LabVIEW NXG Core 1: Acquire, Analyze, Visualize Participant Guide

Course Software Version LabVIEW NXG 1.0
May 2017 Edition
Part Number 326667A-01

Copyright

© 2017 National Instruments. All rights reserved.
Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

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

End-User License Agreements and Third-Party Legal Notices

You can find end-user license agreements (EULAs) and third-party legal notices in the following locations:

- Notices are located in the `<National Instruments>\_Legal Information` and `<National Instruments>` directories.
- EULAs are located in the `<National Instruments>\Shared\MDF\Legal\License` directory.
- Review `<National Instruments>\_Legal Information.txt` for more information on including legal information in installers built with NI products.

Trademarks

Refer to the *NI Trademarks and Logo Guidelines* at ni.com/trademarks for more information on National Instruments trademarks.

ARM, Keil, and µVision are trademarks or registered of ARM Ltd or its subsidiaries.

LEGO, the LEGO logo, WEDO, and MINDSTORMS are trademarks of the LEGO Group.

TETRIX by Pitsco is a trademark of Pitsco, Inc.

FIELDBUS FOUNDATION™ and FOUNDATION™ are trademarks of the Fieldbus Foundation.

EtherCAT® is a registered trademark of and licensed by Beckhoff Automation GmbH.

CANopen® is a registered Community Trademark of CAN in Automation e.V.

DeviceNet™ and EtherNet/IP™ are trademarks of ODVA.

Go!, SensorDAQ, and Vernier are registered trademarks of Vernier Software & Technology. Vernier Software & Technology and vernier.com are trademarks or trade dress.

Xilinx is the registered trademark of Xilinx, Inc.

Taptite and Trilobular are registered trademarks of Research Engineering & Manufacturing Inc.

FireWire® is the registered trademark of Apple Inc.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

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

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

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

The ExpressCard™ word mark and logos are owned by PCMCIA and any use of such marks by National Instruments is under license.

The mark LabWindows is used under a license from Microsoft Corporation. Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

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

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

Patents

For patents covering National Instruments products/technology, refer to the appropriate location: **Help»Patents** in your software, the patents.txt file on your media, or the *National Instruments Patent Notice* at ni.com/patents.

Worldwide Technical Support and Product Information

ni.com

Worldwide Offices

Visit ni.com/niglobal to access the branch office websites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

For further support information, refer to the *Additional Information and Resources* appendix. To comment on National Instruments documentation, refer to the National Instruments website at ni.com/info and enter the Info Code feedback.

Exercise 2-1 Connecting your NI Hardware

Goal

Explore the physical connections between the sensors/signals, NI DAQ hardware, and the computer.

Description

You are trying to create an application that acquires the following measurements:

- Temperature measurement from thermocouple sensor
- Vibration measurement from accelerometer sensor
- Voltage measurement from solar panel

In this exercise, you will explore how to connect the sensor, NI DAQ device, and the computer.

Simulated Hardware Option

Complete the steps in this section if you do not have access to the cDAQ Demo Kit hardware used in classroom version of this course.

Use MAX to create the following simulated DAQ hardware. The configuration of this simulated cDAQ device matches the configuration of the physical cDAQ Demo Kit used in LabVIEW NXG classroom training.

Table 2-1. Simulated CompactDAQ Device

cDAQ-9178 Chassis			
Slot	**Instrument**	**Name**	**Description**
1	NI 9236	cDAQ1Mod1	Strain/Bridge Input
2	NI 9213	cDAQ1Mod2	Thermocouple Input
3	NI 9472	cDAQ1Mod3	Digital Output
4	NI 9263	cDAQ1Mod4	Voltage Output
5	Empty	N/A	N/A
6	Empty	N/A	N/A
7	NI 9234	cDAQ1Mod7	Sound and Vibration Input
8	NI 9215	cDAQ1Mod8	Voltage Input

1. Create simulated hardware.

   ☐ Launch **LabVIEW NXG** from the Windows Start button.

   ☐ Select the **Hardware** tab.

   ☐ Click the **Launch NI MAX** link on the right to open the Measurement & Automation Explorer (MAX).

☐ In the Measurement & Automation Explorer, select **Devices and Interfaces**, click the **Create New** button.

☐ In the Create New dialog box, select **Simulated NI-DAQmx Device or Modular Instrument**, and click the **Finish** button.

☐ Expand **CompactDAQ Chassis** and select **NI cDAQ-9178**. Name the simulated device `cDAQ-9178` under Settings.

☐ Click the **Configure Simulated cDAQ Chassis** button and create simulated instruments for each of the slots as described in Table 2-1.

☐ Close the Measurement & Automation Explorer when finished.

cDAQ Demo Kit

If you are in the classroom version of this course and you have access to a cDAQ Demo Kit, complete the steps in this section.

Refer to Figure 2-3 to connect the cDAQ Demo Kit.

Figure 2-3. cDAQ Demo Kit Connections

Explore the Connections Between the NI DAQ Device and Sensors/Signals

1. In LabVIEW NXG, go to the **Hardware** tab.
2. Navigate to the Configure Hardware overlay of your DAQ device.
   - ☐ Click the cards until you find a DAQ card that has a Configure button.
     - **CompactDAQ Device**—CompactDAQ Chassis»CompactDAQ Module
     - **PCI/PXI DAQ Device**—PC/PXI»DAQ device
     - **USB DAQ Device**—USB DAQ device
   - ☐ Click the **Configure** button to launch the Configure Hardware overlay.

Figure 2-4. Configure Hardware Overlay—NI 9213 Temperature Input Module

- ☐ Click the **Pinout** button. Explore the Pinout Information and close the dialog box when you are finished.
- ☐ Click the **Manual** button. Explore the information contained in the manuals.

Additional Instructions if you are using the cDAQ Demo Kit

1. Notice how the thermocouple, accelerometer, and solar panel voltage are connected to the corresponding CompactDAQ modules.

Sensor/Signal	cDAQ Device
Thermocouple	NI 9213
Accelerometer	N 9234
Solar Panel	NI 9215

Figure 2-5. NI 9213 Pinout and Signal Descriptions

2. **(Your Turn)** Explore how the rest of the sensors/signals are connected to the cDAQ device.

Signal/Sensor	DAQ Device
Strain Gage	NI 9236
LEDs	NI 9472
Voltage Controls Fan Speed	NI 9263

End of Exercise 2-1

Exercise 2-2 Validate I/O (DAQ)

Goal

Create a project with DAQ hardware, and verify that the I/O behaves correctly.

Scenario

You want to validate that your DAQ system is acquiring the following data correctly:

- Voltage signal—acquire analog input voltage measurement
- Thermocouple sensor—acquire analog input temperature measurement
- Accelerometer sensor—acquire analog input acceleration measurement
- Fan speed control—generate analog output voltage
- Green LED—generate digital output voltage

Guided Instruction

1. Create a new project.

   ☐ Launch LabVIEW NXG.

   ☐ Select **File»New»Project**.

   ☐ Select **File»Save All**.

   ☐ Set the following items.

   – **Project Name**—Validate Signals

   – **Location**—Click button and browse to the `<Exercises>\LabVIEW NXG Core 1` directory.

   ☐ Click **Save**.

2. Explore your newly created folder and files.

   ☐ In Windows Explorer, navigate to the `<Exercises>\LabVIEW NXG Core 1` directory.

   ☐ Notice that a new `Validate Signals` folder has been created.

   ☐ Notice that the Validate Signals folder contains a new LabVIEW project file, `Validate Signals.lvproject`.

Validate Signals

Validate Analog Input Voltage

1. Connect an analog input voltage signal to your DAQ device.

   ☐ **(cDAQ Demo Kit)** The X Acceleration signal is connected to channel 0 of the NI 9234 module in slot 7.

   Verify the following Sound & Vibration Signal Simulator settings.

   – Fan Speed Control switch: `DIAL`

   – Fan switch: `UNBALANCED`

   – Turn the dial all the way to the right (clockwise).

   The Unbalanced Fan should now be rotating at its fastest speed, which is around 5000 RPM (i.e. 83.3 Hz, or rotations per second). The X Acceleration signal is expected to look like an 83.3 Hz sine wave signal.

2. Add an Analog Input Voltage measurement panel.

   ☐ In the Project Files tab, click the **Add Item** button and select **Analog Input** to create a measurement panel.

   ☐ Click the Stop button and save the new task.

   ☐ In the file tab, right-click the newly created Analog Input.task item and select **Rename**. Rename the item `Analog Input Voltage.task`.

3. Configure the measurement.

   ☐ On the **Channel** tab in the configuration pane on the right, configure the settings in Table 2-2.

Table 2-2. Analog Input Task Channel Settings

Name	Input 0
Signal Type	Voltage
Channel	cDAQ1Mod7/ai0
Voltage Input Minimum	-50 mV
Voltage Input Maximum	50 mV
Advanced»Coupling	AC

4. Explore auto managed timing.

   ☐ Select the Task tab and configure the timing settings as shown in the following figure and run the task.

   ☐ Set History Duration to values of `0.1 s`, `1.0 s`, `2.0 s`, `5.0 s`, and `10.0 s` to adjust how many seconds of data are shown on the measurement panel chart.

5. Manually manage the timing settings.

   Notice that the timing settings are Auto Managed by default.

   ☐ Disable the **Auto Managed Timing** checkbox.

☐ Set the desired sample rate.

– What sample rate is required for you to get a good shape of your analog voltage signal?_____

Note To accurately determine the frequency of your signal-of-interest, you must set your sampling rate to greater than 2x the frequency of the signal. To accurately display the shape of your signal-of-interest, you should set your sampling rate to greater than 10-15x the frequency of the signal.

(cDAQ Demo Kit) The maximum expected frequency of the X Acceleration signal that we want to measure is ~609 Hz, so you should set the sampling rate to greater than 1218 Hz (609 Hz x 2). For this exercise, set the sampling rate to `2.5600 kHz`.

6. Shutdown your analog input signal, if necessary.

   If you are using the cDAQ Demo Kit, turn the dial on the Sound and Vibration Signal Simulator all the way to the left to turn off the fan.

7. Try an alternate method of launching a measurement panel through the **Hardware** tab.

   ☐ Click the Home button.

   ☐ Click the **Hardware** tab.

   ☐ Navigate to the hardware you want to validate. For example, navigate to **CompactDAQ Chassis»C Series Sound and Vibration Input (NI 9234)»Create Analog Input**.

Notice that this creates a new measurement panel in your project. LabVIEW chooses a default measurement panel (analog input voltage, digital output, strain gage, and so on) and channel that matches the hardware you chose.

You can now continue configuring your measurement panel (channel, settings, rename, and so on) like in the previous steps.

☐ Stop, save, and close the measurement panel when finished.

Select one or more of the following signals to validate.

Complete one or more of the following sections to try validating different types of signals.

Validate Thermocouple

1. Add a Thermocouple measurement panel.

   ☐ In the Project Files tab, click the **Add Item** button and select **Analog Input** to create a measurement panel.

   ☐ Click the Stop button and save the new task.

   – In the File tab, right-click the newly created Analog Input.task item and select **Rename**. Rename the item `Thermocouple.task`.

2. Configure the measurement.

   ☐ On the **Channel** tab, configure the settings shown in Table 2-3.

Table 2-3. Thermocouple Input Task Channel Settings

Name	Input 0
Signal Type	Thermocouple
Channel	cDAQ1Mod2/ai0
Unit	Degree Celsius (°C)
Thermocouple Input Minimum	20 °C
Thermocouple Input Maximum	30 °C
Thermocouple Type	J
CJC Source Type	Constant Temperature

3. Configure the task timing.

   ☐ On the **Task** tab, configure the timing settings as shown in the following figure and run the task.

   ☐ Set History Duration to values of `0.1 s`, `1.0 s`, `2.0 s`, `5.0 s`, and `10.0 s` to adjust how many seconds of data are shown on the measurement panel chart.

4. Manually manage the timing settings.
   Notice that the timing settings are Auto Managed by default.
   - ☐ Disable the **Auto Managed Timing** checkbox.
   - ☐ Set the desired sample rate.
5. **(cDAQ Demo Kit)** Verify the signal is correct.
   - ☐ Verify that the thermocouple measurement panel displays the expected temperature. For example, room temperature is usually around 25 degrees Celsius.
   - ☐ Warm up the thermocouple by rubbing the thermocouple with your fingers. Verify that the temperature rises on the measurement panel.
   - ☐ Click the Stop button to stop the acquisition.

Validate AO Fan Control

1. Add an Analog Output measurement panel.
   - ☐ In the Project Files tab, click the **Add Item** button and select **Analog Output** to create a measurement panel.
2. Configure the measurement.
   - ☐ On the **Channel** tab, configure the settings shown in Table 2-4.

Table 2-4. Analog Output Task Channel Settings

Name	Output 0
Signal Type	Voltage
Channel	cDAQ1Mod4/ao1

3. Validate the signal is correct.

   ☐ Click the Run button to start outputting your specified voltage.

   ☐ You can change the output voltage in the Voltage field of the Channel tab, as shown in the following figure. The current analog output voltage is shown in the middle of the measurement panel.

4. **(cDAQ Demo Kit Option)** Verify that the output voltage is correct.

   ☐ The cDAQ1Mod4/ao1 analog output channel is connected to the Fan Control input of the Sound & Vibration Signal Simulator.

   ☐ Verify that the following Sound & Vibration Signal Simulator switch settings

   – Fan Speed Control switch: `BNC`

   – Fan switch: `UNBALANCED`

   ☐ As you change the analog output voltage from 0 V to 1 V to 2 V to 3 V to 4 V to 5 V, you should notice and hear the fan speed increase from not moving to the maximum fan speed.

5. Click the Stop button to stop the analog output voltage generation.

Validate X Accelerometer

1. Add an Accelerometer measurement panel.

   ☐ In the Project Files tab, click the **Add Item** button and select **Analog Input** to create a measurement panel.

   ☐ Click the Stop button and save the new task.

   ☐ In the file tab, right-click the newly created Analog Input.task item and select **Rename**. Rename the item `Accelerometer.task`.

2. Configure the measurement.

   ☐ On the **Channel** tab, configure the settings in Table 2-2.

Table 2-5. Accelerometer Task Channel Settings

Name	Input 0
Signal Type	Accelerometer
Channel	cDAQ1Mod7/ai0
IEPE Accelerometer Input Minimum	-1 g
IEPE Accelerometer Input Maximum	1 g
Excitation Source	None
Sensitivity	175 mV/g
Sensitivity Units	mV/g

3. Configure the sample rate on the Task tab.

   Note To accurately determine the frequency of your signal-of-interest, you must set your sampling rate to greater than 2x the frequency of the signal. To accurately display the shape of your signal-of-interest, you should set your sampling rate to greater than 10-15x the frequency of the signal.

   (cDAQ Demo Kit) The maximum expected frequency of the X Acceleration signal that we want to measure is ~609 Hz, so you should set the sampling rate to greater than 1218 Hz (609 Hz x 2). For this exercise, set the sampling rate to `2.5600 kHz`.

4. **(cDAQ Demo Kit)** Verify the signal is correct.

   ☐ Run the measurement panel.

   ☐ Firmly tap the Sound &Vibration Signal Simulator and see corresponding acceleration spike in the measurement panel.

   ☐ Adjust fan speed. See the acceleration amplitude increase as the fan speed increases in the measurement panel.

Validate Digital Output

1. Add a Digital Output measurement panel.

   ☐ In the Project Files tab, click the **Add Item** button and select **Digital Output** to create a measurement panel.

2. Configure the measurement.

   ☐ On the **Channel** tab, configure the settings shown in Table 2-4.

Table 2-6. Analog Output Task Channel Settings

Name	Line 0
Channel	cDAQ1Mod3/port0/line0

3. Validate the signal is correct.

   ☐ Click the Run button to start outputting your specified digital voltage state.

   ☐ Update the digital voltage state to output by clicking the High/Low switch, shown in the following figure.

4. **(cDAQ Demo Kit)** Verify that the output voltage is correct.

   The cDAQ1Mod3/port0/line0 digital output channel is connected to the cDAQ Demo Kit LEDs.

   ☐ Setting cDAQ1Mod3/port0/line0 to **Low** should cause the cDAQ Demo Kit LED0 to be unlit.

   ☐ Setting cDAQ1Mod3/port0/line0 to **High** should cause the cDAQ Demo Kit LED0 to be lit.

5. Click the **Stop** button to stop the digital output generation.

Your Turn

Try validating one or more of the following sensors or signals:

- Analog Input Voltage—Light sensor on cDAQ1/Mod8/ai0.
- Analog Input Accelerometer—Y Accelerometer on cDAQ1/Mod7/ai1
- Digital Output—Red LED on cDAQ1/Mod3/port0/line7
- Digital Input—Trigger button on cDAQ1/PFI0

On the Job

What type of sensors and/or signals do you need to acquire data from?

End of Exercise 2-2

Exercise 2-3 Interactively Analyze Acquisition Data

Goal

Validate acquisition data by using Analysis Panels, and export data to Excel for further analysis.

Scenario

You want to validate that your DAQ system is acquiring the following data correctly:

- Accelerometer sensor—acceleration measurement

To verify the acquisition data is correct, you need to do additional analysis on the raw acquisition data. In this case, you need to view tone measurements of the acceleration signal in the frequency domain.

(cDAQ Demo Kit) The Sound and Vibration Signal Simulator fans are four pole fans with seven blades each. Therefore, we expect to see 4th and 7th order vibrations. The unbalanced fan has one of its blades cut off, so it should have significant 1st order vibrations as well.

We expect that the unbalanced fan rotates at a maximum frequency of ~5,200 RPM (rotations per minute). At 5,200 RPM, we expect the frequency-domain graph to show spikes at the following vibration frequencies:

- 1st order vibration: ~87 Hz (5,200 RPM / 60 seconds).
- 4th order vibration: ~348 Hz (1st order vibration * 4)
- 7th order vibration: ~609 Hz (1st order vibration * 7)

Figure 2-6. Example of Frequency-Domain Graph

Instructions

1. Open your previously created project `<Exercises>\LabVIEW NXG Core 1\Validate Signals\Validate Signals.lvproject`.

2. In the Project Files tab, double-click the **Analog Input Voltage.task** item to open the measurement panel that you previously configured.

3. Connect an analog input voltage signal to your DAQ device.

   (cDAQ Demo Kit Option) In the cDAQ Demo Kit, the X Acceleration signal is connected to channel 0 of the NI 9234 module in slot 7.

   ☐ Verify the following Sound and Vibration Signal Simulator settings and turn the dial clockwise all the way to the right:

   – Fan Speed Control switch: `DIAL`

   – Fan switch: `UNBALANCED`

   The Unbalanced Fan should now be rotating at its fastest speed, which is approximately 5,000 RPM, which is 83.3 Hz, or rotations per second. The X Acceleration signal should look like an 83.3 Hz sine wave signal.

Capture Data from the Measurement Panel

1. Click the Run button to start acquiring data.

2. Click the Stop button when the measurement panel shows a set of data that you would like store or analyze.

3. Capture data by right-clicking the measurement panel graph and selecting **Capture Data**. This saves the data as a data item in the **Captured Data** tab.

Note If you want to capture a subset of the data in the graph, then first crop the data using the black triangles in the top graph, as shown in the following figure, before capturing the data.

4. View your captured data.

   ☐ In the Captured Data tab, right-click the Capture 1 data item and select **Rename**.

   ☐ Rename the captured data `Analog Input Voltage Capture`.

   ☐ Double-click the **Analog Input Voltage Capture** data item to view the captured data.

- ☐ Use the pull-down menu section to view your captured data in **Graph**, **Statistics**, and **Table** formats.

5. Select **File»Save All** to include this data item your project.

Note You can delete unwanted data items by right-clicking the data item and selecting **Delete selected item**.

Analyze Your Data—Analysis Panels

1. In the **Analog Input Voltage Capture** tab in the middle section, set the pull-down menu near the top-right corner back to **Graph**.

2. In the configuration pane, select the Plots tab and notice the Interactive Analysis section.

   Selecting any of the items in this section will launch an interactive Analysis Panel you can use to analyze your data, such as filter, FFT spectrum, tone measurements, amplitude measurements, and statistics.

3. **(Simulated Hardware)** Import captured data from a real unbalanced fan if you are using simulated hardware.

   - ☐ In the Captured Data tab, click the Import button and browse to `<Exercises>\LabVIEW NXG Core 1\Example Captured Data\Unbalanced Fan Data.tdms`.

   - ☐ Double-click the **Unbalanced Fan Data** data item to view the captured data.

   - ☐ Notice that the Configuration Pane displays the captured data's waveform properties, such as Rate and Duration.

4. Explore the Analyze Tone Measurements analysis panel.

   - ☐ In the Interactive Analysis section, click **More Analysis Panels»Tone Measurements**.

   - ☐ In the configuration pane, set the following values.

Function Configuration	Multiple Tone Measurements
Max Number of Tones	6

Figure 2-7. Analysis Panel Tone Measurements

In Figure 2-7, you see a large tone at a 1st order frequency of 84.78 Hz, which is close to the expected value stated in the Scenario section of this exercise. As expected, you also see tones at the 4th order (339 Hz) and 7th order (593 Hz) frequencies.

5. Close the Analysis Panel when you are finished.

Additional Analysis Panels

If you have time, explore some of these other Analysis Panels.

Amplitude Measurements	View the maximum peak, minimum peak, and peak-to-peak results
Statistics Analysis Panel	Perform statistical calculations the signal
Filter Analysis Panel	Apply a lowpass filter to the signal

Analyze Data—Export to File

1. In the Captured Data tab, select the Analog Input Voltage Capture data item, and then click the Export button. Set Save as type to **CSV Files**. Save the file as `<Exercises>\LabVIEW NXG Core 1\Validate Signals\Analog Input Voltage Captured Data.csv`.

2. Explore the data in Microsoft Excel.

   ☐ Open Microsoft Excel.

☐ In Excel, browse to and open `<Exercises>\LabVIEW NXG Core 1\Validate Signals\Analog Input Voltage Captured Data.csv`.

	A	B
1	Timestamp	10/20/2016 11:25
2	Interval	0.000390625
3	Channel Name	Sensor 0
4	Unit	V
5	0.000000000	0.000968955
6	0.000390625	0.000867673
7	0.000781250	0.002351512
8	0.001171875	0.003029367
9	0.001562500	0.000997631
10	0.001953125	-0.009879175
11	0.002343750	0.004592523
12	0.002734375	-0.005926746
13	0.003125000	-0.006920649
14	0.003515625	-0.002820569
15	0.003906250	-0.006719916
16	0.004296875	-0.008202535
17	0.004687500	-0.008486245
18	0.005078125	-0.005560057
19	0.005468750	-0.005912713
20	0.005859375	0.004155536

Time (s)

Amplitude (V)

Now that you have exported your data to file, you can continue to analyze your data in Excel and other programs, if necessary.

(Optional) Record Data from the Measurement Panel

In addition to capturing data from an existing graph, you can also record data.

1. Record data from the Analog Input Voltage task.

   ☐ In the Project Files tab, double-click the **Analog Input Voltage.task** item to open the measurement panel that you previously configured.

   ☐ Click the Record button to start recording data.

   ☐ Click the Stop button when you have finished recording all the data you need.

2. View your recorded data.

   ☐ In the Captured Data tab, right-click the **Recording 1** data item and select **Rename**.

   ☐ Rename the recording `Analog Input Voltage Recording`.

   ☐ Double-click the **Analog Input Voltage Recording** data item to view the captured data.

   Notice that the Configuration Pane displays the captured data's waveform properties, such as Rate and Duration.

3. Select **File»Save All** to save the project.

   This data item is now included in your project. Because your recorded data is a data item, you can perform all the same actions on it that you practiced in this exercise.

Your Turn

Analyze the maximum and minimum values that the Light Sensor (included in cDAQ Demo Kit) outputs. Use the following general steps as a guideline.

1. Create a new project.
2. Add an Analog Input measurement panel that measures the voltage from the Light Sensor using the **cDAQ»NI 9215 Module»Channel ai0**.
3. Record or capture data that contains the Light Sensor detecting both a bright light and darkness.
4. Use the Amplitude Measurements analysis panel to determine the Maximum Peak (V) and Minimum Peak (V) values of your captured data.

On the Job

After acquiring data from your hardware, what analysis/processing do you need to do?

If there is an Analysis Panel that will accomplish that, write down the name of the Analysis Panel here.

End of Exercise 2-3

Exercise 3-1 Connect Hardware (Non-NI Instrument)

Goal

Connect a non-NI Instrument to your computer, so that the instrument can be programmatically controlled by LabVIEW NXG.

Description/Scenario

You are trying to create an application that acquires data from a non-NI instrument (e.g., non-NI oscilloscope, non-NI digital multimeter).

In this exercise, you will explore how to connect a non-NI instrument to your computer.

Connect a non-NI Instrument to Your Computer

In the classroom version of this course, the NI Instrument Simulator simulates a non-NI instrument that has a GPIB port.

Figure 3-2. NI Instrument Simulator

For this course, imagine that the NI Instrument Simulator represents a non-NI oscilloscope instrument that has a GPIB port.

In the classroom version of this course, your computer has a GPIB instrument control device, such as the GPIB-USB-HS, which allows your computer to communicate with a GPIB instrument.

Figure 3-3. GPIB Instrument Control Device for Computer

Follow these steps to connect the GPIB instrument to your computer.

1. Notice that the NI Instrument Simulator has a GPIB port.
2. Connect the NI Instrument Simulator instrument to your computer GPIB interface (for example, GPIB-USB-HS) with a GPIB cable.
3. Now your computer has the ability to programmatically control (send commands, receive data) the NI Instrument Simulator (that it, non-NI instrument).

Set Up the NI Instrument Simulator

This exercise uses the NI Instrument Simulator to simulate an instrument. Before NI MAX can recognize the NI Instrument Simulator as a device with a GPIB interface, you must configure the NI Instrument Simulator using the Instrument Simulator Wizard.

Complete the following steps to configure the NI Instrument Simulator to have a GPIB interface.

1. Configure the NI Instrument Simulator.

   ☐ Power off the NI Instrument Simulator.

   ☐ Set the configuration switch on the rear panel to CFG, as shown in Figure 3-4.

Figure 3-4. NI Instrument Simulator

   ☐ Power on the NI Instrument Simulator using the power switch on the front of the unit.

   ☐ Verify that the **PWR** LED is lit and the **RDY** LED is flashing.

- ☐ Launch the NI Instrument Simulator Wizard from **Start»All Programs»National Instruments»Instrument Simulator»Instrument Simulator Wizard**.

Note This wizard is installed with the NI Instrument Simulator Software, available for download at `ni.com`.

- ☐ Click **Next**.
- ☐ Click **Next**.
- ☐ On the **Select Interface** page, select **GPIB Interface** and click **Next**.
- ☐ Select **Change GPIB Settings** and click **Next**.
- ☐ Select **Single Instrument Mode** and click **Next**.
- ☐ Set **GPIB Primary Address** to `1`.
- ☐ Set **GPIB Secondary Address** to `0(disabled)`.
- ☐ Click **Next**.
- ☐ Click **Update**.
- ☐ Click **OK** when you get the message that the update was successful.
- ☐ Power off the NI Instrument Simulator using the power switch on the front of the unit.
- ☐ Set the configuration switch on the rear panel to **NORM**.
- ☐ Power on the NI Instrument Simulator using the power switch on the front of the unit.
- ☐ Verify that both the **PWR** and **RDY** LEDs are lit.

Implementation

1. Launch **LabVIEW NXG** from the Windows Start button.
2. Go to the **Hardware** tab. Click the **Launch NI MAX** link on the right to open the Measurement & Automation Explorer (MAX).
3. View the settings for the GPIB interface.
   - ☐ In the Measurement & Automation Explorer, expand the **Devices and Interfaces** section to display the installed interfaces. If a GPIB interface is listed, the NI-488.2 software is correctly loaded on the computer.
   - ☐ Select the GPIB interface.
   - ☐ Examine but do not change the settings for the GPIB interface.

4. Communicate with the GPIB instrument.

   - ☐ Make sure the GPIB interface is still selected in the **Devices and Interfaces** section.
   - ☐ Click the **Scan for Instruments** button on the toolbar.
   - ☐ Expand the GPIB interface selected in the **Devices and Interfaces** section. An instrument named `Instrument Simulator` appears.
   - ☐ Click `Instrument Simulator` to display information about it in the right pane of MAX. Click the **Settings** tab on the bottom. Notice the NI Instrument Simulator has a GPIB primary address.
   - ☐ Click the **Communicate with Instrument** button on the toolbar. An interactive window appears. You can use it to query, write to, and read from that instrument.
   - ☐ Enter `*IDN?` in the **Send String** text box and click the **Query** button. The instrument returns its make and model number in the **String Received** indicator as shown in Figure 3-5. You can use this communicator window to debug instrument problems or to verify that specific commands work as described in the instrument documentation.

Figure 3-5. Communication with the GPIB instrument

   - ☐ Enter `MEASURE:VOLTAGE:DC?` in the **Send String** text box and click the **Query** button. The NI Instrument Simulator returns a simulated voltage measurement.
   - ☐ Click the **Query** button again to return a different value.
   - ☐ Click the **Exit** button when done.

5. Set a VISA alias of `MyGPIBInstrument` for the NI Instrument Simulator so you can use the alias instead of having to remember the primary address.

   ☐ While `Instrument Simulator` is selected in MAX, click the **VISA Properties** tab.

   ☐ Enter `MyGPIBInstrument` in the **VISA Alias on My System** field. You use this alias later in the course.

   ☐ Click **Save**.

6. Select **File»Exit** to exit MAX.

7. Click **Yes** if prompted to save the instrument.

8. Now that you have configured your instrument in MAX, you will be able to see your instrument in LabVIEW NXG in the **Hardware** tab. Notice that your VISA alias for the instrument shows up here.

End of Exercise 3-1

Exercise 3-2 Validate I/O (Non-NI Instrument)

Goal

Validate I/O from a non-NI instrument by running its corresponding instrument driver example programs.

Guided Instruction

Install the Instrument Driver

Follow these steps to install the instrument driver for your non-NI instrument.

1. Click the Windows Start button and launch **NI Package Manager**.
2. In the Available section, click **Drivers**.
3. Select **NI Instrument Simulator**. Use the search box to find this driver faster.
4. Click **Install**.
5. When finished, close the NI Package Manager.

Run the Instrument Driver Example Programs

Follow these steps to run the instrument driver example programs for your non-NI instrument. These following steps are specifically for the NI Instrument Simulator, but the steps are similar if you are using a different non-NI instrument.

1. Launch LabVIEW NXG.
2. Go to the LEARNING tab.
3. Navigate to **Examples»Hardware Input and Output»Instrument Driver Examples» NI Instrument Simulator** and click **Create**.

   This creates a project that includes multiple example programs.
4. In the Project Files tab, double-click one of the example programs to open it.
5. In the example program, set the VISA resource name control to `MyGPIBInstrument`.

   Note `MyGPIBInstrument` corresponds to the VISA alias you defined for your instrument in the previous exercise. If you did not define a VISA alias for your GPIB instrument, its name will default to `<GPIB Interface ID>::<Primary Address of the Instrument>::INSTR`, for example `GPIB0::1::INSTR`.
6. Click the Run button on the toolbar.

7. Use the example program to validate the I/O of your non-NI instrument.

Your Turn

Find and install an instrument driver for a non-NI instrument that you might use back at your job, and explore its example programs.

End of Exercise 3-2

Exercise 4-1 Exploring an Existing Project and VI

Goal

- Explain the purpose of the main LabVIEW NXG project/VI development environment items.
- Differentiate between control and indicator.
- Differentiate between numeric, Boolean, and string data types.

Explore an existing project.

1. Open `Explore Existing Project.lvproject` in the `<Exercises>\LabVIEW Core 1\Explore Existing Project` directory.

2. Notice that the Project Files tab contains the following two items.

   ☐ Thermocouple.task

   ☐ Temperature Acquisition.gvi

3. Explore and configure the Thermocouple task item.

   ☐ Double-click the **Thermocouple.task** item.

   ☐ In the Thermocouple.task Channel tab, set Channel to a thermocouple channel on your DAQ device.

   – Set Channel to **cDAQ1Mod2/ai0**.

Explore and run the VI in the project.

1. Open the VI in the project.

   ☐ In the Project Files tab, double-click the **Temperature Acquisition.gvi** item.

2. Run the VI.

   ☐ Notice that Current Temperature displays the temperature.

   ☐ Run the VI a couple more times. Set the Max Temperature Threshold to values above and below the current temperature, and observe how this affects Threshold Exceeded and Message.

Explore the VI Panel

The Panel is the user interface of your VI.

1. Examine the VI Panel.

   ☐ Verify that Panel is selected on the VI toolbar.

2. Circle whether each of the following panel object is either a control (panel input) or indicator (panel output).

   ☐ Max Temperature Threshold Control | Indicator

   ☐ Current Temperature Control | Indicator

   ☐ Threshold Exceeded? Control | Indicator

   ☐ Message Control | Indicator

3. Circle whether each of the following panel objects is a numeric, Boolean, or string data type.

   ☐ Max Temperature Threshold Numeric | Boolean | String

   ☐ Current Temperature Numeric | Boolean | String

   ☐ Threshold Exceeded? Numeric | Boolean | String

   ☐ Message Numeric | Boolean | String

Explore the Diagram

The diagram defines the functionality of the VI and contains the code for the VI.

1. Click **Diagram** in the VI toolbar to display the diagram.

   ☐ On the diagram, identify the following items. How many can you find of each item?

   - Controls
   - Indicators
   - Constants
   - Free labels
   - Nodes

2. Use the Context Help to learn more about the items on the diagram.

   ☐ Press <Ctrl-H> to open the Context Help window or select **Help»Show Context Help**.

   ☐ Move the Context Help window to a convenient area where the window does not hide part of the diagram.

   ☐ Place your cursor over each of the different color wires to see which data type they represent.

   ☐ The Context Help window content changes to show information about the object that your mouse is over.

3. Get detailed help for the Select node.

1	**Select Node**—Place your cursor over the node and in the Context Help window. Click the More Help link to launch the LabVIEW NXG Help and learn more about this node.

Check Your Answers

1. Refer to the following figures to verify that you identified all items correctly.

Figure 4-3. Pane Items

1	Numeric Control	3	Boolean Indicator
2	Numeric Indicator	4	String Indicator

Figure 4-4. Diagram Items

1	Free Labels	4	Control Terminal
2	Nodes	5	Constants
3	Indicator Terminals		

On the Job

Describe the controls and indicators that you need in the VIs that you will create at your job. (name, data type)

End of Exercise 4-1

Exercise 4-2 Exploring Hardware Example Programs

Goal

Explore hardware example programs included in LabVIEW NXG.

Instructions

NI-DAQmx Example Programs

1. Explore NI-DAQmx examples.

1	From the LabVIEW NXG home screen, select the Learning tab.
2	Select the Examples tab.
3	Select Hardware Input and Output.

2. Select **NI-DAQmx** and then click the specific example you want to explore to launch an example NI-DAQmx project.

3. Explore the VIs in the example instrument driver project.

   ☐ Open and explore the VIs from the Project Files tab of the project.

   ☐ Explore the panel.

   ☐ Explore the diagram. Remember, you can read a description of a node by using the Context Help window <Ctrl-H>.

4. Close the project when finished. Do not save.

Non-NI Instrument Example Programs

1. Explore Non-NI Instrument Examples.

   ☐ From the LabVIEW NXG home screen, select the LEARNING tab and navigate to **Examples»Hardware Input and Output»Instrument Driver Examples**.

2. Select the instrument driver example that you want to explore to launch an example instrument driver project.

   If you are taking this course in a classroom with an NI Instrument Simulator, you can explore the NI Instrument Simulator instrument driver example.

3. Explore the VIs in the example instrument driver project.

   ☐ Open and explore the VIs from the Project Files tab of the project.

   ☐ Explore the panel.

   ☐ Explore the diagram. Remember, you can read a description of a node by using the Context Help window <Ctrl-H>.

4. Close the project when you are finished. Do not save the project.

End of Exercise 4-2

Exercise 5-1 Create a Simple Project and VI

Goal

Create a new project and new VI that executes the straight-line equation y = mx + b.

Instructions

1. Create a new project.

   ☐ In LabVIEW NXG, select **File»New»Project**.

2. Save the project.

   ☐ Select **File»Save All.**

   ☐ Set the Project Name field to `Simple VI`.

   ☐ Click the ... button next to the Location field, and browse to the `<Exercises>\LabVIEW NXG Core 1\Simple VI` directory.

   ☐ Click the **Save** button.

3. Create a new VI in your project

   ☐ In the Project Files tab, click the Add Items button and select **VI**.

   ☐ Rename the VI as `Straight-Line Equation.gvi`.

4. Explore the palette on the panel.

   Use palettes to locate items when you want to explore the options available to you or when you are not sure of the name of the control, indicator, or node you need.

   ☐ Select the Numerics, Buttons, and Text palette categories to explore various types of controls and indicators.

5. Create the user interface for your VI on the panel.

   ☐ Select **Panel** in the VI toolbar.

   ☐ Click the Search button on the palette.

   Tip Press <Ctrl-Space> to quickly access the search.

   ☐ Type `numeric control` in the search text box.

   ☐ Click **Numeric Control** in the search results and drag it to the panel to place the object. Do this three times to place three numeric controls on the panel. Rename them as `Slope (m)`, `Y-Intercept (b)`, and `x`.

   ☐ Press <Ctrl-Space> to Search the palette.

   ☐ Type `numeric indicator` in the search text box.

   ☐ Select Numeric Indicator in the search results and add it to the panel. Rename it as `y`.

☐ Arrange the items on the panel as shown in the following figure.

6. Develop the code for your VI on the diagram.

☐ Select **Diagram** in the VI toolbar.

☐ Click the Unplaced Items icon and drag the Slope (m), Y-Intercept (b), X, and Y items from the Unplaced Items tray to the diagram.

☐ Press <Ctrl-Space> and type `multiply` in the search text box. Select **Multiply** in the search results and add the Multiply node to the diagram.

☐ Add an **Add** node to the diagram.

☐ Wire the diagram as shown in the following figure.

7. Practice locating the palette of a node.

   ☐ Press <Ctrl-Space> and search for the term `Multiply`.

   ☐ In the search results, click the **Show In palette** icon next to a node to display the palette category that contains the node. This allows you to browse related nodes.

8. Practice using the global search feature.

   ☐ Type `Multiply` in the Search bar in the upper right hand corner of LabVIEW.

   Note As you type, the global search automatically looks for matches in the several places, including Online Help and LabVIEW palettes.

   ☐ Examine the search results.

9. Test the Straight-Line Equation VI using the values given in the following table.

   ☐ Enter the input values in the controls.

   ☐ Click the Run button.

☐ For each set of inputs, compare the given outputs to the outputs listed in the following table. If the VI works correctly, they should match.

Input		Output	
Slope (m)	2	y	7
x	1		
Y-Intercept (b)	5		
Slope (m)	2	y	15
x	5		
Y-Intercept	5		
Slope (m)	-3	y	-16
x	2		
Y-Intercept (b)	-10		

Your Turn

1. Create a new project and name it Average.lvproject.
2. Add a new VI named Average.gvi to the project.
3. Modify the VI to calculate the average of 5 numbers.
   (X1 + X2 + X3 + X4 + X5) / 5 = Average
4. Run the VI and verify that you get the correct results.

End of Exercise 5-1

Exercise 5-2 Create an Acquire-Analyze-Visualize VI

Goal

Create a simple VI that acquires data, analyzes data, and displays the results.

Create the Project

1. Create a new project.

   - ☐ In LabVIEW, select **File»New»Project**.
   - ☐ Select **File»Save All** and enter `Acquire Analyze Visualize` in the Project name field.
   - ☐ Click the ... button to set Location to the `<Exercises>\LabVIEW NXG Core 1\Acquire Analyze Visualize` directory.
   - ☐ Click the **Save** button.

Create a DAQmx Thermocouple Measurement Panel Task

1. Add a DAQmx thermocouple measurement panel task.

   - ☐ In the Project Files tab, add an **Analog Input** task.

   - ☐ In the Project Files tab, right-click the newly created Analog Input.task item and select **Rename**. Rename the item as `Thermocouple.task`.

2. Configure the measurement.

   ☐ In the Channel tab, configure the task as shown in the following figure.

3. Validate the signal is correct using measurement panel.

   ☐ Verify that the thermocouple measurement panel displays the expected temperature. For example, room temperature is usually around 25 degrees Celsius.

   ☐ Warm the thermocouple by rubbing the thermocouple with your fingers. Verify that the temperature rises on the measurement panel.

   ☐ Click the **Stop** button to stop the acquisition.

Acquire Data

1. Create a new VI in your project.

   ☐ In the Project Files tab, add a VI and name it `Acquire Analyze Visualize.gvi`.

2. Modify the diagram, as shown in the following figure, to acquire data.

1 DAQmx Start—Right-click the **task in** input and select **Create Constant**. Set the constant to Thermocouple.
2 DAQmx Read—In the Item tab, set the values shown in Table 5-1.
3 DAQmx Clear
4 **Current Temperature (deg C) Indicator**—Right-click the **data** output of the DAQmx Read node and select Create Indicator. Rename the indicator as `Current Temperature (deg C)`.

Note This constant populates with any DAQmx measurement panel tasks that you have created in this project.

Table 5-1. DAQmx Read Configuration Settings

Channel Type	Analog Input
Data Format	Floating Point
Channel Count	Single Channel
Sample Count	Single Sample

– Right-click the data output terminal and select **Create Indicator**. Rename the indicator as `Current Temperature (deg C)`.

3. Create the user interface for your VI.

☐ Go to the panel.

☐ Drag the Current Temperature (deg C) item from the Unplaced Items palette to the panel.

4. Run the VI. Your VI returns a single temperature acquired from the DAQ device.

Analyze and Visualize

1. Modify the diagram, as shown in the following figure, to analyze the acquired data.

1 **Greater?**—Add a Greater? node to the diagram.
2 **Input Terminal**—Right-click the **y** input of the Greater? node, select **Create Control** and rename the terminal `Maximum Temperature Threshold (deg C)`.
3 Boolean Output Terminal—Right-click the output of the Greater node and select **Create Indicator**. Rename the indicator `Threshold Exceeded?`
4 **Select**—Add a Select node to the diagram.
5 **String Constant**—Add a string constant to the diagram and wire it to the true case of the Select node. Enter `System in danger of overheating. Turn on the fan!`
6 **String Constant**—Right-click the false case input of the Select node and select **Crete Constant**. Enter `System is normal.`
7 **String Output Terminal**—Right-click the output of the Select node and select **Create Indicator** and rename it `Message`.

Tip Hold down the <Ctrl> key, hover over a wire, and click to create a wire branch.

2. Create the panel, as shown in the following figure, to visualize your results.

☐ Switch to the panel and drag the Maximum Threshold (deg C), Threshold Exceeded?, and Message controls/indicators from the Unplaced Items palette to the panel.

- ☐ Right-click the Maximum Threshold (deg C) control and select **Replace» Numerics»Horizontal Slider**. Resize the control as desired. In the Item tab for this control, set the following values, or whatever values are appropriate for our thermocouple sensor

Default	30
Minimum	15
Maximum	50
Interval	1
Coercion	To Interval

- ☐ Right-click the Threshold Exceeded? indicator and select **Replace»Buttons» Round LED**. Resize the indicator as desired.
- ☐ Resize the Message indicator to be larger.

3. Run the VI. Adjust the Maximum Threshold control to be above and below the Current Temperature, and compare the indicator results.

Your Turn

1. Add a new VI named `Celsius and Fahrenheit.gvi` to the project.
2. Modify the VI to acquire a temperature, display the temperature in degrees Celsius, convert the temperature to and display the temperature in degrees Fahrenheit. The VI should have the following indicators.
   - ☐ Current Temperature (deg C)
   - ☐ Current Temperature (deg F)

(Hint) Formula for converting deg Celsius to deg Fahrenheit:

$$degrees\ Fahrenheit = degrees\ Celsius \times \frac{9}{5} + 32$$

3. Run the VI. Use a calculator to verify that the conversion between Celsius and Fahrenheit is correct.

End of Exercise 5-2

Exercise 5-3 Measurement Panel Task vs Full DAQmx API

Goal

Use and compare the Measurement Panel Task and Full DAQmx API approaches of acquiring data from a DAQmx device.

Instructions

Approach 1: Use Measurement Panel Task

This approach helps you get started quickly if you have already created a measurement panel task.

1. Open `<Exercises>\LabVIEW NXG Core 1\Measurement Panel Task vs Full DAQmx API\Measurement Panel Task vs Full DAQmx API.lvproject`.

2. Examine the measurement panel task in the project.

   - ☐ Open the **My Analog Input Voltage.task** measurement panel from the Project Files tab.

   - ☐ Examine the settings in the Channel tab, and set Channel to **cDAQ1Mod7/ai0**.

☐ Examine the settings in the **Task** tab. Notice the Actual Sample Rate is set to 2.560 kHz.

Note Adjust the Desired Sample Rate to a rate that is supported by your DAQ device, if necessary.

3. Open **Measurement Panel Task Method.gvi** from the Project Files tab.

4. In the VI toolbar, select **Split** so you can see both the panel and diagram at the same time.

5. Examine the diagram.

   ☐ Notice the task constant, which references the My Analog Input Voltage task in the project. This task defines the channel and timing settings.

☐ Click the **DAQmx Read** node and view the Function Configuration section in the Item tab.

6. Examine the behavior of this VI.

☐ On the panel, set the Number of Samples control to `5120`.

Note Because the sample rate defined in My Analog Input Voltage.task is 2.560 kHz (samples per second) and the Number of Samples is set to 5120, the DAQmx Read will read 5120 samples representing 2 seconds of data.

☐ Run the VI.

☐ Notice that this VI makes a single acquisition and stops. The data graph indicator now shows 5120 samples. Notice that the Time x-axis shows that graph contains 2 seconds of data.

7. Close the VI.

Approach 2: Use Full DAQmx API

This approach allows all the configuration to be contained in the VI and provides more flexibility. This is the recommended approach for the remainder of this course.

1. Create a copy of the Measurement Panel Task Method VI.
   - ☐ In the Project Files tab, right-click **Measurement Panel Task Method.gvi** and select **Duplicate**.
   - ☐ Rename the duplicated item in the Project Files tab as `Full DAQmx API Method.gvi`.
   - ☐ Select **File»Save All**.
2. Open **Full DAQmx API Method.gvi** from the Project Files tab.
3. Delete the DAQmx task constant from the diagram.
4. Programmatically create and configure the DAQmx task, as shown in Figure 5-8

Figure 5-8.

5. Add a **DAQmx Create Virtual Channel** node to the diagram.

   ☐ When you add the DAQmx Create Virtual Channel node, select the configuration settings shown in the following figure.

Note These settings match the Measurement Type defined in the Channel tab of the Analog Input Voltage.task measurement panel.

   ☐ On the diagram, hover your cursor over the **physical channels** input. Right-click and select **Create control**.

   ☐ Right-click the **measurement range** input and select **Create control**.

6. Add a **DAQmx Timing** node to set the sample rate and sample mode.

   ☐ When you add the DAQmx Timing node, select the configuration settings shown in the following figure.

Note These settings match the Sample Timing Type defined in the Task tab of the Analog Input Voltage.task measurement panel.

   ☐ Right-click the **rate** input and select **Create control**. Rename it as `Desired Sample Rate (Hz)`.

   ☐ Right-click the **sample mode** input and select **Create Constant**. Set the constant to **Finite Samples** because this VI will only read a finite number of samples.

   ☐ Wire the **Number of Samples** control to the **samples per channel** input of the DAQmx Timing node.

   In finite acquisition mode, the samples per channel input tells the DAQmx task how many finite samples the DAQmx task should acquire.

Note Hold down the <Ctrl> key, hover over a wire, and click to create a wire branch.

7. Add a **Task Properties** node to get the actual sample rate.

1 **Timing**—Click the top pull-down menu and select the **Timing** category.
2 **Sample Clock Rate**—In the lower pull-down menu, select **Sample Clock»Rate**.
3 **Actual Sample Rate (Hz)**—Right-click the SampClk.Rate output and select **Create indicator**. Rename the indicator as `Actual Sample Rate (Hz)`.

8. Complete the wiring as shown in Figure 5-8.

9. Switch to the panel and drag the controls from the Unplaced Items tray to the panel, as shown in the following figure.

10. Set the Number of Samples control to `5120`.

11. Set the values of the added controls to match the values in the Analog Input Voltage measurement panel.

12. You can find the values for the measurement panel in the locations shown in Table 5-2

Table 5-2. Control Values

Control Name	Measurement Panel Control Value Location
Physical channels	**Channel tab»Channel**
Desired Sample Rate (Hz)	**Timing tab»Actual Sample Rate** **Note** Enter this value into the control in units of Hz, not kHz.
Maximum value	**Channel tab»Maximum**
Minimum value	**Channel tab»Minimum**
Input terminal configuration	**Channel tab»Terminal Configuration**

13. Run the VI.

14. Because you are using the DAQmx Create Channel and DAQmx Timing nodes, you can modify the physical channel and sample rate directly from your VI. This VI is no longer directly connected to the Analog Input Voltage.task measurement panel. All the configuration for this DAQmx task is contained in this VI.

    - ☐ Change the Desired Sample Rate (Hz) control value from `2560` to `5120` to update the sample rate.
    - ☐ Run the VI, and notice that the Time x-axis now shows that only 1 second of data was acquired.
    - ☐ Change the Desired Sample Rate (Hz) control value to 3333. Run the VI, and notice that the Actual Sample Rate indicator value is different from the Desired Sample Rate.

    Note It is important for you to check the actual sample rate that the DAQ device used to acquire your data. Do not assume that actual sample rate is the same as the desired sample rate you entered. The supported sample rates vary depending on your DAQ device or module.

    - ☐ Change the PhysicalChannel control to a different channel, such as cDAQ1Mod7/ai2).
    - ☐ Run the VI, and notice that you are now acquiring data from a different channel and sensor.

15. Select **File»Save All**.

16. Close the project when finished.

On the Job

If your application will use NI-DAQmx, which approach will you use? (Circle one)

`Measurement Panel Task Approach` OR `Full DAQmx API Approach`

End of Exercise 5-3

Exercise 6-1 Debugging

Goal

Use debugging tools and troubleshooting techniques to fix a broken VI that returns unexpected data.

Instructions

Edit-Time Errors

Locate and correct errors that prevent the VI from running.

1. Open and examine the Analyze Temperature VI.

   - ☐ Open `Debug.lvproject` in the `<Exercises>\LabVIEW NXG Core 1\Debug` directory.
   - ☐ Open **Analyze Temperature (broken Run button).gvi** from the Project Files tab in the Navigation Pane.
   - ☐ Notice the Run button on the toolbar appears broken, indicating that the VI is broken and cannot run.

2. Examine the diagram of the Analyze Temperature VI, as shown in the following figure.

   This VI acquires a single temperature measurement, displays the temperature in degrees Celsius, converts the temperature to degrees Fahrenheit, and determines if the acquired temperature is within the user-specified range.

3. Find and fix each error.

   ☐ Click the broken **Run** button to display the **Errors and Warnings** tab. Examine the errors and warnings listed.

1	Severity—Denotes whether an issue is an error or a warning.
2	Source—Identifies the object that is causing the error or warning.
3	Message—Provides more detail about why the error or warning exists.

   ☐ Double-click each error/warning to highlight the area on the diagram that contains the error.

   ☐ Use the information in the **Error and Warnings** tab to fix each error.

   ☐ Notice that after you have fixed every error, the Run button no longer appears broken. You can now run the VI.

4. Save the VI.

Run-Time Errors

Identify and correct errors that cause the VI to behave unexpectedly and return incorrect responses.

When the results of your application are not what you expect, you can use a set of tools to determine where errors occur within your code.

Although errors are often detected automatically, sometimes your code can run successfully but not as intended. When this happens, you need to identify the source of the unintended behaviors.

The following debugging tools can help you in this process:

- Execution highlighting
- Probes
- Breakpoints
- Single-stepping

1. Test the VI

   ☐ From the Project Files tab, open **Analyze Temperature (incorrect behavior).gvi**.

   ☐ Switch to the panel and run the VI.

   The DAQmx Read node outputs the Temperature in degrees Celsius to the corresponding indicator.

   ☐ Use a calculator to determine what the correct Temperature in degrees Fahrenheit should be:

   Temperature (deg F) = Temperature (deg C) * 9/5 + 32
   = _____ * 9/5 + 32
   = _____

   Does the Temperature (deg F) indicator value match your calculation? _____

   ☐ Notice that even though the VI runs, the VI returns an incorrect result in the Temperature (deg F) indicator. In the following steps, you will identify the source of this error using the debugging tools and correct the error.

2. Animate the flow of data through the diagram.

   ☐ Switch to the diagram and click the **Highlight Execution** button to enable execution highlighting and then run the VI.

   ☐ Notice that you can see how data flows through the wires. At the output of each node, you can see the data value displays momentarily.

3. Probe the wire values.

   ☐ Add probes to the wires, as shown in the following figure, by right-clicking each wire and selecting **Set Probe**.

   ☐ Notice that after setting a probe, the Navigation pane now shows the Debugging tab, which shows an item for each probe.

   ☐ Run the VI.

☐ Notice in the **Debugging** tab, the Probes section has updated with latest data values on the corresponding wires. The row for ID 1 shows the data value for the wire labeled with Probe 1.

☐ Double-click any Probe item in the **Debugging** tab to highlight the corresponding Probe on the VI diagram.

☐ You can view the latest data values of a probe by looking at the values in the **Debugging** tab or hovering your cursor on the probe in the VI diagram.

☐ By examining these probes, you determine that Probe 1 shows the correct temperature in Celsius and Probe 4 shows an incorrect value for temperature in degrees Fahrenheit. This means the mistake in this VI must occur in the nodes between Probe 1 and 4.

What is the mistake?

Answer: A quick examination of the values obtained from those probes will reveal that the algorithm used to convert from °C to °F is incorrect.

☐ Remove all the probes by right-clicking each probe on the diagram and selecting **Delete Probe**.

Note You can also remove probes by selecting a probe in the **Debugging** tab and pressing the <Delete> key.

4. Debug the VI using the single-stepping tool.

   With execution highlighting, execution slows down, and the code executes until completion. With single-stepping, you can execute a single node at a time, causing the program to pause after the node completes.

   - ☐ In the **Debugging** tab, click the **Step Over** button to start single-stepping through the VI. If Highlight Execution is still turned on, execution highlighting shows the flow of data on the diagram from one node to another. When you single-step through code, nodes are highlighted to indicate they are ready to execute.

   - ☐ Click the **Step Over** button after each node to step through the entire diagram. Each time you click the **Step Over** button, the current node executes and pauses at the next node.

   - ☐ When you step over the last node in the diagram, gray lines appear in the diagram to indicate that the all the nodes in the diagram have finished executing. Click the **Step Over** button one more time to finish running the VI.

5. Set a breakpoint to pause the VI when its execution reaches a specified point in the program.

   When looking for a problem in your code, you may have an idea of the general area where the problem exists. To help focus on this area, you can use a breakpoint to pause the VI at a specified point in the program.

   - ☐ Imagine that you have determined that the problem in this VI occurs right after the Add node executes.

   - ☐ Right-click the Add node and select **Set Breakpoint**.

   - ☐ Notice that now there is a numbered red dot on the Add node on the diagram. Notice that the numbered red dot corresponds to a breakpoint item in the **Debugging** tab.

- ☐ Verify that Highlight Execution is turned off and run the VI.

  Notice that the VI pauses when it reaches the breakpoint.

- ☐ Now that the VI has reached the area you want to examine, you can start using your debugging tools. For example, you can turn on Highlight Execution and click on the **Step Over** button to start single-stepping through the VI.

- ☐ When finished, remove the breakpoint by right-clicking the breakpoint on the diagram and selecting **Delete Breakpoint**.

Note If you select **Disable Breakpoint** instead, the breakpoint will remain on the diagram but it will no longer pause execution. Then, you can right-click and enable the breakpoint again later when you want the breakpoint to start pausing execution again.

6. Save the VI.

End of Exercise 6-1

Exercise 7-1 Introduction to While Loops

Goal

Use While Loop to execute code repeatedly and explain how its components (iteration terminal, Condition terminal) work.

Guided Instruction

Explore the behavior of a VI without a While Loop.

1. Explore the While Loop Basics VI.

   - ☐ Open `<Exercises>\LabVIEW NXG Core 1\While Loop Introduction\While Loop Introduction.lvproject`.
   - ☐ From the Project Files tab, open **While Loop Basics.gvi**.
   - ☐ Select **Split** to display both the panel and the diagram at the same time.

2. Explore the panel and diagram.

   - ☐ Run the VI several times with different values in the Total Time in Seconds control. (e.g. 10, 70, 3750).

     Notice that each time you click the Run button, the VI calculates the hours, minutes, and seconds and then stops. What if you wanted to click the Run button once and have the VI continuously perform this calculation until the user clicks a Stop button?

Execute Code Repeatedly

1. Place a While Loop around the code you want to execute repeatedly.

   ☐ Select **While Loop** from the **Program Flow** palette category.

   ☐ Click and drag your mouse around the code performing the calculation and refer to the following figure to complete the code.

1	Iteration terminal—outputs the current iteration number of the While Loop. The first iteration of a loop is 0.
2	Current Iteration Indicator—Right-click the Iteration terminal and select **Create»Indicator**. Rename the indicator `Current Iteration`.
3	Condition terminal—Right-click the terminal and select **Create»Control** to create a Stop button. When a True value is passed to the condition terminal, the While Loop will exit.

2. Update the panel.

   ☐ Switch to the Panel.

   ☐ Click the Unplaced Items and drag the Stop control and Current Iterations indicator onto the Panel.

Your panel should resemble the following figure.

3. Increase the Significant Digits on the Current Iteration indicator so the iteration number displays correctly.

   ☐ Select the Current Iteration indicator.

   ☐ On the Item tab, under Display Format, set the value to `10` **Significant Digits**.

4. Explore the functionality of the edited VI.

   ☐ Run the VI.

   ☐ Change the value of the Total Time in Seconds control. Notice that the VI continuously runs the code in the While Loop, so the Hours, Minutes, and Seconds indicators are continuously updated.

   ☐ Notice that the Current Iteration indicator is incrementing.

   ☐ Click the Stop button to stop the VI.

5. Explore the functionality of the VI further using Highlight Execution.

   ☐ Select Split to display the panel and the diagram at the same time.

   ☐ Turn on Highlight Execution.

   ☐ Run the VI.

   ☐ On the diagram, notice that the Current Iteration terminal outputs a **`0`** on the first iteration and increases by 1 for each subsequent iteration.

   ☐ Click the **Stop** button on the panel. On the diagram, notice that the Stop control passes a **True** value to the Condition terminal, which causes the While Loop to exit.

6. Turn off Highlight Execution and save the VI.

Explore the Functionality of While Loop Output Tunnels

1. Count the total number of iterations.

1	Numeric Indicator—Add a numeric terminal to the diagram and select **Indicator** in the Item tab.
2	Increment node—This adds 1 to the value output of the Iteration terminal to display the correct number of iterations, because the first iteration of the While Loop is 0. Rename the indicator `Total Iterations`.
3	Iteration Output—Wire the output of the Iteration terminal through a tunnel on the While Loop to the Increment node.

2. Update the panel.

   ☐ Drag the Total Iterations indicator from the Unplaced Items gallery to the panel and arrange it as shown in the following figure.

   ☐ Select the Total Iterations indicator and in the Item tab, configure the display format to `10` **Significant Digits**.

3. Explore functionality of the While Loop output tunnel.

   ☐ Run the VI.

   Notice that the Total Iterations indicator is not updating.

   ☐ Click the **Stop** button. Notice that the Total Iterations indicator now updates.

- ☐ A While Loop behaves like a node in the LabVIEW dataflow model. The While Loop does not execute until all of its inputs, which are input tunnels, are available. A While Loop does not output anything through an output tunnel until it finishes executing, when it finishes its last iteration due to an input to the Conditional terminal.

- ☐ Turn on Highlight Execution and run the VI again.

  Observe the flow of data on the diagram at the output tunnel when you click the Run button. Observe the flow of data when you click the **Stop** button.

4. Save the VI.

Explore the Functionality of While Loop Input Tunnels

1. Create an input tunnel.

   - ☐ Drag the **Total Time in Seconds** control out of the While Loop, so that your diagram looks like the following figure.

2. Explore the functionality of the While Loop input tunnel.

   - ☐ Select **Split** to display the diagram and the panel at the same time.

   - ☐ On the panel, set the Total Time in Seconds control to 65.

   - ☐ Turn on Highlight Execution, so you can watch the dataflow for this diagram.

   - ☐ Run the VI.

     Notice that because of dataflow rules, the While Loop does not execute until the input tunnel has received data.

     Notice that for each iteration of the While Loop, the input tunnel outputs a value of 65.

   - ☐ If you change the value of the Total Time in Seconds control to 80 while the While Loop is still running, what value do you think the input tunnel will output? _____

- ☐ On the panel, change the value of the Total Time in Seconds control to 80.

  Notice that the input tunnel still outputs a value of 65.

  Because of the rules of dataflow, the While Loop must wait until all input tunnels receive data before the While Loop can begin executing. When the While Loop begins executing, the input tunnels will only output the original data they received. The While Loop input tunnels only read data once (input tunnels do not continuously read data).

3. Move the Total Time in Seconds control back into the While Loop, as shown in the following figure, so that the While Loop reads the current value of the control during every iteration.

4. Save the VI and project.

On the Job

In your applications, do you have code that you need to place in a While Loop?

If so, describe the code or task that you need to execute repeatedly in a While Loop.

End of Exercise 7-1

Exercise 7-2 Using Timing Nodes in a Loop

Goal

Use timing nodes in a loop to set the loop period, decrease CPU usage, and view the current date/time and elapsed time.

Guided Instruction

Set Loop Period

In this part of the exercise r your While Loop.

1. Explore the Set Loop Period VI.

   ☐ Open `<Exercises>\LabVIEW NXG Core 1\Using Timing Nodes in a Loop\Using Timing Nodes in a Loop.lvproject`.

   ☐ From the Project Files tab, open **Set Loop Period.gvi**.

2. Run the VI.

   Notice that the loop runs as fast as possible.

3. Modify the diagram, as shown in the following figure, to set the loop to execute every 1,000 ms.

1 **Wait Until Next Multiple**—Set the units to Milliseconds when you add this node to the diagram.
2 **Millisecond Multiple**—Right-click the input and select **Create Constant**. Set the Constant to `1000`.

Note You have configured this node to wait until the OS timer reaches a multiple of 1,000 before this node completes execution. As a result, this loop executes one iteration every 1,000 milliseconds.

Note that in the first iteration, the Wait Until Next Multiple node will probably wait less 1,000 milliseconds. For example, if the OS timer is at 5800 when the first iteration starts, then the Wait Until Next Multiple node will only wait 200ms on the first iteration because the next multiple of 1,000 is 6,000. However, the Wait Until Next Multiple node should wait 1,000 ms on the subsequent iterations because the next multiple will be 7,000, 8,000, 9,000, 10,000, and so on.

4. Observe the behavior of the VI.

- ☐ Select **Split** to display the diagram and panel at the same time.
- ☐ Run the VI.

  Notice that the While Loop executes one iteration every 1,000 milliseconds after the first iteration completes.
- ☐ Stop the VI.
- ☐ On the diagram, change the millisecond multiple constant to `500`.
- ☐ Run the VI again and notice that the While Loop now executes one iteration every 500 ms.
- ☐ Stop the VI.

5. Experiment running the VI with different values in the Wait Until Next Multiple constant.

   ☐ Try to modify the VI, so that the While Loop executes one iteration every 2 seconds.

6. Save and close the VI when you are finished.

Allow the CPU Time to Complete Other Tasks

In this section, you want to ensure that your loop allows the CPU time to complete other tasks. In other words, you want to make sure that your VI does not needlessly have a high CPU usage.

1. Explore the CPU Usage of a Loop VI.

   ☐ From the Project Files tab, open **CPU Usage of Loop.gvi**.

   ☐ Examine the diagram.

   Notice that there is no Wait node in the loop, which means the loop will run as fast as possible.

2. Observe the behavior of the VI.

   ☐ Run the VI.

   ☐ Change the values of the Slope (m), x, and Y-Intercept (b) controls. Notice that the y indicator continuously updates because the While Loop continuously performs the calculation.

3. Examine the CPU usage.

   ☐ Launch the Windows Task Manager.

   ☐ In Process tab, observe the CPU usage of LabVIEW.exe. Because the While Loop is running as fast as possible, LabVIEW will have a higher CPU usage.

   CPU Usage of LabVIEW.exe: _____

   Note In Windows 7, right-click the Windows taskbar and select **Start Task Manager**.

4. Stop the VI when finished.

5. Modify the diagram, as shown in the following figure, to add a Wait node to lower the CPU usage of this VI.

Note Adding even a small amount of time to wait will decrease the CPU usage significantly.

6. Run the VI.
7. In the Windows Task Manager, notice that the CPU usage of LabVIEW.exe is now much lower.
8. Stop the VI.
9. Save and close the VI when finished.

View Current Date/Time and Elapsed Time

In this section, you explore the nodes that return the current date/time and elapsed time.

1. From the Project Files tab, open **Using Timestamp and Elapsed Timer Nodes.gvi** and examine the diagram.
   - ☐ Notice that the While Loop continuously performs a slope calculation.
   - ☐ The Get Current Time node returns the current date and time. This node does not perform any waiting.
   - ☐ The Elapsed Timer node returns how much time has elapsed since the first time the node executed in the loop. This node does not perform any waiting. When the elapsed time is greater than or equal to the target time, then the Elapsed Timer node will output a True from the “Target Time Has Elapsed” terminal. Since this output is wired to the Condition terminal of the While Loop, the While Loop will exit after the target time has elapsed.
   - ☐ Use the Context Help window (press <Ctrl-H>) to find more information on how the Get Current Time and Elapsed Timer nodes work.
   - ☐ Because none of the nodes in this While Loop perform any waiting, how many iterations per second does this While Loop execute? _____

Answer—The While Loop executes its iterations as fast as possible because the While Loop does not contain a node that performs any waiting.

2. Examine the behavior of the Using Timestamp and Elapsed Timer Nodes VI.

   ☐ On the panel, set the Target Time (s) control to 15.

   ☐ Run the VI and notice that the Elapsed Time (s) indicator continuously displays how much time has elapsed since the While Loop started.

   ☐ Notice that the Current Time indicator continuously displays the current date and time.

   ☐ Notice that the While Loop is executing its iterations as fast as possible.

   ☐ Notice that the While Loop exits after 15 seconds has elapsed.

   ☐ In the Windows Task Manager, observe the high CPU usage of the LabVIEW.exe process while this VI is running and record the value.

   _____

3. Reduce the CPU usage of this VI.

   This VI does not have to run the While Loop iterations as fast as possible.

   ☐ How can you modify this VI to reduce the CPU usage?

   ☐ Use your ideas to modify the VI to reduce the CPU usage.

   Did your modification work? What is the CPU usage of the LabVIEW.exe process now?

   _____

   See the next page for the solution.

Answer

To reduce the CPU usage of the Using Timestamp and Elapsed Timer Nodes VI, you can add a Wait node inside the While Loop, as shown in the following figure. Even adding a small amount of time to wait will decrease the CPU usage significantly.

End of Exercise 7-2

Exercise 7-3 While Loop using DAQmx API Timing

Goal

Create a VI that continuously acquires data from a DAQmx device.

Predict the DAQmx Read wait time and While Loop frequency based on the Sample Rate and Number of Samples inputs.

Guided Instruction

Examine Finite Acquisition VI

1. Explore the Finite Acquisition VI.

   - ☐ Open `<Exercises>\LabVIEW NXG Core 1\While Loop with Hardware API\While Loop with Hardware Timing.lvproject`.
   - ☐ Open **Finite Acquisition.gvi** from the Project Files tab.
   - ☐ Notice the controls and graph indicator on the panel.
   - ☐ Switch to the diagram and notice that the code performs a finite acquisition.
   - ☐ Click the following three nodes and examine the configuration of each one in the Item tab.

1	**DAQmx Create Channel**—Notice the Channel Type and Measurement Type.
2	**DAQmx Timing**—This node sets the sample rate of the acquisition. The sample mode must be set to **Finite Samples** for a finite acquisition.
3	**DAQmx Read**—This node reads data from the DAQ device. Notice that Read By is set to **Samples**. As a result, the DAQmx Read node has a number of samples per channel input, which tells this node to wait until ***n*** samples of acquisition data is available to read from the DAQ device.

2. Examine the functionality of the Finite Acquisition VI.

   - ☐ On the panel set the following control values:
     - **Physical Channel:** cDAQ1Mod8/ai0
     - **Desired Sample Rate (Hz):** 1,000
     - **Number of Samples:** 1,000

- ☐ Run the VI and notice that this VI performs a single acquisition and is finished. The DAQmx Read waits until 1,000 samples of data is available to read from the DAQ device. Then the DAQmx Read outputs 1,000 samples of data to the data graph indicator.
- ☐ Set the Number of Samples to 5,000, and run the VI again.

  Notice that the DAQmx Read must now wait until 5,000 samples of data is available to read from the DAQ device.

Note When the sample rate is 1,000 Hz (1,000 samples per second), 5,000 samples of data represents 5 seconds of data.

3. Close the VI when you are finished.

Create Continuous Acquisition VI

1. Create a copy of the Finite Acquisition VI.
   - ☐ In the Project Files tab, right-click **Finite Acquisition.gvi** and select **Duplicate**.
   - ☐ Rename the duplicated item `Continuous Acquisition.gvi` in the Project Files tab.
   - ☐ Select **File»Save All**.
2. Open **Continuous Acquisition.gvi** from the Project Files tab.

3. Modify this VI, as shown in the following figure, to acquire data continuously until the user clicks the Stop button.

1 **sample mode** constant—Set this constant to Continuous Samples so you can continuously read acquisition data in the While Loop.
2 **While Loop**—Add the While Loop around the DAQmx Read node. Because the Number of Samples control is outside the While Loop, the DAQmx Read node will only see the first value of the Number of Samples control when the VI begins running. This is the desired behavior for this VI because we intend for the user to set all control values before running the VI and not modify values after the VI begins running.
3 **Stop** terminal—Right-click the Condition terminal and select Create Control. Rename the control `Stop`.—
4 **OR**—Add an OR node between the Stop terminal and Condition terminal and wire the error wire to input 0. This stops the While Loop when the user clicks the Stop button or when an error occurs.

☐ Switch to the panel and drag the Stop control from the Unplaced Items tray to the panel.

4. Examine how the Desired Sample Rate (Hz) and Number of Samples values affect how long the DAQmx Read node waits and how often the graph indicator updates.

   ☐ On the Panel, set the following control values:

   – **Physical Channel:** cDAQ1Mod8/ai0

   – **Desired Sample Rate (Hz):** 1,000

   – **Number of Samples:** 1,000

   ☐ Run the VI.

   ☐ Notice that the VI now continuously acquires and displays data until you click the Stop button.

   ☐ Notice that the data graph updates about once per second.

   This is because the DAQmx Read node is configured to read by samples. At a sample rate of 1,000 Hz, the DAQ device acquires 1000 samples per second. Therefore for each iteration of the While Loop, the DAQmx Read node must wait approximately 1 second for 1,000 samples to be available to read and then outputs that data.

   ☐ Launch the Windows Task Manager and go to the Performance tab. Notice that the CPU is not at 100% because the DAQmx Read waits, which allows the CPU time to execute other tasks.

   Note In Windows 7, right-click the Windows taskbar and select **Start Task Manager** to launch the Task Manager.

   ☐ Click the Stop button to stop the VI.

5. Try to predict what will happen in the following scenarios.

   ☐ Sample Rate: 1,000
   Number of Samples: 500
   How often will the graph on the panel update? _____

   ☐ Sample Rate: 1,000
   Number of Samples: 2,000
   How often will the graph on the panel update? _____

6. Test the sample rate and number of samples combinations. Record the results, and compare the results with your predictions in the previous step. You must stop the VI between each test because the Number of Samples control is outside the While Loop.

   ☐ Sample rate: 1,000
   Number of Samples: 500
   How often did the graph on the panel update? _____

   ☐ Sample rate: 1,000
   Number of Samples: 2,,000
   How often did the graph on the panel update? _____

Note At a sample rate of 1,000 Hz, the DAQ device acquires 1,000 samples per second. Therefore, the DAQmx Read node must wait approximately 0.5 seconds for 500 samples and 2 seconds for 2,000 samples.

7. Save the VI and close the project.

(Optional) Explore Instrument Control Example VI

1. Go to the LabVIEW NXG launcher. If you have a project open, click the following icon in the toolbar to go to the LabVIEW NXG home screen:

2. Click the **Learning** tab.
3. Click **Examples**.
4. Click **Instrument Driver Examples**.
5. Click the **Keysight 344XX Series** instrument driver example, if installed, to launch the project.
6. From the Project Files tab, open **Keysight 344XX Acquire and Chart - External Triggered.gvi**.
7. Explore the diagram.
   - ☐ Notice that the timing is done by the hardware, which acquires samples based on external triggers. Therefore, no Wait node is necessary in the loop.
8. Close the VI and project when finished.

End of Exercise 7-3

Exercise 7-4 Using Shift Registers

Goal

Use a While Loop and a shift register to use previous iteration data to keep track of the maximum acquired value.

Instructions

1. Open `<Exercises>\LabVIEW NXG Core 1\Using Shift Registers\Using Shift Register.lvproject`.
2. From the Project Files tab, open **Track the Maximum Temperature.gvi**.
3. Explore and run VI.
   - ☐ On the diagram, modify the minimum value, maximum value, and thermocouple type constant values to match your thermocouple.
   - ☐ On the panel, set the Thermocouple Channel control to a channel attached to thermocouple.
   - ☐ Run the VI.
   - ☐ Notice that this VI performs a continuous acquisition at a rate of 10 Hz. The DAQmx Read reads one temperature value per loop iteration.
   - ☐ Click the **Stop** button on the panel when finished.
4. Add a shift register to the While Loop. This shift register will store the value of the maximum acquired value.
   - ☐ Right-click the border of the While Loop and select **Create Shift Register**.

5. Modify the diagram, as shown in the following figure.

1 **Negative Infinity constant**—Wire this constant to initialize the shift register. This ensures that the DAQmx Read output in the first iteration will become the new maximum value stored in the shift register (because any number is greater than negative infinity).
2 **Max and Min node**—Use this node to compare the maximum acquired value with the current value and output the higher value of the two.
3 **Max Temperature Acquired indicator**—Right-click the output of the **Max and Min** node and select **Create»Indicator**. Rename the indicator as `Max Temperature Acquired (deg C)`. Go the panel, and drag this indicator from the Unplaced Items tray to the panel.

6. Test the VI, and observe the shift register logic.

   ☐ Turn on Highlight Execution.

   ☐ Run the VI. Touch the thermocouple to make the temperature increase and decrease, and observe how the logic of the shift register, Max & Min, and DAQmx Read work together to keep track of the maximum temperature acquired.

   The Max Temperature Acquired indicator should show the maximum temperature acquired over all While Loop iterations.

   ☐ Turn off Highlight Execution to see the VI run at regular speed.

Your Turn

1. From the Project Files tab, open **Threshold Exceeded.gvi**.

2. Run and explore the VI. The Threshold Currently Exceeded? Indicator turns on if the DAQmx Read output exceeds the Max Threshold control value.

3. Add a Boolean indicator named `Threshold Ever Exceeded?` to the panel.

4. Modify the VI so that the Threshold Ever Exceeded? Boolean indicator turns on and stays on if the DAQmx Read output ever exceeds the Max Threshold control value. (Hint: Use a shift register that stores the data if the max threshold has ever been exceeded)

Answer:
You can compare your answer with the solution in `<Solutions>\LabVIEW NXG Core 1\Exercise 7-4` directory.

On the Job

In your own application(s), do you have a need to use a shift register to keep track of a value in the previous iteration?

End of Exercise 7-4

Exercise 7-5 (Self-Study) Stacked Shift Registers

Goal

Use a stacked shift register to store and access values from multiple previous iterations.

Guided Instruction

1. Open `<Exercises>\LabVIEW NXG Core 1\Stacked Shift Registers\Using Stacked Shift Registers.lvproject`.

2. From the Project Files tab, open **Running Average Using Stacked Shift Register.gvi**.

3. Explore the stacked shift register on the diagram.

   ☐ Notice the stacked shift register.

   ☐ The first (top) left shift register outputs data from the previous iteration.

   ☐ The second left shift register outputs data from the second-most recent iteration. If there was a third left shift register, it would output data from the third most recent iteration.

4. Examine the functionality of the diagram.

   ☐ The purpose of the stacked shift register in this VI is to help calculate the average of the three-most recent temperature measurements. The DAQmx Read in the While Loop outputs the current measurement, the first (top) left shift register outputs the previous measurement, and the second left shift register outputs the measurement before the previous measurement.

   ☐ The Average Temperature Chart displays the running average of the three most recent temperature measurements.

   ☐ Notice that this VI reads one temperature measurement before the While Loop and uses this initial temperature measurement to initialize each stacked shift register.

5. Test the VI.

   ☐ Run the VI.

   ☐ Compare the Average Temperature Chart and the Temperature Chart. The Average Temperature Chart should be smoother due to the averaging.

   ☐ Turn on Highlight Execution to examine how the stacked shift register stores and outputs data.

   ☐ Click the Stop button to stop the VI when finished.

6. Resize the stacked shift register to access data from even earlier loop iterations.

   - ☐ Hover your mouse cursor over the bottom border of the last left shift register.
   - ☐ When the mouse cursor changes to the resize cursor, click and drag the shift register downwards to add more elements shift register to access data from even earlier loop iterations.

     Alternatively, you can right-click the stacked shift register and select **Add Last Element**.

Your Turn

Modify the VI so that the Average Temperature Chart displays the running average of the five most recent temperature measurements.

To view the answer, open the project in `<Solutions>\LabVIEW NXG Core 1\Exercise 7-5 (Your Turn)`.

End of Exercise 7-5

Exercise 8-1 Creating and Viewing an Array

Goal

Create and view a numeric array data type.

Instructions

1. Create a new project. Save the project as `Create Array.lvproject` in the `<Exercises>\LabVIEW NXG Core 1\Create Array` directory.

2. Create and open a new VI. Rename the VI as `Create Array.gvi`.

3. Create an array of numeric controls.

   ☐ Add an **Unconfigured Array** to the panel. Rename it as `My Numeric Array`.

   ☐ Add a **Numeric Control** inside the unconfigured array.

4. Resize the array to show 6 elements.

   ☐ Select the entire array control. To do this, hover your cursor near the border of the array until a blue line outlines the array. Then click your mouse button. The array should now have a darker blue outline and three dots outside of the outline, as shown in the following figure.

   ☐ To resize the array horizontally, drag the dot on the far right until 6 elements appear on the panel.

   Notice that the six elements appear grayed out. This is because these elements are uninitialized and contain no actual data.

5. Enter data into the array control. Enter `5`, `10`, `15`, `20`, `25` into the array, as shown below. Notice that in this example, the last element is still grayed out and uninitialized.

6. Check your understanding of the array index.

   ☐ What is the value of the array element at index 2? _____

   ☐ What is the value of the array element at index 4? _____

   ☐ What is the value of the array element at index 0? _____

   ☐ **Answers:** 15, 25, 5

   Remember, arrays are zero-indexed, which means the very first element of the array is at index 0.

7. Examine the array index display.

1	**Index display**—Controls which array element is shown at the leftmost element. Use the increment and decrement buttons to change the value in the index display.

If you set the index display to 2, then the leftmost element in the array will show the element at index 2, and the element to its right will show the element at index 3.

   ☐ Increase and decrease the index display to examine how it affects which elements are shown in the array control.

8. View how the array data appears on a graph indicator.

   ☐ Place a Graph indicator on the panel.

   ☐ Create the diagram, as shown in the following figure, to pass the array data to the graph.

   ☐ Switch to the panel and run the VI. Notice how the 1D array data is visualized in the Graph indicator.

☐ Click the Plot legend of the Graph indicator. In the Item tab, set the Plot to **Line & Point**. The Graph indicator now displays points that represent the data values in the array control. The lines connect the points sequentially.

– On the y-axis of the Graph, double-click Amplitude and rename it as **`Data Value`**.

– On the x-axis of the Graph, double-click Time and rename it as `Index`.

Notice that the My Numeric Array control contains data values, but does not contain any timing information. Therefore, the x-axis only represents the array index number for each array data value. This is why you see a point at x-axis values of 0, 1, 2, 3, and 4.

Create a Two-Dimensional (2D) Array

1. Change the number of dimensions from 1 to 2.

   ☐ Select the array control.

   ☐ In the Item tab, set Dimensions to 2.

2. View two dimensions (rows and columns) of data.

   ☐ Select the entire array control. To do this, hover your cursor near the border of the array until a blue line outlines the array. Then click your mouse button. The array should now have a darker blue outline and 3 dots outside of the outline.

   ☐ Click the middle-bottom dot and drag it downwards until 4 rows are displayed on the panel.

3. Enter data into the 2D array, as shown in the following figure.

4. Examine the index display.

   ☐ Notice that there are now two numeric controls in the Index Display.

   ☐ The top numeric control refers to the row index. The bottom numeric control refers to the column index.

   ☐ Increase and decrease the index display controls to examine how it affects which elements are shown in the array control.

   ☐ If you wanted to the upper leftmost array element to show the element containing 12, what should the row index and column index be?

Row Index	
Column Index	

   ☐ **Answer**: row index is 1; column index is 3

5. View how the 2D array data appears on a graph indicator.

   ☐ Switch to the panel and run the VI. Observe how the 2D array data is visualized in the Graph indicator.

   ☐ Hover your cursor over the Plot legend, and resize it to show three plots.

   ☐ Click each plot in the Plot legend of the Graph indicator, and set the plots to **Line & Point** in the Item tab.The Graph indicator now displays points that represent the data values in the array control. The lines connect the points sequentially.

Configure the Appearance of an Array Control

1. Select the My Numeric Array control.

2. In the Layout section of the Item tab, you can configure whether to show the Index Display, Horizontal Scrollbar, and Vertical Scrollbar.

☐ Check and uncheck these items to observe their effect on the appearance of the My Numeric Array control.

End of Exercise 8-1

Exercise 8-2 Examine the Waveform Data Type

Goal

Recognize the components and visualization of the waveform data type.

Instructions

1. Open `<Exercises>\LabVIEW NXG Core 1\Waveform Data Type\Waveform Data Type.lvproject`.
2. Open **Examine Waveform Data Type.gvi** from the Project Files tab.
3. Examine the VI.
   - ☐ Examine the diagram. Notice that the Waveform Control passes its data directly to the Graph indicator.
   - ☐ Examine the panel, which consists of a Waveform Control and Graph indicator.
   - ☐ Examine the Waveform Control, which has a waveform data type. Take note of the following information contained in the waveform data type.

Initial Time (t0)	
Interval Time (dt)	
Data Values (Y)	
Index 0	
Index 1	
Index 2	
Index 3	
Index 4	
Index 5	
Index 6	
Index 7	
Index 8	
Index 9	
Index 10	

   - ☐ Based on the number of data values and the interval time, what do you think is the total duration in seconds of this waveform?

4. Run the VI.

   ☐ The waveform data type contains both data values and timing information. Therefore, the x-axis of the Graph indicator is in units of Time (seconds).

5. Modify the interval time (dt).

   ☐ Change the interval time from `0.1` seconds to `0.2` seconds.

   What effect do you think this change will have on the Graph indicator?

   ☐ Run the VI.

   ☐ Notice that the data values have remained the same. However, the amount of time between each point on the graph is now 0.2 seconds, and the total duration of the waveform on the graph has increased to 2 seconds.

6. Modify the initial time (t0).

   Currently the x-axis of the Graph indicator shows relative time in units of seconds, where the initial time is always displayed as 0.

   ☐ View the initial time (t0) in units of absolute date/time on the Graph indicator.

   – Click **Time (s)** on the x-axis of the Graph indicator.

   – In the Item tab, set the Display Format to **Absolute Time**.

   Notice that the time axis now displays units of absolute date and time.

   ☐ Modify the date and time in the initial time element of the Waveform Control.

   What effect do you think this change will have on the Graph indicator?

   ☐ Run the VI.

   Notice that the initial time on the left side of the Time axis now shows the new initial time.

Note To go back to displaying the time in seconds, select the **Time (s)** axis, and set the Display Format back to **Automatic**. You can also try experimenting with the other Display Format options.

7. Try making additional modifications to t0, dt, and Y array, and examine the effects on the Graph indicator.

1D Array of Waveform

1. Open **Examine 1D Array (Waveform) Data Type.gvi** from the Project Files tab.

2. Examine the 1D Array (Waveform) control on the panel.

   Notice that this 1D array contains three waveform elements. All three waveform elements contain the same initial time (t0) and interval time (dt).

1	The waveform at index 0 contains data values representing 1 sine wave.
2	The waveform at index 1 contains data values representing 1.5 sine waves with an amplitude offset of 3.
3	The waveform at index 2 contains data values representing 1.5 sawtooth waves with an amplitude offset of 6.

3. Examine the diagram. The 1D Array (Waveform) control passes its data directly to the Graph indicator.

4. Observe the behavior of the VI.

   ☐ Run the VI.

   ☐ Notice that when you pass a 1D array of waveforms to a graph, the graph displays each waveform in the array as a separate plot.

5. Observe the effects of modifying the waveforms in the 1D Array (Waveform) control.

   - ☐ Change the dt of one of the waveforms from `0.1` to `0.2`. Run the VI to observe the effect.
   - ☐ Change the t0 of one of the waveforms from `6:05:10.000` to `6:05:11.000`.
   - ☐ Run the VI to observe the effect.

End of Exercise 8-2

Exercise 8-3 Using Analysis Nodes to Analyze Data

Goal

Use analysis nodes on your VI diagram to analyze data.

Instructions

Analysis Nodes with Waveform Data Type

1. Open `<Exercises>\LabVIEW NXG Core 1\Analysis Nodes\Analysis Nodes.lvproject`.

2. From the Project Files tab, open **Using Analysis Nodes (Waveform data type).gvi**.

3. Examine the panel and diagram.

   ☐ Set the Physical Channel, Desired Sample Rate (Hz), and Number of Samples controls appropriately to acquire a signal from your DAQ device.

	Simulated cDAQ	cDAQ Demo Kit
Physical Channel	cDAQ1Mod8/ai0	cDAQ1Mod7/ai0
Desired Sample Rate (Hz)	1000	2560
Number of Samples	1000	256
Unbalanced fan	N/A	Maximum fan speed

   ☐ Notice that this VI performs a continuous acquisition.

   ☐ Notice that the **data** indicator is a waveform data type.

4. Capture and view the data that you want to analyze.

   ☐ Run the VI.

   ☐ Stop the VI when the graph indicator contains the data that you want to analyze.

   ☐ Right-click the graph indicator and select **Capture Data**.

   ☐ In the Captured data tab, rename your captured data item as `Sample Acquisition Data`.

   ☐ Double-click your captured data item to view it.

5. Use an Analysis Panel to analyze your captured data.

   ☐ In the Plots tab, select the **Filter** analysis panel in the Interactive Analysis section.

Note Select an analysis panel other than Filter if a different analysis panel is more appropriate for your acquisition data.

The Filter analysis panel displays the original signal in the Input Signal graph.

   ☐ Adjust the configuration settings until you get your desired result in the Filtered Signal graph.

   – **(Simulated cDAQ)** Set cutoff frequency to `2`.

   – **(cDAQ Demo Kit)** Set cutoff frequency to `100`.

Figure 8-15. Analysis Panel: Filter

6. Copy the configured analysis panel code into your VI diagram.

   ☐ At the bottom of the analysis panel, click and expand the **Configured Code** section.

1	Analysis node—In this example, the analysis node configured is a Butterworth Lowpass Filter (Waveform) with the constants that you set in the analysis panel settings.
2	Copy to clipboard—Copies the configured code to use in your VI.

7. Use the configured code from the analysis panel in a VI.

   ☐ Click the Copy to clipboard button.

   ☐ Switch to the diagram of the Using Analysis Nodes (Waveform data type) VI and paste the configured code onto the diagram by pressing <Ctrl-V> or selecting **Edit»Paste**.

   ☐ To conserve diagram space, right-click this node select **Terminals»Dock Constants**.

Note You can still see the constant values by hovering over the constants or on the Item tab for the node.

   ☐ Right-click the **filtered signal** output and select **Create»Indicator**.

☐ Switch to the panel and drag the filtered signal indicator from the Unplaced Items tray onto the panel.

☐ Wire the node as shown in the following figure.

Figure 8-16. Using Analysis Nodes (Waveform data type) VI Diagram

8. View the results of the analysis node.

☐ Run the VI.

☐ Compare the data in the Acquired Signal indicator and Filtered Signal indicator on the panel.

☐ Stop the VI when finished.

9. Adjust the configuration of the analysis node.

☐ On the diagram, click the analysis node.

Notice that you can adjust settings in the Item tab.

☐ You can also adjust the settings from the Analysis Panel. Click the **Analysis Panel** button.

☐ In the Analysis Panel, select the input signal for the panel to analyze by clicking the pull-down menu next to the Input Signal graph.

☐ Select **acquired signal <date/time>**, which corresponds to the data you captured earlier in this exercise.

☐ Modify the analysis panel configuration settings in the Item tab. For example, change the cutoff frequency.

☐ Update the node on the diagram by clicking the **Update Node** button in the Configured code section of the analysis panel.

☐ On the diagram, select the analysis node. View the configuration pane, and notice that the configuration settings (e.g. cutoff frequency) have been updated.

Analysis Nodes with 1D DBL Data Type

1. From the Project Files tab, open **Using Analysis Nodes (1D DBL data type).gvi**.
2. Switch to the diagram.
3. Examine the DAQmx Read node configuration.

   ☐ Select the DAQmx Read node and view the Item tab.

   Notice that the DAQmx Read node is configured to output single channel, multiple sample data using the a 1D array of double-precision floating point numbers.

4. Examine the Butterworth Lowpass Filter (DBL) configuration.

   ☐ Select this node and view the Item tab.

   ☐ Click the Analysis panel button and notice that the Analysis Panel does not work. The Analysis Panel only works if the incoming signal is the waveform data type.

   ☐ Notice that the Butterworth Lowpass Filter (DBL) node has an additional sampling frequency input. This is because the 1D DBL data type only contains data value and sample number information and does not contain any timing information. The Butterworth Lowpass Filter and many other analysis nodes require knowing the sampling frequency to perform.

- ☐ Notice that the sampling frequency input is the same as the actual sample rate output of the DAQmx Task Properties node. This is how you should wire the DBL versions of analysis nodes that have the sampling frequency input.

Note The Butterworth Lowpass Filter (Waveform) node from earlier in this exercise does not need a sampling frequency input because the waveform data type already includes the interval time (dt) timing information.

5. Run the VI, and notice that you get the same results as the earlier waveform data type section.
6. Stop the VI.

Your Turn

Experiment with using other analysis nodes.

End of Exercise 8-3

Exercise 8-4 Visualizing N-Channel Data

Goal

- Become familiar with the different data types that represent N-channel data
- View multiple channels of acquisition data in chart, graph, and array indicators.

Scenario

Select the column that best matches the needs of your applications.

Waveform	Array
I need to store the t0 (initial time of the measurement) and dt (time interval between samples) of my acquired data.	I only need to store the numeric values of the samples of my acquired data.

Check the option that best matches the needs of your applications.

	N Channels, 1 Sample
	N Channels, N Samples

Based on your answers to the previous questions, check which scenario(s) are most relevant to your own applications.

	N Channels, 1 Sample | Data type: 1D DBL Array
	N Channels, N Samples | Data type: 1D Waveform Array
	N Channels, N Samples | Data type: 2D DBL Array

Instructions

1. Open `<Exercises>\LabVIEW NXG Core 1\Visualize N-Channel Data\Visualize N-Channel Data.lvproject`.

2. Follow the instructions for the scenarios you selected in the *Scenario* section.

N Channel, 1 Sample (1D DBL Array)

In this section, you create a VI that acquires and visualizes one temperature sample from each of the four thermocouples at a rate of 1 Hz.

1. From the Project Files tab, open **N-Channel 1-Sample (1D DBL).gvi**.

2. Examine the properties of the DAQmx Create Channel node.

   ☐ Select the DAQmx Create Channel node and view the node configuration pane information in the Item tab.

   ☐ Notice that this node is configured for a Thermocouple measurement. You can modify this configuration if you want to acquire from a different type of measurement.

3. Examine the DAQmx Timing node.

   ☐ Notice that the sample rate is set 1 Hz. This sets the DAQ device to acquire 1 sample per channel per second.

4. Examine the DAQmx Read node.

   ☐ Select the DAQmx Read node and view the Item tab.

   ☐ Notice that this node is set to **Multiple Channels** and **Single Sample**. This node will return a 1D DBL array that contains a single sample from multiple channels.

5. Create the panel.

1	**Chart**—Expand the plot legend to display four plots.
2	**Numeric Array Output**—Resize this indicator to show 4 or more elements and rename it `1D Numeric Output`.

6. Modify the diagram, as shown in the following figure, to visualize the NChan1Samp data.

Figure 8-19. N-Channel 1-Sample (1D DBL) VI Diagram

1 **Array to Cluster**—To plot a single sample from multiple channels on a Chart, you must pass the Chart a cluster data type containing the samples. This node converts the 1D array data type to a cluster data type.

In the Item tab, set Cluster Size to 4.

2 **Chart**—Drag this indicator from the Unplaced Items tray to the diagram. Charts show historical data and append new data.

3 **1D Numeric Output**—Drag this indicator from the Unplaced Items tray to the diagram. This array indicator only displays the current array data in a specific iteration.

Note The cluster size is set to 4 because you will be acquiring a single sample from four different channels in this exercise. You will learn more about the cluster data type in the *Grouping Data of Mixed Data Types (Clusters)* lesson.

7. Configure the Chart properties.

   ☐ Resize the Plot Legend to show 4 plots.

   ☐ Double-click Time on the x-axis and rename it as `Sample`. The 1D DBL data type does not contain any timing information, so the x-axis just represents the sample number.

   ☐ Double-click the minimum and maximum numbers on the axis and change them to `0` and `10`. Now the chart will show the most recent 10 samples per channel acquired.

   ☐ Click the chart and view the Item tab. Notice that the History Length is set to 1024 by default.

   This setting determines the maximum number of points per plot that the chart can show.

8. Your panel should look similar to Figure 8-20 when the VI is running.

Figure 8-20. N-Channel 1-Sample (1D DBL) VI Panel

9. Test the VI.

   ☐ On the panel, set the Thermocouple Channels control to **cDAQ1Mod2/ai0:3**. This creates a DAQmx task that includes cDAQ1Mod2 thermocouple channels 0, 1, 2, and 3.

   Note Set the Thermocouple Channels control to channels that are appropriate for your DAQ device.

   ☐ Run the VI and notice the following things.

   – The rate input (1 Hz) of the DAQmx Read node controls the loop period.

   – Every iteration, one sample from 4 channels are updated on the Chart.

   – Every iteration, the 1D DBL Array indicator only shows the data for the current iteration.

   – The x-axis of the Chart is in units of Sample number (not seconds).

   – The Chart shows the most recent 10 samples per channel acquired.

   ☐ Double-click the minimum and maximum numbers on the x-axis and change them to `0` and `5`. Now the chart will show the most recent 5 samples per channel acquired.

   ☐ Stop the VI when finished.

10. To clear the chart between runs of the VI, right-click the chart and select **Clear Data.**

N Channel, N Samples (1D Waveform Array)

In this section, you create a VI that acquires and visualizes 4 channels of analog input voltage. Each loop iteration acquires/visualizes 1000 samples per channel. The samples are continuously acquired at a HW-timed rate of 1000 Hz. The waveform data type contains the sample values (Y), the start time for the samples (t0), and the time interval between each sample (dt).

(cDAQ Demo Kit) Modify this VI to acquire 3 channels of voltage (X acceleration, Y acceleration, and tachometer) from cDAQ1Mod7/ai0:2 at a sample rate of 2,560 Hz.

1. From the Project Files tab, open **N-Channel N-Samples (1D Waveform).gvi**.
2. Examine the DAQmx Create Channel node.
   - ☐ Select the DAQmx Create Channel node and view the Item tab.

   Notice that this node is configured for an Analog Input Voltage measurement. Modify the configuration if you want to acquire from a different type of measurement.

3. Examine the sample rate and number of samples controls.
   - ☐ The Desired Sample Rate (Hz) control sets sample rate of this DAQ task.
   - ☐ The Number of Samples control sets how many samples per channel the DAQmx Read node will wait for and output in each loop iteration.
   - ☐ If the sample rate is 1000 Hz and number of samples is 1000, then during each While Loop iteration, the DAQmx Read node acquires from multiple channels at a sample rate of 1000 Hz and outputs 1000 samples per channel.

4. Examine the DAQmx Read node.

   ☐ Select the DAQmx Read node and view the Item tab.

   Notice that this node is set to **Multiple Channels**, **Multiple Samples**, and **Waveform**. This node will return a 1D waveform array that contains multiple samples from multiple channels.

5. Create the panel.

   ☐ Add a **Graph** to the panel. Resize the Plot Legend to show 4 plots.

   ☐ Add a **Chart** to the panel. Resize the Plot Legend to show 4 plots.

   ☐ Create a 1D Waveform Array indicator.

   – Add an **Unconfigured Array** on the panel. Rename it `1D Waveform Indicator`.

   – Add a **Waveform** control inside the unconfigured array.

   – Resize the array to show 4 elements.

   – On the Item tab for the array, select **Indicator** as the Behavior.

6. Drag each of the waveform indicators from the Unplaced Items tray to the diagram. Modify the diagram, as shown in the following figure, to visualize the NChanNSamp (waveform) data.

Figure 8-21. N-Channel N-Samples (1D Waveform) VI Diagram

1 **Waveform Graph**—Graphs display the current array data in a specific iteration.
2 **Waveform Chart**—Charts show historical data and append new data.
3 **1D Waveform Indicator**—This indicator only displays the current data in a specific iteration. Each array element is the waveform data received from an individual channel.

7. Configure the Graph properties.

   ☐ Switch to the panel and double-click **Time** on the x-axis and rename it as `Time (s)`. The Waveform data type contains timing information in units of seconds.

8. Configure the Chart properties.

   ☐ Double-click **Time** on the x-axis and rename it `Time (s)`. The Waveform data type contains timing information in units of seconds.

   ☐ Double-click the minimum and maximum numbers on the x-axis and change them to `0` and 5. Now the chart will show the most recent 5 seconds of data acquired.

   ☐ Select the Chart and change History Length from 1,024 to 5,000 in the Item tab.

   This setting determines the maximum number of points per plot that the chart can show.

Note Because this VI acquires 1000 samples per channel per second, changing the History Length to 5,000 configures the chart to show 5,000 samples per channel, which is equivalent to most recent 5 seconds of data.

9. Your panel should look similar to the following figure when the VI is running.

Figure 8-22. N-Channel N-Samples (1D Waveform) VI Panel

10. Test the VI.

- ☐ On the panel, set the Physical Channel control to **cDAQ1Mod8/ai0:3**. This creates a DAQmx task that includes cDAQ1Mod8 analog input channels 0, 1, 2, and 3.
- ☐ Run the VI.
- ☐ Notice that the following things happen every iteration.
  - Multiple samples from 4 channels are appended to the Waveform Chart.
  - The Waveform Graph shows the data only for the current iteration (that is, the 1,000 samples read in that specific iteration).
  - The 1D Waveform indicator shows the data only for the current iteration.I
- ☐ Notice that the x-axis of the Waveform Chart is in units of Time (s). This is because the waveform data type includes timing information.
- ☐ Stop the VI when finished.

N Channel, N Samples (2D DBL Array)

In this section, you create a VI that acquires and visualizes 4 channels of analog input voltage. Each loop iteration acquires/visualizes 1,000 samples per channel. The samples are continuously acquired at a HW-timed rate of 1,000 Hz. The 2D DBL array data type contains the sample values.

(cDAQ Demo Kit) and **(Simulated Hardware)** Modify the VI to acquire 3 channels of voltage (X acceleration, Y acceleration, and tachometer) from cDAQ1Mod7/ai0:2 at a sample rate of 2,560 Hz and number of samples of 256.

1. From the Project Files tab, open **N-Channel N-Samples (2D DBL).gvi**.

2. Examine the DAQmx Create Channel node.

   ☐ Switch to the diagram and select the DAQmx Create Channel node and view the Item tab.

   Notice that this node is configured for an Analog Input Voltage measurement. Modify the configuration if you want to acquire from a different type of measurement.

3. Examine the sample rate and number of samples controls.

   ☐ The Desired Sample Rate (Hz) control sets the sample rate for this DAQ task.

   ☐ The Number of Samples control sets how many samples per channel the DAQmx Read node will wait for and output in each loop iteration.

   ☐ If the sample rate is 1,000 Hz and number of samples is 1,000, then during each While Loop iteration, the DAQmx Read node will acquire from multiple channels at a sample rate of 1,000 Hz and output 1,000 samples per channel.

4. Examine the DAQmx Read node.

   ☐ Select the DAQmx Read node and view Item tab.

   Notice that this node is set to **Floating Point, Multiple Channels**, and **Multiple Samples**. This node will return a 2D DBL array that contains multiple samples from multiple channels.

5. Create the panel.

   ☐ Switch to the panel and add the following indicators.

   – Graph indicator

   – Chart indicator

   – 2D numeric output indicator

6. Drag the indicators from the Unplaced Items tray to the diagram and modify the diagram, as shown in the following figure, to visualize the NChanNSamp (waveform) data.

Figure 8-23. N-Channel N-Samples (2D DBL) VI Diagram

1 **Transpose 2D Array**—You must transpose the 2D array before passing it to the chart for the chart to correctly interpret the 2D array as 4 channels of data with 1,000 samples per channel.
2 **Graph indicator**—Graphs display the current array data in a specific iteration.
3 **Chart indicator**—Charts show historical data and append new data.
4 **2D Numeric Array indicator**—This indicator only displays the current data in a specific iteration. Each row contains the data points for a channel. When you run this VI, you will notice that for a “4 channel, 1,000 samples per channel” acquisition, the 2D array has 4 rows and 1,000 columns.

7. Configure the Graph properties.

   ☐ Switch to the panel.

   ☐ Resize the Plot Legend to show 4 plots.

   ☐ Double-click **Time** on the x-axis and rename it as `Sample`. The 2D DBL data type does not contain any timing information, so the x-axis just represents the sample number.

8. Configure the Chart properties.

   ☐ Resize the Plot Legend to show 4 plots.

- ☐ Double-click **Time** on the x-axis and rename it as `Sample`. The 2D DBL data type does not contain any timing information, so the x-axis just represents the sample number.
- ☐ Double-click the minimum and maximum numbers on the x-axis and change them to `0` and `5000`. Now the chart will show the most recent 5,000 samples per channel acquired.
- ☐ Select the Chart. In the **Item** tab, change History Length from 1024 to `5000`. This setting determines the maximum number of points per plot that the chart can show.

Note Because this VI acquires 1,000 samples per channel per second, changing the History Length to 5,000 configures the chart to show 5,000 samples per channel, which is equivalent to most recent 5 seconds of data.

9. Your panel should look similar to the following figure when the VI is running:

Figure 8-24. N-Channel N-Samples (2D DBL) VI Panel

10. Test the VI.

- ☐ On the panel, set the Physical Channels control to **cDAQ1Mod8/ai0:3**. This creates a DAQmx task that includes cDAQ1Mod8 analog input channels 0, 1, 2, and 3.

Note Set the Physical Channels control to channels that are appropriate for your DAQ device.

- ☐ Run the VI.
- ☐ Notice that the following things happen every iteration.
  - Multiple samples from 4 channels are appended to the Chart.
  - The Graph shows the data only for the current iteration (that is, the 1,000 samples read in that specific iteration).
  - 2D Numeric Array indicator shows the data only for the current iteration.
- ☐ Notice that the x-axis of the Chart indicates the sample number of a data point (not seconds). This is because the 2D DBL data type does not includes time data.
- ☐ Stop the VI when finished.

End of Exercise 8-4

Exercise 8-5 Extract a Subset of Multi-Channel Acquisition Data

Goal

Extract a single channel of data using the Index Array node.

Extract multiple channels of data using the Array Subset node.

Scenario

In this exercise, the acquisition data is a 1D waveform array containing the following data.

1D Waveform Array Index	Signal Description
0	Accelerometer mounted in X direction
1	Accelerometer mounted in Y direction
2	Tachometer (2 pulses/revolution)

Hardware Setup

(cDAQ Demo Kit) Set the Sound & Vibration Fan switch to `Unbalanced`.

Explore acquisition data in an existing VI.

1. Open `<Exercises>\LabVIEW NXG Core 1\Extract Data from N-Channel Array\Extract Data from N-Channel Array.lvproject`.

2. From the Project Files tab, open **Extract Data from N-Channel Array (1D Waveform).gvi**.

3. Explore the panel and diagram.

   The diagram code performs a continuous acquisition.

   By examining the Item tab of the DAQmx Read node, you can determine that this node reads N samples from N channels and returns the data as a 1D waveform array.

4. Examine the behavior of the VI.

   - ☐ On the panel, set the Physical Channel control to **cDAQ1Mod7/ai0:2**.

     This creates a DAQmx task that includes cDAQ1Mod7 analog input channels 0, 1, and 2.

   - ☐ Set Desired Sample Rate (Hz) to `2560`.

   - ☐ Set Number of Samples to `256`.

   Note Set the Physical Channel, Rate, and Number of Samples controls to values that are appropriate for your DAQ device.

5. Run the VI.

   - ☐ Notice that the All Channels indicator displays data for the three channels specified by the Physical Channel control:

     - AI0 channel
     - AI1 channel
     - AI2 channel

Because AI0 and AI1 contain a different measurement type with different ranges than AI2, you should probably separate them into different data sets.

Extract Data for a Single Channel

1. Modify the diagram, as shown in the following figure, to extract data for a single channel.

1	**Index Array node**—When you pass this node a 1D waveform array and an index value, this node will return a single waveform element at the specified index in the 1D waveform array.

   - ☐ Switch to the panel, and drag the Indexed Channel graph indicator from the Unplaced Items palette to the panel.

2. Use the Context Help and LabVIEW Help to read about the Index Array function.

3. Test the VI.

   - ☐ On the panel, set the following values.

Physical Channel	cDAQ1Mod7/ai0:2
Rate	2560
Number of Samples	256

   - ☐ Run the VI.
   - ☐ Change the value of the Index control to `0`, `1`, and `2`. Notice that this control determines which single channel is extracted and displayed in the Indexed Channel indicator.
   - ☐ Stop the VI when finished.

4. Save the VI.

Extract a Subset of Channels

1. Modify the diagram, as shown in the following figure, to extract a subset of channels.

1	Index Array node—Use this node to extract the third channel (index 2). Right-click the Index control and select **Change to Constant**. Set the constant to `2`
2	Array Subset node—Use this node to extract the first and second channels (indices 0 and 1) – Right-click the index input and select **Create Constant.** Set the constant to `0`. – Right-click the length input and select **Create Constant**. Set the constant to `2`. – Right-click the subarray output and select **Create Indicator**. Rename the indicator as `Subset of Channels`.

☐ Go to the panel, and drag the Subset of Channels indicator from the Unplaced Items palette to the panel.

2. Use the Context Help and detailed LabVIEW Help to read about the Array Subset node.

3. Test the VI.

☐ On the panel, set the following values.

Physical Channel	cDAQ1Mod7/ai0:2
Rate	2560
Number of Samples	256

☐ Run the VI.

☐ Notice that the Indexed Channel indicator displays the AI2 channel.

- ☐ Notice that the Subset of Channels indicator displays the AI0 and AI1 channels.
- ☐ Stop the VI when finished.

4. Save the VI.

Extract a Subset of Channels—Alternate Approach using Index Array and Build Array

1. From the Project Files tab, open **[Alternate Method] Extract Data from N-Channel Array (1D Waveform).gvi**.

   - ☐ Notice how this VI uses the Index Array to extract multiple channels and then group them together into a new array using the Build Array node.

2. Add a new Index Array node and a Build Array node.

   - ☐ Notice how you can resize the Index Array to extract multiple array elements. Notice how you can resize the Build Array node to take multiple inputs. Use the Context Help and detailed LabVIEW Help to read about these two nodes.

Explore how to extract channels from 2D DBL Array data type

1. From the Project Files tab, open **Extract Data from N-Channel Array (2D DBL).gvi**.

2. Examine the behavior of the VI.

   ☐ On the panel, set the following values.

Physical Channel	cDAQ1Mod7/ai0:2
Rate	2560
Number of Samples	256

   ☐ Run the VI.

   ☐ Stop the VI when finished.

3. On the diagram, view the Item tab of the DAQmx Read node.

   ☐ Notice that the DAQmx Read is using the **Floating Point** data type. Notice that the output data type is a 2D DBL array.

4. Notice that the Index Array node is used to extract one channel of data (1D array = 2,560 samples from channel 0).

5. Notice that wiring a 2D array to the Index Array node causes the Index Array node to show both a row index input and column index input.

   ☐ If you want to extract a row (1D array) from a 2D array, only wire the index of the row to the index (row) input and leave the index (col) input unwired.

   ☐ If you want to extract a column (1D array) from a 2D array, only wire the index of the column to the index (col) input and leave the index (row) input unwired.

- ☐ If you want to extract an individual element (single numeric) from a 2D array, then wire the row index to the index (row) input and wire the column index to the index (col) input.

  In this VI, the 2D array wired to the Index Array node contains 3 rows and 2,560 columns, where each row represents 2,560 samples acquired from one channel. Therefore, to extract one channel of data (represented by 1D array), this VI wires the row index and leaves the column index unwired to extract one row of data from the 2D array.

6. Notice that the Array Subset function is used to extract two channels of data.

   - ☐ A Row Index of 0 and a Length of 2 causes this node to extract 2 rows of data starting from row index 0.

Your Turn

Option 1

1. Create a VI that acquires N-channel N-sample data and try extracting individual or groups of channels from that acquisition data array.

2. **(cDAQ Demo Kit)** Try acquiring 4 channels (10 samples per channel) of thermocouple data. Extract an individual channel or groups of thermocouple channels.

Option 2: Experiment with Index and Array Subset nodes

1. From the Project Files tab, open **[Unguided] Example 2D Array.gvi**.

2. Familiarize yourself with the functionality of the Index Array and Array Subset nodes by wiring the 2D array to these nodes, experimenting with how the input terminal values of these nodes affect their output values, and reading the LabVIEW Help.

You can resize the Index Array node to extract more than one element/row/column.

End of Exercise 8-5

Exercise 8-6 Auto-Indexing Tunnels

Goal

Explore the behavior of auto-indexing input and output tunnels.

Instructions

1. Open `<Exercises>\LabVIEW NXG Core 1\Auto-Indexing\Auto-Indexing.lvproject`.
2. From the Project Files tab, open **Auto-Indexing Tunnels.gvi**.
3. Modify the diagram, as shown in the following figure, to create an auto-indexing input tunnel and auto-indexing output tunnel to the For Loop.

Figure 8-29. Auto-Indexing Tunnels VI Diagram

☐ Wire the 1D DBL Array control to the left border of the For Loop, which by default creates an auto-indexing input tunnel. Then, wire the input tunnel to the Input Tunnel Result indicator.

Note If you wire an array to a For Loop or While Loop, you can read and process every element in that array by enabling auto-indexing. When you wire an array from an external node to an input tunnel on the loop border and enable auto-indexing on the input tunnel, elements of that array enter the loop one at a time, starting with the first element.

☐ Wire the output of the Multiply node to the right border of the For Loop, which by default creates an auto-indexing output tunnel. Then, wire the output tunnel to the Output Tunnel Result indicator.

Note For auto-indexing output tunnels, scalar elements accumulate sequentially into 1D arrays, 1D arrays accumulate into 2D arrays, and so on.

4. Select the auto-indexing input tunnel. Notice in the Item tab, **Auto index values** is selected. This is where you can set the behavior of any loop input tunnel to enable or disable auto-indexing.

Figure 8-30. Input Tunnel Configuration Pane

5. Select the auto-indexing output tunnel. Notice in the Item tab, Auto index values is selected. This is where you can set the behavior of any loop output tunnel to **Auto index values**, **Concatenate values**, or **Keep last value**.

Figure 8-31. Output Tunnel Configuration Pane

6. Turn on Highlight Execution.

7. Run the VI.

   - ☐ On the panel, observe the behavior or each control/indicator on the panel.
   - ☐ On the diagram, observe how elements of the 1D DBL Array enter the loop one at a time, starting with the first element.
   - ☐ Observe how the Output Tunnel Result indicator is a 1D array that consists of every result passed into the auto-indexing output tunnel while the For Loop was executing.
   - ☐ Notice how the For Loop output tunnel does not output anything until the entire For Loop has finished executing.
   - ☐ Run the VI multiple times with Highlight Execution turned on to make all the above observations.

Contrast Auto index values behavior with Keep last value behavior

1. Open **Compare Auto-Index and Last Value.gvi**.

2. Explore the panel and diagram.

   ☐ Notice that there is both an Auto Index and Keep Last Value indicator for each For Loop input tunnel and output tunnel.

   ☐ On the diagram, select each tunnel. Notice in the configuration pane how each tunnel is configured.

   ☐ Turn on Highlight Execution.

3. Run the VI.

   ☐ Observe the differences in data type and behavior for the input tunnels.

   ☐ Observe the differences in data type and behavior for the output tunnels.

End of Exercise 8-6

Exercise 8-7 Process Data For Each Channel in N-Channel Data Array

Goal

Process data for each channel by using For Loop to iterate through each channel in an n-channel data array.

Process Channels Using Node Compatible with N-Channel Array Data Types

1. Open `<Exercises>\LabVIEW NXG Core 1\Process N-Channel Array\Process N-Channel Array.lvproject`.
2. From the Project Files tab, open **Process N-Channel Array.gvi**.
3. Examine the VI.
   - ☐ Explore the panel and diagram.
   - ☐ Notice that the DAQmx Read data output is a 1D Waveform Array data type.
   - ☐ Run the VI. Stop the VI when finished.

4. Modify the diagram, as shown in the figure below, to process all channels by scaling their values by 100 and filtering the signals using the following items.

Figure 8-34. Process N-Channel Array VI Diagram

1 **Multiply node**—The input of this function can adapt to accept the 1D Waveform Array data type.
2 Right-click the **input 1** input and select **Create Constant**. Set the constant to `100`. This node will multiply all Y values in every waveform element in the array by 100.
3 Right-click the output and select **Create Indicator**. Rename the indicator `Scaled Channels`.
4 **Filter node**—In the Item tab, verify that the node is set to **Lowpass** and **Butterworth**.
5 Wire the 1D Waveform wire from the DAQmx Read node to the input of the Filter node. Notice that the Filter node automatically adapts to accept the 1D Waveform data type. The Filter node will process each waveform contained in the array.
6 Right-click the cutoff frequency input and select **Create Control**.
7 Right-click the output terminal and select **Create Indicator**. Rename the indicator as `Filtered Channels`.

Note Many math nodes can adapt to accept the 1D Waveform Array data type. A real-world application can use math nodes to scale acquired data from units of voltage to the appropriate engineering unit, such as temperature (deg Celsius), acceleration (g), etc.

5. Switch to the panel and drag the terminals you created on the diagram from the Unplaced Items tray to the panel. Arrange your panel similar to the following figure.

Figure 8-35. Process N-Channel Array VI Panel

6. Examine the behavior of the VI.

   ☐ Run the VI.

   – **(Simulated cDAQ)** Set the Cutoff Frequency control to `1`.

   – **(cDAQ Demo Kit)** Set the Unbalanced Fan to the maximum fan speed. Set the Cutoff Frequency control to `100`.

   ☐ Notice that the Scaled Channels and Filtered Channels graph indicators show that the Multiply and Filter nodes processed all channels in the 1D Waveform array.

   ☐ Stop the VI when finished.

Process Channels Using a Node that is Incompatible with N-Channel Array Data Types

1. Modify the diagram, as shown in the following figure, to run all channels through a node that is not compatible with the 1D Waveform data type.

Figure 8-36. Process N-Channel Array VI Diagram

1 For Loop—Add a For Loop to the diagram
2 RMS node—The input of this node is not compatible with 1D Waveform or 2D DBL data types.
3 Wire the DAQmx Read VI output wire to the left border of the For Loop and into the Filter VI input. Notice the auto-indexing input tunnel. On the first iteration of the For Loop, this auto-indexing input tunnel will output the first element in the array. On the second iteration of the For Loop, this input tunnel will output the second element in the array, and so on.
4 Wire the rms value output of the RMS node to the right border of the For Loop. This creates an auto-indexing output tunnel. When the For Loop finishes executing, the output of this auto-indexing output tunnel will be a 1D array with its elements containing what the auto-indexing output tunnel received in each For Loop iteration.
5 Right-click the auto-indexing output tunnel and select **Create Indicator**.

Note This For Loop code would work for any N-channel array, whether it contains 2 channels or 16 channels.

2. Arrange your panel similar to the following figure.

Figure 8-37. Process N-Channel Array VI Panel While Running

3. Examine the behavior of the VI.
   - ☐ Run the VI.
   - ☐ Notice that the RMS values 1D DBL array indicator shows the RMS value of each channel. This shows that the For Loop processed all channels in the 1D Waveform array.
   - ☐ Stop the VI when finished.
4. Save the VI.

Your Turn

The following pseudocode shows how to take one channel of data and process it to display the data in the frequency domain (frequency vs amplitude graph).

1. Create a copy of the VI in this exercise.
   - ☐ In the Project Files tab, right-click the VI and select **Duplicate**. Rename the project `[Challenge] Process N-Channel Array.gvi`.
2. Modify the VI to perform the above algorithm on all channels.

Answer

The answer is located in the `<Solutions>\LabVIEW NXG Core 1\Exercise 8-7` directory.

End of Exercise 8-7

Exercise 8-8 Explore Additional Array Examples

Goal

Explore the following examples that require the use of the array data type:

- Voltage ramp
- Stimulus-response
- Generate buffered output
- Programmatically create 2D array

Guided Instruction

1. Open `<Exercises>\LabVIEW NXG Core 1\Additional Array Examples\Additional Array Examples.lvproject`.

2. Open, explore, and run the following examples. Use Highlight Execution and Context Help to better understand each example.

   ☐ Voltage ramp

   – Voltage ramp.gvi

   ☐ Stimulus-response

   – Stimulus-Response (Scalar).gvi

   – Stimulus-Response (Waveform).gvi

   ☐ Generate buffered output

   – Generate Output (Waveform).gvi

   – Generate Output (1D DBL).gvi

   ☐ Programmatically create 2D array

   – Programmatically Create 2D Array.gvi

3. Close the VIs and project when finished.

End of Exercise 8-8

Exercise 9-1 Execute Code Based on a Condition

Goal

Use a Case Structure to execute code based on a condition.

Hardware Setup/Scenario

cDAQ Demo Kit Option

1. Set the Sound & Vibration switch to **Unbalanced**.
2. Set the Sound & Vibration switch in the Fan Control section to **Dial**.

Execute Code Conditionally Based on a Measurement Result—Conditionally Display Data on a Graph

1. Open `<Exercises>\LabVIEW NXG Core 1\Execute Conditional Code\Execute Conditional Code.lvproject`.
2. From Project Files, open **Condition Based on Measurement Result.gvi**.
3. Explore the VI.
   - ☐ Explore the panel and diagram.
   - ☐ Run the VI.

     (cDAQ Demo Kit)

     - Turn the Fan Control dial and observe the effect on the Current Acceleration graph.
     - Tap the Sound & Vibration Signal Simulator and observe the effect on the Current Acceleration graph.

     (Simulated cDAQ)

     - Simulated data will appear on the Current Acceleration graph.
   - ☐ Stop the VI when finished.

4. Modify the VI, as shown in the following diagram, to display acceleration data on the Last Exceeded Acceleration graph only if the acceleration data has exceeded a user-specified threshold.

Figure 9-10. Condition Based on Measurement Results Diagram

1 **Absolute Value**—Returns the absolute values of the input.
2 **Waveform Min Max node**—The Y max output returns the maximum value of the Y array in the waveform.
3 **Greater?**—Right-click the y input of the Greater? node and select **Create Control**. Rename it as `Acceleration Threshold (g)`.
4 **Case Structure**—This VI will use the Case Structure to update the Last Exceeded Acceleration graph indicator only when the acquired acceleration exceeds a threshold.
5 **Last Exceeded Acceleration**—Add a graph indicator to the panel and wire the corresponding terminal as shown.

5. Configure panel items.

   ☐ Drag the Acceleration Threshold (g) control from the Unplaced Items tray to the diagram.

   ☐ Configure the **Last Exceeded Acceleration** graph to show the absolute time, so you can see the exact date and time of when the acquired data last exceeded the threshold.

   – Click **Time** on the x-axis.

   – In the Item tab, select **Absolute Time** in the Display format section.

   – On the diagram, drag this indicator from the Unplaced Items tray to the True case of the Case Structure.

   ☐ Wire this indicator as shown in Figure 9-10.

6. Arrange the panel, as shown in the following figure.

Figure 9-11. Condition Based on Measurement Results Panel

7. Set the control values, as shown in the Figure 9-11. Otherwise set the values to whatever is appropriate for your hardware setup.

8. **(cDAQ Demo Kit)** Lower the Acceleration Threshold to `0.75`. Set the maximum and minimum values of the Current Acceleration y-axis to `1.25` and `-1.25`.

   You should be able to exceed the acceleration threshold in the next step by firmly tapping the Sound & Vibration Signal Simulator in the cDAQ Demo Kit.

9. You can also try turning the dial to rotate the unbalanced fan at its maximum speed and adjust the Acceleration Threshold control value.

10. Examine the behavior of the VI.

    ☐ Run the VI.

    ☐ Adjust the value of the Acceleration Threshold (g) control and/or the acquired signal, so that the Current Acceleration exceeds the Acceleration Threshold from time to time.

    ☐ Notice that every time the Current Acceleration graph contains a value above the Acceleration Threshold (g) control value, that data is passed to the Last Exceeded Acceleration graph.

- ☐ Notice that the x-axis of the Last Exceeded Acceleration graph shows the absolute date and time of the last acceleration waveform that exceeded the threshold.
- ☐ Stop the VI when finished.

11. Select **File»Save All** to save the VI and project.

Execute Code Conditionally Based on a User Setting/Parameter/Configuration

Explore a DAQmx example that sets the triggering behavior based on a user-input.

1. From the Project Files tab, open **Condition based on User Input (Triggering).gvi**.

   Note This VI is based on the Finite Input with Voltage Probe.gvi DAQmx example program located in Learning tab under **Examples»Hardware Input and Output»NI-DAQmx»Programmatic Configuration»NI-DAQmx Analog Input**.

2. Explore the diagram.

   - ☐ Notice that the Case Structure executes code based on the value of the Trigger Enabled? control.

3. Test the VI.

   - ☐ Set the following control values.

Physical Channel	cDAQ1Mod8/ai0
Digital Trigger Source	cDAQ1/PFI0
Edge	Rising
Triggering Enabled?	On

☐ Run the VI.

The VI is now waiting on the cDAQ1/PFI0 line to detect a rising edge from a digital signal.

☐ **(cDAQ Demo Kit)** Physically press the TRIGGER button on the cDAQ Demo Kit. The VI should immediately return data on the Graph indicator.

If you set the Triggering Enabled? control to OFF on the panel and run the VI again, you will see that VI immediately returns data and no longer waits for a trigger.

Note The cDAQ1/PFI0 line is connected to the physical Trigger button on the cDAQ Demo Kit.

☐ **(Simulated cDAQ)** Because you are using a simulated DAQ device, the cDAQ1/PFI0 line randomly changes between True and False. When this line goes from False to True, the VI detects this as a rising edge, which triggers the VI to acquire data.

☐ On the diagram, use Highlight Execution to observe how the Triggering Enabled? control value determines which case the Case Structure executes.

4. Close the VI when finished.

Challenge

Add code to **Condition Based on Measurement Result.gvi** to display a running total of number of times the threshold has been exceeded since the VI started running.

☐ **Hint:** Use a shift register to store the running total. Increment the running total in the Case Structure.

☐ **Solution:** Open `<Solutions>\[Solution Challenge] Condition Based on Measurement Result.gvi`.

On the Job

1. In your own applications, do you need to execute code conditionally based on a measurement result? If so, describe below.

2. In your own applications, do you need to execute code conditionally based on a user-input (e.g. Boolean on/off button)? If so, describe below.

End of Exercise 9-1

Exercise 10-1 Using High-Level I/O Nodes

Goal

Use high-level file I/O nodes to write an array of acquisition data to file.

Hardware Setup

- cDAQ Demo Kit—setup X & Y accelerometer channel (Mod7/ai0:1)
- Simulated—setup ai0:3 of an analog input module (Mod7 or Mod8)

Guided Instruction

Finite Acquisition of *N* Channels and *N* Samples/Channel

1. Open `<Exercises>\LabVIEW NXG Core 1\High-Level Write to File\High-Level Write to Text File.lvproject`.
2. From the Project Files tab, open **High-Level Write to Text File – N-Channel.gvi**.
3. Examine the VI.
   - ☐ Notice that the DAQmx Read node is configured to read from Multiple Channels and Multiple Samples as a 2D DBL array.
   - ☐ Notice that the DAQmx Timing node configures this VI to perform a finite acquisition of n samples per channel (Number of Samples control) at a defined sample rate (Sample Rate control).

4. Modify the diagram, as shown in the following figures, to write the acquisition data to file.

1	**Write Delimited Spreadsheet**—Use the first Write Delimited Spreadsheet node to write the header to the file. Add the node to the diagram.
2	Right-click the file path input and select **Create Control**. Rename the control as `Data log filepath`.
3	**String Array Constant**—Place a String Array Constant on the diagram. In the Item tab, set Dimensions to `2`. Resize the constant and set the values in the first row, as shown. Wire the constant to the 2D data input of the Write Delimited Spreadsheet node, which will adapt to the 2D string array data type.
4	Notice that the append to file input is unwired, so it will default to False. This means that if a file already exists at the data log filepath, this node will overwrite any data in that text file.

☐ Configure the Data log filepath control.

- Switch to the panel, and drag Data log filepath control from the Unplaced Items tray to the panel.
- Select the control and view the file path Item tab.

- Set Action to **Save As**. This allows the user to browse to a new or existing filepath where the user wants to save the log file.
- Set Pattern Label to `Text file`.
- Set Pattern to `txt`.

☐ Refer to the following figure to transpose the 2D array to represent data by column instead of row and write numeric data to the file.

1 **Transpose 2D Array**—Use this node to transpose the 2D array so that each channel's data is represented by a column instead of a row. In this exercise, this is how we want the spreadsheet file to show the data.

2 **Write Delimited Spreadsheet**—use the second Write Delimited Spreadsheet node to write the numeric data to the file.
The previous Write Delimited Spreadsheet node added headers to the text file, and this Write Delimited Spreadsheet node appends data to the file after the headers.

3 Wire the file path terminals between the two Write Delimited Spreadsheet nodes, as shown in the figure. This tells the second node to write to the same file as the first node.

5. Complete the wiring as shown in the following figure.

6. Create the text file.

☐ On the panel, set the Physical Channel control to the following value:

- **(cDAQ Demo Kit)** cDAQ1/Mod7/ai0:3
- **(Simulated cDAQ)** cDAQ1/Mod7/ai0:3

☐ On the panel, click the ... button of the Data log filepath control. In the Save As dialog, set the filepath to `<Exercises>\LabVIEW NXG Core 1\High-Level Write to File\Finite Acquisition Data.txt`.

☐ Run the VI.

7. Explore the text file created by this VI.

   - ☐ In Windows Explorer, navigate to and double-click the data log file to view its contents.
   - ☐ Notice the file includes a header describing each channel.
   - ☐ Verify that the log file looks similar to the following figure.

```
Channel 0       Channel 1       Channel 2       Channel 3
-0.150  0.442   0.750   1.348
0.330   0.719   1.095   1.666
0.705   1.099   1.336   1.940
0.972   1.325   1.663   1.965
1.083   1.570   1.897   2.280
1.645   1.879   2.163   2.493
1.638   2.135   2.551   2.913
2.095   2.472   2.695   3.179
2.322   2.654   2.906   3.389
2.684   3.036   3.276   3.510
2.963   3.250   3.625   3.897
3.103   3.298   3.682   3.874
3.428   3.528   3.967   4.225
3.684   3.961   4.091   4.242
3.667   3.931   4.319   4.266
3.886   4.035   4.412   4.395
3.948   4.431   4.417   4.559
4.276   4.495   4.692   4.778
4.384   4.463   4.753   4.763
4.496   4.782   4.824   4.700
4.642   4.691   4.873   4.876
```

   - ☐ Try opening the text file using Microsoft Excel. Verify that the log file looks similar to the following figure.

	A	B	C	D
1	Channel 0	Channel 1	Channel 2	Channel 3
2	-0.15	0.442	0.75	1.348
3	0.33	0.719	1.095	1.666
4	0.705	1.099	1.336	1.94
5	0.972	1.325	1.663	1.965
6	1.083	1.57	1.897	2.28
7	1.645	1.879	2.163	2.493
8	1.638	2.135	2.551	2.913
9	2.095	2.472	2.695	3.179
10	2.322	2.654	2.906	3.389
11	2.684	3.036	3.276	3.51
12	2.963	3.25	3.625	3.897
13	3.103	3.298	3.682	3.874
14	3.428	3.528	3.967	4.225
15	3.684	3.961	4.091	4.242
16	3.667	3.931	4.319	4.266
17	3.886	4.035	4.412	4.395
18	3.948	4.431	4.417	4.559

Your Turn

1. Use the LabVIEW NXG Help to learn more about each node.
2. Experiment with the VI to better understand its functionality. For example:
   - ☐ What is the effect on the log file if you modify the values in the 2D string array for the header text?
   - ☐ What is the effect on the log file if you remove the Transpose 2D array node?
   - ☐ What is the effect on the VI if you remove the filepath input of the first Write Delimited Spreadsheet node?

On the Job

1. Do any of your applications require logging data from an N-channel N-sample finite acquisition?
2. If so, how many channels? _____

   How many samples per channel? _____

   List the column header names below.

End of Exercise 10-1

Exercise 10-2 Using Low-Level File I/O Nodes to Stream Data to a Text File

Goal

Stream single channel, single sample temperature data to text file.

Instructions

1. Open `<Exercises>\LabVIEW NXG Core 1\Low-Level Stream to Text File (1Chan 1Samp)\Low-Level Stream to Text File (1Chan 1Samp).lvproject`.

2. From the Project Files tab, open **Stream to Text File – 1 channel 1 sample.gvi**.

3. Examine the VI.

   - ☐ Notice that the DAQmx Create Channel node configures the task to acquire temperature measurements from a thermocouple.

   - ☐ Notice that the DAQmx Timing node configures the task to acquire measurements continuously at a sample rate of 1 Hz.

   - ☐ Notice that the DAQmx Read node reads 1 sample from 1 channel as a single DBL data type.

   - ☐ How frequently does the While Loop execute?_____ times per second

     Answer: The While Loop executes 1 time per second. The DAQmx Read node reads one sample each iteration. The sample rate is 1 Hz, so the DAQ device acquires 1 sample per second. Therefore, the While Loop only executes one time per second because the DAQmx Read node must wait until a sample is available to read.

4. Modify the diagram, as shown in the following figure, to stream data to a text file.

1	**Open/Create/Replace File**—Specify the filepath of the data log file. In this exercise, leave the file path input unwired so that when the user runs this VI, this node will launch a file dialog window for the user to choose the filepath for the data log file.
2	**Concatenate Strings**—Use this node to create the header string with an End of Line constant.
3	**Write to Text File**—Use the first Write to Text File node to write the header to the first line of the file before the VI enters the While Loop.
4	**Format Into String**—Use this node to convert the numeric DBL data into a string data type because the Write to Text File node requires a string input. This node, as configured in this exercise, will format a DBL input of 5.123456789 into the string "5.123<end-of-line character>". Refer to the LabVIEW Help for more details on this node and the format string syntax.
5	**Write to Text File**—Use the second Write to Text File node to continuously write data to file inside the While Loop.
6	**Close File**—Closes the file. You should always close a file when you are done accessing the file in the VI.
7	**Close File**—Closes the file. You should always close a file when you are done accessing the file in the VI.

5. Save the VI and project.

6. Run the VI.

   ☐ On the panel, set the Thermocouple Channel control to a single thermocouple channel on your DAQ device (e.g. cDAQ1/Mod2:ai0).

   ☐ Click the Run button.

   ☐ Save the file as `<Exercises>\...\Low-Level Stream to Text File (1Chan 1Samp)\Stream Data (1Chan 1Samp).txt`.

- ☐ Let the VI run for about 10 seconds, so the VI can acquire and log approximately 10 temperature measurements.
- ☐ Click the Stop button.

7. Explore the text file.
   - ☐ In Windows Explorer, navigate to and double-click the data log file to view its contents.
   - ☐ Notice the file includes a header describing each channel.
   - ☐ Verify that the log file looks similar to the following figure.

8. Try opening the text file using Microsoft Excel.

Your Turn

Create log file with two column headers (“Current Temperature (deg C),” “Current Temperature (deg F)”) and two columns of data.

End of Exercise 10-2

Exercise 10-3 Stream Multi-Channel Acquisition Data to Text File

Goal

Explore examples that stream NChan1Samp and NChanNSamp acquisition data to file.

Guided Instruction

N Channels, 1 Sample (1D DBL) Streaming Example

1. Open `<Exercises>\LabVIEW NXG Core 1\Low-Level Stream to File (NChan)\Low-Level Stream to Text File (NChan).lvproject`.

2. From the Project Files tab, open **Low-Level Stream to Text File (NChan 1Samp).gvi**.

3. Explore the diagram.

Figure 10-8. Low-Level Stream to Text File (NChan 1Samp) VI diagram

1 **Array to Spreadsheet String**—This node is configured to create a spreadsheet string, which adds a delimiter between each array element and adds an end-of-line constant at the end of the string. In this VI, it turns the incoming string array into the following string:

```
Thermocouple 0 (deg C)<tab>Thermocouple 1 (deg C)<tab> Thermocouple 2 (deg C)<tab> Thermocouple 3 (deg C)<end-of-line>
```

2 **Array to Spreadsheet String**—Use this node to convert the 1D DBL array data into a string data type because the Write to Text File node requires a string input.

The first Array to Spreadsheet String node will create the following result.

The second Array to Spreadsheet String node, as configured in this exercise, would format a 1D DBL input to spreadsheet string, as shown in the following example.

4. Test the VI.

   ☐ Run the VI.

   ☐ In the file dialog, save the log file as `<Exercises>\...\Low-Level Stream to Text File (NChan)\Stream Data (NChan 1Samp).txt`.

   ☐ Click the Stop button after approximately 10 seconds.

   ☐ In Windows Explorer, open and explore the text file.

5. Use a probe to view wire values to better understand the functionality of the nodes and VI. You can also create indicators off of wires and view the indicator values on the panel, if you prefer.

N Channels, N Samples (2D DBL) Streaming Example

1. From the Project Files tab, open **Low-Level Stream to Text File (NChan NSamp).gvi**.

2. Explore the diagram.

Figure 10-9. Low-Level Stream to Text File (NChan NSamp) VI Diagram

1 **Array to Spreadsheet String**—This VI configures this node to create a spreadsheet string, which adds a delimiter between each array element and adds an end-of-line constant at the end of the string. In this example, it turns the incoming string array into the following string:
`Thermocouple 0 (deg C)<tab>Thermocouple 1 (deg C)<tab> Thermocouple 2 (deg C)<tab> Thermocouple 3 (deg C)<end-of-line>`
2 Transpose 1D Array
3 **Array to Spreadsheet String**—Use this node to convert the 2D DBL array data into a string data type because the Write to Text File node requires a string input.

☐ The first Array to Spreadsheet String node, as configured in this exercise results in the following output.

☐ The Transpose 2D Array node rearranges the elements of the array similar to the following figure.

☐ The Second Array to Spreadsheet String node, as configured in this exercise, would format the output of the Transpose 2D Array node to a spreadsheet string, as shown in the following figure.

3. Test the VI.
   - ☐ Run the VI.
   - ☐ In the file dialog, save the log file as `<Exercises>\LabVIEW NXG Core 1\Low-Level Stream to Text File (NChan)\Stream Data (NChan NSamp).txt`.
   - ☐ Click the **Stop** button after approximately 10 seconds.
   - ☐ In Windows Explorer, open and explore the text file.
4. Use a probe to view wire values to better understand the functionality of the nodes and VI. You can also create indicators off of wires and view the indicator values on the panel, if you prefer.

On the Job

Answer the following questions for your own applications.

1. Do you need to continuously stream multi-channel data to a text file?

2. If so, which of the above examples is closest to meeting your needs?

   N Channels, 1 Sample (1D DBL) Streaming Example

   OR

   N Channels, N Samples (2D DBL) Streaming Example

3. What modifications do you need to make to these examples to meet your needs?

End of Exercise 10-3

Exercise 10-4 Programmatically Create Filenames Based on Current Timestamp

Goal

Explore example code that programmatically creates filenames based on the current date and time.

Examine Code to Generate a Timestamped Filename in Relative Path

1. Open `<Exercises>\LabVIEW NXG Core 1\Timestamped Filepath\Timestamped Filepath.lvproj`.
2. From the Project Files tab, open **Generate Timestamped Filepath.gvi**.
3. Examine the diagram.

Figure 10-15. Generate Timestamped Filepath VI Diagram

1 **Get Current Time**—Returns a timestamp of the current time.
2 **Format Date and Time String**—Converts a timestamp or numeric value into a string that displays the corresponding time. Using the time format string in this diagram, this node will output a string similar to the following

```
<year><month><day>_<hour><minute><second>_Acquired Data.txt
```

Example: **`20170601_101530_Acquired Data.txt`**
3 **Get Component Directory**—Returns the path to the directory containing the current project (.lvproject)
4 **Build Path**—Creates a new path by appending a filename or relative path to an existing path. In this VI, the output of this node will be a timestamped filename in the directory containing the current project. (for example, `C:\Exercises\LabVIEW NXG Core 1\Timestamped File\20170601_101530_Acquired Data.txt`)

Note Refer to the help for how to use the time format string input.

4. Test the code.

   - ☐ Run the VI multiple times.
   - ☐ On the panel, notice that the generated filename changes to match the current date and time.
   - ☐ Notice that the filepath directory is the same as the directory containing the current project file (.lvproject).

Examine Timestamped Filename Code in a Data Logging VI

1. From the Project Files tab, open **[Timestamped File] High-Level Write to Text File.gvi**.

2. Examine the diagram. Notice how the timestamped filename code passes the timestamped filepath to the file path input of the file I/O node.

Figure 10-16. [Timestamped File] High-Level Write to Text File VI Diagram

3. Test the VI.

   - ☐ Run the VI.
   - ☐ Navigate to the log file in the `<Exercises>\LabVIEW NXG Core 1\Timestamped File` directory.
     Notice that the filename indicates the date/time when the VI created that file.
   - ☐ Run the VI a couple more times.
   - ☐ Examine the additional log files that the VI generated in the `<Exercises>\LabVIEW NXG Core 1\Timestamped File` directory.
     Notice that their filenames also indicate the date/time when the VI created those files.

Your Turn

Modify the VI so that the VI creates a timestamped filename that looks similar to the following format and saves the file in the `<Exercises>\LabVIEW NXG Core 1\Timestamped File` directory.

```
<year><month><day>_<hour><minute><second>_<your own custom filename>
```

Example: 20161201_090030_Batch ABC.txt

On the Job

Would any of your applications benefit from creating a new timestamped filename every time you run your VI?

End of Exercise 10-4

Exercise 10-5 Read and Analyze Data from Text File in LabVIEW NXG

Goal

Read and analyze data from file in LabVIEW NXG.

Instructions

1. Examine the text file containing the data that we want to read into LabVIEW NXG.
   - ☐ Open and examine `<Exercises>\LabVIEW NXG Core 1\Read Delimited File\Acquired Data (NChan NSamp).txt`.
   - ☐ Notice that the first row is a header.
   - ☐ Notice that the text file contains 4 columns of data. Each column represents a different channel.
2. Open `<Exercises>\LabVIEW NXG Core 1\Read Delimited File\Read Delimited File into VI.lvproject`.
3. From the Project Files tab, open **Read Delimited File into VI.gvi**.

4. Examine the diagram.

Figure 10-18. Read Delimited File into VI Diagram

1 **Get Component Directory**—Returns the path to the directory containing the current project (.lvproject)
2 **File Dialog**—Displays a dialog box with which a user can specify the path to a new or existing file or directory.
3 **Read Delimited Spreadsheet (string)**—Returns the contents of the delimited text file as a 2D string array.
4 **Index Array**—Extract the first row, which contains a string for each column header.
5 **Read Delimited Spreadsheet (DBL)**—Returns the contents of the delimited text file as a 2D DBL array.
6 **Array Subset**—The first line in this tab-delimited text file is a header. Use this function to remove the first line, so that the array output only contains the numeric data.
7 **Transpose 2D Array**—Transposes the 2D array. The output 2D array contains one row for every channel.

Note If you wire a 2D DBL array to a graph, the graph will interpret each row of the 2D DBL array as a separate plot, which is why you must transpose the 2D array in this VI.

5. Examine the functionality of the VI.

   ☐ Run the VI.

   ☐ In the file dialog, navigate to `<Exercises>\LabVIEW NXG Core 1\Read Delimited File\Acquired Data (NChan NSamp).txt`.

   ☐ Use Highlight Execution and Probes to examine the functionality of this VI.

Your Turn

1. Open and examine `<Exercises>\LabVIEW NXG Core 1\Read Delimited File\Acquired Data (NChan NSamp).txt`.

2. Modify the diagram to process the multi-channel data that this VI has read from the text file.

   ☐ Display the minimum and maximum values of each channel.

   ☐ Your final panel should look similar to the following figure.

Figure 10-19. "Your Turn" Solution VI Panel

End of Exercise 10-5

Exercise 11-1 Create and Use a SubVI

Goal

Create the icon and connector pane for a VI so that you can use the VI as a subVI.

Scenario

You will explore a VI that generates a timestamped filepath in the same directory containing the project. Create an icon and a connector pane so that you can use this VI as a subVI.

Design

The Generate Timestamped Filepath VI contains the following inputs and outputs.

Inputs	Output
Appended filename	timestamped relative filepath
Error In	Error Out

Create SubVI

1. Open `<Exercises>\LabVIEW NXG Core 1\Create SubVI\Create and Use SubVI.lvproject`.
2. From the Project Files tab, open **Support»Generate Timestamped Filepath.gvi**.

Note It is common practice to place subVIs and support files in a subdirectory and place the main top-level VI in the parent directory. This allows the main top-level VI to be more visible and accessible to the user.

3. Place an **Error In** control and **Error Out** indicator on the panel.

Figure 11-11. Generate Timestamped Filepath SubVI Panel

4. Modify the diagram, as shown in the following figures, to add a Case structure to handle errors passed to this subVI.

   - ☐ Place a Case structure around the code.
   - ☐ Drag the Error In control and Error Out indicator from the Unplaced Items tray to the diagram.
   - ☐ Wire the Error In control to the Selector terminal of the Case structure.

☐ If necessary, swap the code in a case by right-clicking the Case structure border and selecting **Cases»Make This Case No Error** or **Cases»Make This Case Error**.

Figure 11-12. No Error Case

Figure 11-13. Error Case

5. Connect the inputs and outputs to the connector pane, as shown in Figure 11-14.

Figure 11-14. SubVI—Icon and Connector Pane

☐ In the upper-right corner of the VI toolbar, click the **Edit Icon** button.

☐ Resize the number of input and output terminals.

☐ Click and assign the input and output terminals.

☐ Select the **appended filename** input terminal.

☐ In the Item tab, set this terminal's usage to **Required**.

Now when you use this subVI in a main VI, the main VI Run button will be broken until the appended filename input terminal is wired.

6. Create an icon, as shown in Figure 11-14.

   ☐ Click the search icon in the left palette, and search `timestamp` and `path` to place the corresponding graphics onto the VI icon. You can resize the graphics.

   ☐ Click the **Done** button below the VI toolbar when finished.

7. Create a description for the subVI.

   ☐ In the VI Description section of the Document tab, enter `Generates a timestamped filepath with an appended user-specified name and the same directory containing the project.`

8. Save the VI.

9. Test the VI when Error In contains no error.

   ☐ On the panel, set the appended filename control to a string value, such as `MyData`.

   ☐ Verify that the Error In control has a status of False.

   ☐ Run the VI.

   ☐ Notice that the VI returns a timestamped relative filepath similar to `C:\Exercises\LabVIEW NXG Core 1\Create and Use SubVI\20170530_093000_MyData.txt`.

10. Test the VI when Error In contains an error.

    ☐ Set the status element of the Error In control to TRUE. Set the code element to `-1`.

    ☐ Run the VI.

    Notice that the VI returns an empty path in the timestamped relative filepath indicator. Notice that the error information is passed to the Error Out indicator.

Use the SubVI in a Main VI

1. From the Project Files tab, open **Main.gvi**.

2. Examine the Open/Create/Replace File node on the diagram.

   Notice that its file path input is unwired, which means when you run the VI, this node will launch a file dialog for the user to specify the filepath.

3. Use the Generate Timestamped Filepath subVI to output a timestamped filename in the same directory as the current .lvproject file.

   ☐ Drag the **Generate Timestamped File.gvi** item from the Project Files tab to the diagram of Main VI.

   ☐ Right-click the appended filename input and select **Create Constant**. Set the constant to string value, such as `MyData`.

   ☐ Wire the timestamped relative filepath output to the Open/Create/Replace File node.

Figure 11-15. Main VI Diagram—Add Generate Timestamped Filepath SubVI

4. View the subVI description.

   ☐ Open the Context Help window by clicking <Ctrl-H>.

   ☐ Hover your mouse cursor over your subVI. Notice that the Context Help window populates with the text that you previously entered for the subVI's VI Description.

5. Examine the behavior of the VI.

   ☐ Set the Thermocouple Channel to an appropriate channel.

   ☐ Run the VI. After a few seconds stop the VI.

   ☐ In Windows Explorer, go to the `<Exercises>\...\Create and Use SubVI` directory and notice the timestamped file created by the VI.

   ☐ Run the VI a couple more times. Notice the additional timestamped files created by the VI.

6. Now you can reuse the Generate Timestamped Filepath subVI in other VIs where you want the same functionality.

Your Turn

Create a subVI that reads in a voltage measurement, converts it to units of acceleration (g) by dividing it by 0.175, and outputs the measurement in units of acceleration (g).

The following figure shows how your completed "Convert Voltage to Acceleration" subVI could be used in an example VI.

Figure 11-16. Example VI with Convert Voltage to Acceleration SubVI

Note For the answer, refer to `<Solutions>\LabVIEW NXG Core 1\Exercise 11-1 (Your Turn)` directory.

On the Job

Is there code you will commonly reuse in your applications? Would the application benefit from putting that code in a subVI?

Do you have any code that uses a lot of nodes but accomplishes one modular task (for example, algorithm)? Would the top-level application be more readable if that code was put inside a subVI?

End of Exercise 11-1

Exercise 12-1Group Related Data Using a Cluster

Goal

- Group related data into a new cluster.
- Create subVI with cluster input/output.
- Pass cluster data into/out of a subVI from a Main VI.

Scenario

Group weather-related data into a Weather Data cluster.

Another developer has created a VI that displays temperature warnings. This VI is part of a temperature weather station project. Your task is to update this VI to use clusters instead of individual controls/indicators for inputs and outputs.

Guided Instruction

Determine Warnings SubVI

In this section, you will create a subVI that handles temperature warnings. All the weather-related data is grouped together in a Weather Data cluster.

1. Open `<Exercises>\LabVIEW NXG Core 1\Clusters\Weather Warning Cluster\Weather Warnings.lvproject`.

2. From the Project Files tab, open **Determine Warnings.gvi**.

3. Place existing controls and indicators in a cluster named `Weather Data`, as shown in the following figure.

1	**Cluster**—Use the Cluster control from the Clusters palette and change the label to `Weather Data`.
2	Select controls and indicators to include in the cluster. <Shift>-click to select multiple objects.
3	Drag the controls and indicators into the **Weather Data** cluster.

4. Resize the cluster so that all the elements are visible and arranged vertically, as shown in the following figure.

1	Autosize cluster—LabVIEW NXG can rearrange and resize the cluster for you. Right-click the border of the **Weather Data** cluster and select **Arrange»Vertical**.

5. Reorder the items in the cluster, as shown in the following figure.

1	Go to the configuration pane of the cluster. To reorder the cluster elements, hover to the left of a data item and click and drag the item to the appropriate order. Verify that your **Weather Data** cluster order matches the order in this figure.

6. Modify the VI to receive and return cluster data.

1	**Weather Data**—<Ctrl>-click the **Weather Data** cluster and drag it to create a copy. Rename the copy `Weather Data In`.
2	**Weather Data**—Right-click the original cluster and select **Change to Indicator**. Rename the indicator `Weather Data Out`.

7. Modify the diagram, as shown in the following figure, to extract and modify data from the input cluster.

1 Drag the **Weather Data In** control and **Weather Data Out** indicator from the Unplaced Items palette to the diagram.
2 **Cluster Properties**—Wire the **Weather Data In** control and expand the Cluster Properties node to display three items. Wire the outputs of the Cluster Properties node to the broken wires in the order shown. Because you moved individual controls and indicators into a single cluster, you must use the Cluster Properties node to access each cluster element.
3 **Cluster Properties**—Wire the **Weather Data In** cluster around the analysis code to the input cluster of the Cluster Properties node. Display two elements and click to select the **Warning?** and **Warning Text** elements. Right-click the top of the node and select **Set All to Write**, so you can modify these element values. Connect the broken wires to the Cluster Properties inputs as shown.

Note If the order of the elements in the Cluster Properties nodes is different than what you want, you can click the elements to change the order.

8. Save the Determine Warnings VI.

Test

1. Enter values in the Current Temperature, Max Temperature and Min Temperature controls in the Weather Data In cluster.
2. Run the VI and verify that the Weather Data indicator displays correct values.
3. Save and close the VI.

Explore Another VI Using the Weather Data Cluster

Main VI

In this section, you explore a full example that uses the weather data cluster in multiple places.

1. Open `<Exercises>\LabVIEW NXG Core 1\Clusters\Full Weather Warnings Example\Full Weather Warnings Example.lvproject`.
2. From the Project Files tab, open **Main.gvi**.

3. Examine the diagram. Notice the following places where VI uses the Weather Data cluster data type.

1 Weather Data cluster constant as input to the 1st **Cluster Properties** node
2 Output of 1st **Cluster Properties** node
3 Input and output terminal of **Determine Warnings subVI**
4 Weather Data indicator
5 Input and output of 2nd **Cluster Properties** node
6 Input terminal of **Log Warnings subVI**

Note Usually clusters should also be made into a G type, so that if you need to modify the cluster data type (e.g. add another item to the cluster), you only need to make the change to the G type and the change will automatically propagate to all instances of that cluster. You will learn more about this in the G Types lesson.

Your Turn

Log SubVI

How would you modify the following subVI diagram to use a cluster instead of 5 individual controls?

Note To view the solution, open the Log Warnings subVI in the Full Weather Warnings Example.lvproject.

On the Job

1. In your own applications, what data should you group into a cluster? Sketch your potential cluster data types below.

End of Exercise 12-1

Exercise 13-1 Using a G Type

Goal

- Create weather data cluster G Type
- Place G Type in several places (e.g. subVI input/output, main VI controls/indicators)
- Update G Type and save to propagate data type changes to all instances of the G Type.

Scenario

As a LabVIEW developer, you can encounter situations where you need to define your own custom data types in the form of clusters and enums. A challenge associated with using custom data types is that you may need to change them later in development. In addition, you may need to change them after they have already been used in VIs. For example, you create copies of a custom data type and use them as controls, indicators, or constants in one or more VIs.

Then you realize that the custom data type needs to change. You need to add, remove, or change items in the cluster data type or the enum.

As a developer you must ask yourself the following questions:

- What should happen to the copies of the custom data types used in VIs that are already saved?
- Should the copies remain unchanged or should they update themselves to reflect changes to the original?

Usually, you want all the copies of the custom data type to update if you update the original custom data type. To achieve this you need copies of the custom data types to be tied to a G Type, which is defined as follows:

G Type—A master copy of a custom data type that multiple VIs can use.

Implementation

In this exercise, you modify the Determine Warnings subVI in such a way that the changes to the Weather Data G Type propagate through the application.

When complete, the Weather Station application monitors temperature and wind information. This exercise modifies the Determine Warnings subVI.

1. Open `<Exercises>\LabVIEW NXG Core 1\G Types\G Types.lvproject`.
2. From the Project Files tab, open **Determine Warnings.gvi**.
3. Experiment with changing an existing cluster.
   - ☐ Place a **File Path Control** in the **Weather Data In** cluster control.
   - ☐ Notice that the Determine Warnings VI is broken. This is because the **Weather Data In** and **Weather Data Out** clusters are no longer the same data type.

- ☐ Open the diagram and notice the broken wire connected to the **Weather Data Out** terminal.
- ☐ Press <Ctrl-Z> to undo the addition of the File Path Control.

4. Make a G Type.
   - ☐ On the panel, right-click the border of the **Weather Data In** control and select **Change to G Type**.

     Notice in the Project Files tab that this has created a new `.gtype` file on your hard drive.
   - ☐ On the diagram, the **Weather Data In** terminal now has a black vertical bar on the edge indicating that it is connected to a G Type.

   - ☐ Right-click the **Weather Data In** control and select **Open G Type** to display the G Type in a new tab.

     The G Type tab looks like the panel of a VI but it does not have a diagram.
   - ☐ Rename the cluster `Weather Data`.
   - ☐ In the Project Files tab, rename the gtype file `Weather Data.gtype`.
   - ☐ Move the gtype into the `support` directory by clicking and dragging the file.
   - ☐ Close the gtype tab when finished.
   - ☐ On the diagram of the Determine Warnings VI, notice the coercion dot on the **Weather Data Out** indicator terminal. This indicates that the indicator is not tied to the G Type.

5. Replace the Weather Data Out indicator with the G Type you just created.
   - ☐ On the panel of the Determine Warnings VI, delete the Weather Data Out indicator.
   - ☐ Drag the **Weather Data.gtype** item from the Project Files tab onto the panel.
   - ☐ Right-click the border of the Weather Data G Type on the panel and select **Change to Indicator**. Rename the indicator as `Weather Data Out`.
   - ☐ Go to the diagram and notice that the Weather Data Out terminal has a black vertical bar on the edge, which indicates that it is connected to a G Type.
   - ☐ Update the VI connector pane to include the Weather Data Out indicator because the indicator you previously deleted had been connected to the connector pane.
     - On the VI toolbar, click the **Edit icon** button.
     - Connect the second-from-the-top terminal on the right side to the Weather Data Out indicator, as shown in the following figure.

Figure 13-4. Determine Warnings SubVI Connector Pane

Note You can no longer add or remove elements to or from a G Type cluster control/indicator on the panel. You must open and edit the gtype file to add or remove the element.

☐ Save the Determine Warnings subVI.

6. Edit the Weather Data G Type to include wind speed information.

☐ Right-click the border of the **Weather Data In** control and select **Open G Type** from the shortcut menu.

☐ Modify the panel as shown in the following figure.

1	Numeric Control—Place a numeric control in the cluster and rename it as `Current Windspeed`

☐ Save Weather Data.gtype and close its tab.

☐ On the Determine Warnings VI, notice that the **Weather Data In** control and **Weather Data Out** indicator have been updated with the changes you made to the Weather Data type G Type. Arrange the panel of the VI as shown in the following figure.

7. Run and save the Determine Warnings subVI.

Your Turn

1. In this project, update **Main.gvi** and **Log Warnings.gvi**. Replace all instances (listed below) of the cluster in these VIs to use the Weather Data G Type.

   ☐ Weather Data instances in Log Warnings.gvi

   – Weather Data In cluster

   Note Make sure you reconnect the new G Type to the connector pane

   ☐ Weather Data instances in Main.gvi

   – Weather Data cluster constant

   – Weather Data indicator

2. Modify the Weather Data G Type to include a numeric **Max Windspeed** element and a Boolean **Windspeed Warning?** element. Rename the original Warning? element as `Temperature Warning?`.

3. Verify that all instances of the G Type in all VIs update to include the new elements.

Your Turn

1. Create a new project and VI.
2. Create an enum that contains 3 items (e.g., Add, Subtract, Multiply).
3. Convert the enum into a G Type.
4. Add the G Type to multiple places and VIs.
5. Modify the enum G Type to include a fourth item (for example, Divide). Save the G Type.
6. Verify that all instances of the enum G Type update to include the new element.

On the Job

1. List all the clusters and enums that will exist in your own application.

2. Should you make these clusters and enums into a G Type?

   If the cluster or enum will exist in more than one place, you should definitely make it into a G Type.

End of Exercise 13-1

Exercise 14-1 Create a State Machine

Goal

Create a VI that implements a state machine using a G Type enum.

Scenario

You must design a VI for a user interface state machine. The VI acquires a temperature every half second, analyzes each temperature to determine if the temperature is too high or too low, and alerts the user if there is a danger of heatstroke or freeze. The program logs the data if a warning occurs. If the user has not clicked the stop button, the entire process repeats. The state machine must also allow for expansion, because processes may be added in the future.

Design

Use a state machine to create the VI in this exercise. The state transition diagram in the following figure describes the logic for this application.

Figure 14-1. Weather Station VI State Transition Diagram

The following table describes the states in this state machine.

State	Description	Next State
Acquire	Set time to zero, acquire data from the temperature sensor	“Analyze”
Analyze	Read panel controls and determine warning level	“Log” if a warning occurs “Check Time” if no warning occurs
Log	Log the data in a tab-delimited ASCII file	“Check Time”
Check Time	Check whether time elapsed is greater than or equal to 0.5 seconds	“Acquire” if time has elapsed “Check Time” if time has not elapsed

Implementation

1. Open `Weather Station.lvproject in the <Exercises>\LabVIEW NXG Core 1\State Machine` directory.

2. Open **Weather Station UI.gvi** from the Project Files tab.

   Figure 14-2 shows the panel of the Weather Station UI VI that has been provided for you. You modify the diagram to create a state machine for the Weather Station.

Figure 14-2. Weather Station UI VI Panel

The following figure shows the starting point of the diagram for the Weather Station UI VI. You edit this diagram to implement a state machine for the Weather Station application.

Figure 14-3. Weather Station UI VI Diagram (Starting)

1 Drag the controls and indicators from the Unplaced Items tray. You use these controls and indicators to program different cases.

3. Create a new G Type to control the Weather Station application.

   ☐ Open the diagram and create an Enum constant to the left of the While Loop.

   ☐ In the configuration pane, add the enum items, as shown in the following figure.

   ☐ Right-click the enum constant on the diagram and select **Change to G Type**.

4. Modify the new G Type and add it to the Weather Station project.

   ☐ Right-click the enum constant and select **Open G Type**.

   ☐ Rename the label on the **Enum** control to `States`.

- ☐ Save the G Type as `Weather Station States.gtype` in the `<Exercises>\LabVIEW NXG Core 1\State Machine\Supporting Files` directory.
- ☐ Close the G Type tab.
- ☐ In the Project Files tab, notice that Weather Station States.gtype has been added to your `Supporting Files` folder.

5. Place a Case structure inside the While Loop, as shown in the following figure. This is part of the state machine design pattern.

Figure 14-4. Weather Station UI VI Diagram—Add Case Structure

6. Control the state machine with the G Type enum and update the framework as shown in the following figure.

Figure 14-5. Weather Station UI VI Diagram—State Enum and State Data Shift Registers

1 **Shift Register**—Right-click the While Loop and select **Create Shift Register**.
2 **Enum G Type** constant—In the configuration pane, enable the **Show Label** checkbox. Rename the label `Beginning State`. Wire the Beginning State constant to the shift register to initialize the shift register to the Acquire state. Wire the shift register to the Selector input of the Case Structure.
3 Add more cases—Right-click the Case structure and select **Cases»Create Case For Every Value** to create different cases for each value in the enum.
4 **Weather Data In**—Drag Weather Data.gtype from the Project Files tab to the diagram to create a G Type cluster constant.
5 **Shift Register**—Place another shift register on the While Loop and wire the Weather Data constant to it.
6 Wire as shown.

Note After you finish wiring the Acquire case in the following step, some tunnels are empty because not all cases are wired yet.

7. Complete the Acquire state shown in the following figure.

Figure 14-6. Weather Station UI VI—Acquire State

1 **DAQmx Read**—In the configuration pane, set the following values:
Channel Type: Analog Input
Data Format: Floating Point
Channel Count: Single Channel
Sample Count: Single Sample

2 **Temperature History**—Move this indicator from the Unplaced Items palette into the Acquire state of the Case structure.

3 **Cluster Properties**—Set the element to Current Temperature. Right-click the element and select Change to Write. Wire the data output of the DAQmx Read node to the Current Temperature input.

4 **Next State enum**—<Ctrl>-click the Beginning State enum and drag a copy into the Acquire case. Rename this copy of the Weather Station States GType Next State. Set the enum to Analyze and wire it through a tunnel on the Case structure to the shift register on the While Loop.

5 **True Constant**—Create a True constant and wire it through the Case structure to the Elapsed Time shift register. The True constant resets the Elapsed Time counter every time the VI executes the Acquire case.

6 Wire the DAQmx task, File reference, Weather Data cluster, enum, error, and Boolean wires through the diagram to their corresponding shift register as shown.

8. Complete the Analyze case as shown in the following figure.

Figure 14-7. Weather Station UI VI—Analyze Case

1 **Upper Limit and Lower Limit**—Move these controls from the Unplaced Items palette to the diagram.
2 **Cluster Properties**—Replaces the Max Temperature and Min Temperature items with the values from the Upper Limit and Lower Limit controls. The Cluster Properties node makes it possible to wire the Upper Limit and Lower Limit values to the Weather Data In input of the Determine Warnings subVI.
3 **Determine Warnings SubVI**—From the Project Files tab, drag **Determine Warnings.gvi** to the diagram.
4 **Cluster Properties**—Returns the value of specific items from the cluster.
5 **Select**—Determines which state to execute next depending on whether or not a warning occurs.
6 **Weather Station States**—Wire two copies of the Weather Station States G Type to the Select node. You can create these copies from the Beginning State enum.
7 **Warning**—Move this indicator from the Unplaced Items palette to the diagram.
8 Wire the DAQmx task and file reference wires through the case as shown

9. Complete the Log case as shown in the following figure.

Figure 14-8. Weather Station UI VI—Log Case

1 **Log Warnings subVI**—From the Project Files tab, drag **Log Warnings.gvi** to the diagram. This subVI logs weather data to file.
2 **Next State**—Create a copy of the Weather Station States enum, label it `Next State`, and set the next state to Check Time. Wire the DAQmx task and Weather Data cluster wires through the case as shown.

10. Complete the Check Time case as shown in the following figure.

Figure 14-9. Weather Station UI VI—Check Time Case

1 **Next State**—Wire two copies of the Weather Station States G Type to the Select node.
2 **Select**—Determines which state to execute next depending on whether or not time has elapsed.
3 **Wait**—In the configuration pane, set Units to **Milliseconds**. This node allows the CPU time to do other tasks while this VI repeatedly executes the Check Time case until the Elapsed Timer node outputs a TRUE value when 0.5 seconds has elapsed.
4 **Stop** button—Move the Stop button terminal from outside the While Loop. Wire the Stop Button terminal to the Or node outside of the Case structure.
5 **Default if unwired**—Right-click these tunnels and select **Default If Unwired**. These Boolean tunnels will output a value of False in cases where these tunnels are unwired.

Wire the DAQmx task, file reference, and Weather Data cluster wires through the case as shown.

11. Save and test the VI.

Test

1. Run the VI.
   - ☐ Name the log file when prompted.
   - ☐ Enter values for the **Upper Limit** and **Lower Limit** controls and observe the behavior of the VI. Does it behave as expected?
2. Stop the VI.
3. Navigate to the `Weather Warning Log.txt` file and open it.
4. Notice the changes in the upper and lower limit values and the placement of tabs and line breaks.
5. Close the log file.
6. Save and close the VI and the project.

On the Job

1. Back at your job, can your application's logic be described by a state transition diagram? If so, then draw the state transition diagram below:

2. Will you use a state machine to implement your state transition diagram in LabVIEW NXG code?

End of Exercise 14-1