in .NET

To read with the NI-DAQmx .NET library, use the reader and stream objects. This programming pattern is similar to the pattern used throughout the .NET Framework for file and network I/O.

The **DaqStream** class has properties related to I/O, such as **CurrentReadPosition**, and methods for raw I/O. To get an instance of the **DaqStream** class that corresponds to a particular Task, use the **Stream** property. You cannot instantiate an instance of the **DaqStream** class directly.

To perform a read in the NI-DAQmx library, create an instance of a reader and pass an instance of the **DaqStream** class in the constructor. Then call a method on the reader class to read the data, as shown in the following snippet:

```
analogInTask = new Task();
analogInTask.AIChannels.CreateVoltageChannel( ... );
//Create the reader and attach it to the stream
AnalogSingleChannelReader reader = new AnalogSingleChannelReader(analogInTask.Stream);

//Perform the read
double[] data = reader.ReadMultiSample(100);
```

Reading data in C/C++

Use the **DAQmxReadAnalogF64()** function to read analog data from an analog input task in C or C++. For additional read functions, see the NI-DAQmx C Reference help. A code snippet of its use follows to read 1000 samples from a single channel

```
int numRead = 0;
DAQmxReadAnalogF64(taskHandle, 1000, 10.0, DAQmx_Val_GroupByScanNumber, data, 1000, &numRead, NULL);
```

Examples to Examine:

LabVIEW	Voltage - SW-Timed Input.vi
	Digital - Finite Input.vi
CVI	One Sample
	Read Dig Chan
.NET	AcqOneVoltageSample
	ReadDigChan

8. NI-DAQmx Write



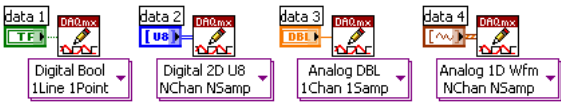
Library»NI-DAQmx»Write Functions [CVI]
ChannelWriter Class [.NET]

The NI-DAQmx Write function writes samples to the specified generation task. The different instances of the function allow for the type of generation (analog or digital), the number of virtual channels, the number of samples, and the data type to be selected. The NI-DAQmx Write Function moves samples from the Application Development Environment (ADE) Memory to the PC Buffer in RAM. These samples are then transferred from the PC Buffer to the DAQ board FIFO in order to be generated.

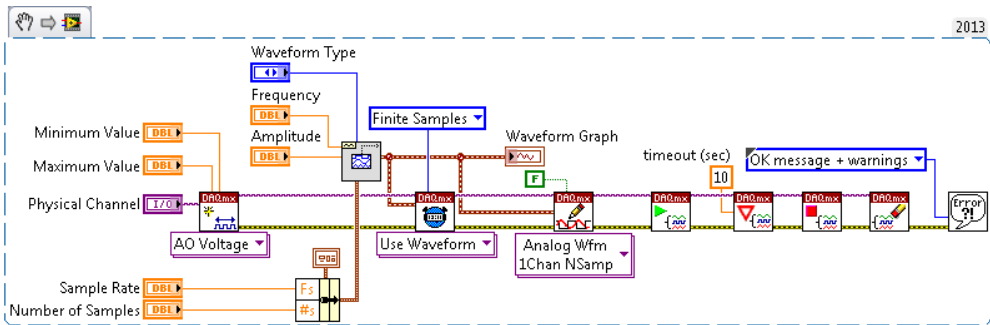
Each instance of the NI-DAQmx Write function has an auto start input to determine if the function will implicitly start the task if it has not already been explicitly started. As was previously discussed in the NI-DAQmx Start Task section of this document, the NI-DAQmx Start Task function should be used to explicitly start a generation task that uses hardware timing. It should also be used to maximize performance if the NI-DAQmx Write function will be executed multiple times.

Writing Data in LabVIEW

The following figures shows four examples of the different instances of the NI-DAQmx Write VI.



The following LabVIEW block diagram, for a finite analog output generation, includes a "False" Boolean constant wired to the auto start input of the NI-DAQmx Write VI because the generation is hardware-timed. The NI-DAQmx Write VI has been configured to write multiple samples of analog output data for one channel to the task as an analog waveform.



Writing Data in .NET

To write with the NI-DAQmx .NET library, use the writer and stream objects. This process is similar to the reading data as discussed above.

To perform a write, create an instance of a `writer` and pass an instance of the **DaqStream** class in the constructor. Then call `WriteMultiSample` method on the writer class to write the data, as shown in the following snippet:

```
analogOutTask = new Task();
analogOutTask.AOChannels.CreateVoltageChannel( ... );
//Create the writer and attach it to the stream
AnalogSingleChannelWriter writer = new AnalogSingleChannelWriter(analogOutTask.Stream);
//Perform the write
double[] data = writer.WriteMultiSample(100);
```

Writing Data in C/C++

To write analog data use the `DAQmxWriteAnalogF64()` function. For information on writing other forms of data see the NI-DAQmx C Reference Help. A code snippet for writing 1000 samples of analog data follows.

```
int written = 0;
DAQmxWriteAnalogF64(taskHandle, 1000, 0, 10.0, DAQmx_Val_GroupByChannel, data, &written, NULL);
```

Examples to Examine

LabVIEW	Voltage - Finite Output.vi
	Digital - Finite Output.vi
CVI	Volt Update
	Write Dig Chan
.NET	GenVoltageUpdate
	WriteDigChan

9. NI-DAQmx Wait Until Done

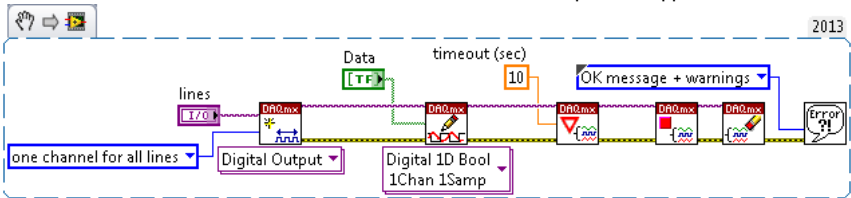


Library»NI-DAQmx»Task Configuration/Control»Wait Until Task Done [CVI]
Task.WaitUntilDone Method [.NET]

The NI-DAQmx Wait Until Done function waits for the data acquisition operation to complete before finishing. This function should be used to ensure that the specified acquisition or generation is complete before the task is stopped. Most commonly, the NI-DAQmx Wait Until Done function is used with finite operations. Once this function has finished executing, the finite acquisition or generation is complete and the task can be stopped without disturbing the operation. Additionally, the **timeout** input allows a maximum wait time to be specified. If the acquisition or generation does not complete within this amount of time the function exits and an appropriate error is generated.

Wait Until Done in LabVIEW

The NI-DAQmx Wait Until Done VI is used in the following LabVIEW block diagram to verify that the finite digital output operation is complete before the task is cleared.



Wait Until Done in .NET

For more information on using this functionality in .NET, see the GenMultVoltUpdates_IntClk shipping example.

Wait Until Done in C/C++

The DAQmxWaitUntilTaskDone() function is used to wait until all sample in the buffer have been generated. Call it after a write or start function.

Examples to Examine

LabVIEW	Digital - Finite Output.vi
	Voltage - Finite Output.vi
CVI	Dig Pulse
	Mult Volt Updates-Int Clk
.NET	GenDigPulse
	GenMultVoltUpdates_IntClk

10. NI-DAQmx Clear Task

task in

error in

error out

DAQmx Clear Task.vi

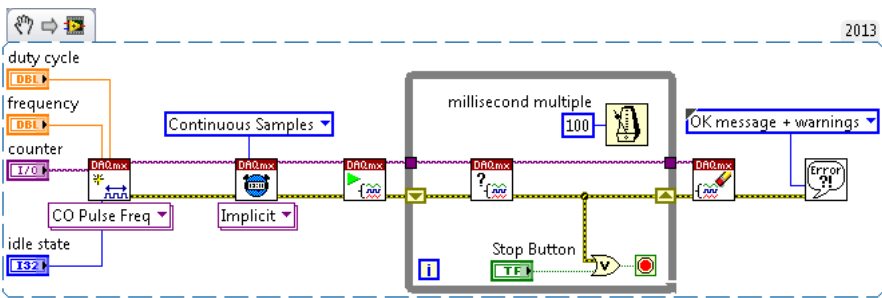
Library»NI-DAQmx»Task Configuration/Control»Clear Task [CVI]

Task.Dispose Method [.NET]

The NI-DAQmx Clear Task function clears the specified task. If the task is currently running, the function first stops the task and then releases all of its resources. Once a task has been cleared, it cannot be used unless it is recreated. Thus if a task will be used again, the NI-DAQmx Stop Task function should be used to stop the task but not to clear it. For continuous operations, the NI-DAQmx Clear Task function should be used to stop the actual acquisition or generation.

Clearing a Task in LabVIEW

In the following LabVIEW block diagram, a continuous pulse train is being generated with a counter. The pulse train continues to be output until the While Loop is exited and the NI-DAQmx Clear Task VI executes.



Clearing a Task in .NET

Once a task is finished reading or writing, simply call the Task.Dispose method.

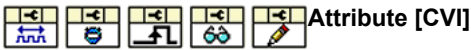
Clearing a Task in C/C++

Once a task is finished reading or writing, use the DAQmxClearTask() function.

Examples to Examine

LabVIEW	Voltage - Continuous Output.vi
	Counter - Count Edges (Continuous Clock).vi
CVI	Cont Gen Volt Wfm-Int Clk
	Cnt Dig Events
.NET	ContGenVoltageWfm_IntClk

11. NI-DAQmx Properties



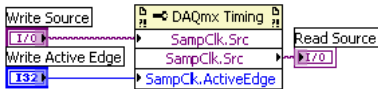
Property [.NET]

NI-DAQmx Properties provide access to all of the properties associated with a data acquisition operation. These properties can be set by writing to the NI-DAQmx Properties, and the current values of the properties can be read from the NI-DAQmx Properties as well.

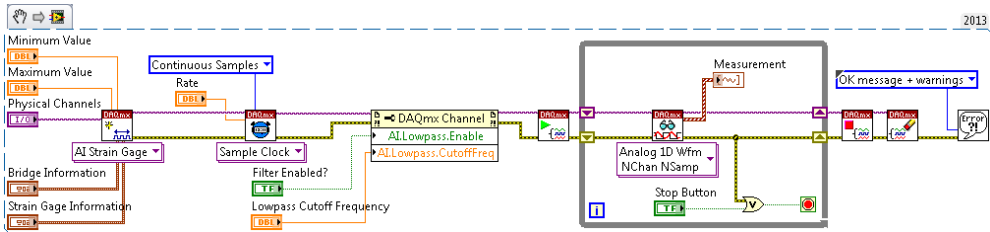
Many properties can be set using the NI-DAQmx functions discussed previously. The **Sample Clock Source** and **Sample Clock Active Edge** properties, for instance, can be set using the NI-DAQmx Timing function. However, some of the less frequently used properties can only be accessed through the NI-DAQmx Properties.

Using Properties in LabVIEW (Property Nodes)

In LabVIEW, one NI-DAQmx Property Node can be used to write to and/or read from multiple properties. For example, the following LabVIEW NI-DAQmx Timing Property Node sets the source of the sample clock. Then, the source of the sample clock is read. Finally, the active edge of the sample clock is set.



In the following LabVIEW block diagram, an NI-DAQmx Channel Property Node is used to enable the hardware lowpass filter and then set the filter cutoff frequency for use with a strain gage measurement.



Using Properties in .NET

As noted above in the Timing section, many properties are set from within subclasses and collections inside of the Task object. A common property use in NI-DAQmx is shown in the following snippet:

```
//Create a new NI-DAQmx Task
Task t = new Task();

//Access the subobject properties of the Task class
t.Timing.SamplesPerChannel = 1000;
```

Using Properties in C/C++

To get or set properties for a task there are individual getter and setter functions for each individual property. For additional information see the list of properties and their functions in the NI-DAQmx C Reference Help under NI-DAQmx C Properties.

Examples to Examine

LabVIEW	Strain - Continuous Input.vi
	Thermocouple (with OTCD) - Continuous Input.vi
CVI	Cont Strain Samples
	Cont Accel Samps-Int Clk-Anlg Start
.NET	AcqStrainSamples
	ContAcqAccelSamp_IntClk_AnalogStart

12. Conclusion

NI-DAQmx saves development time and improve the performance of data acquisition applications. One of the ways NI-DAQmx saves development time is by providing an API that requires only a small number of functions to expose the majority of its functionality. In fact, 80% of data acquisition applications can be solved after learning only the 10 functions discussed in this document.