LabVIEW™ Core 1 Participant Guide

Course Software Version LabVIEW 2019
2019 Edition
Part Number 327415A-01 377415A-01

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.

Table of Contents

Student Guide

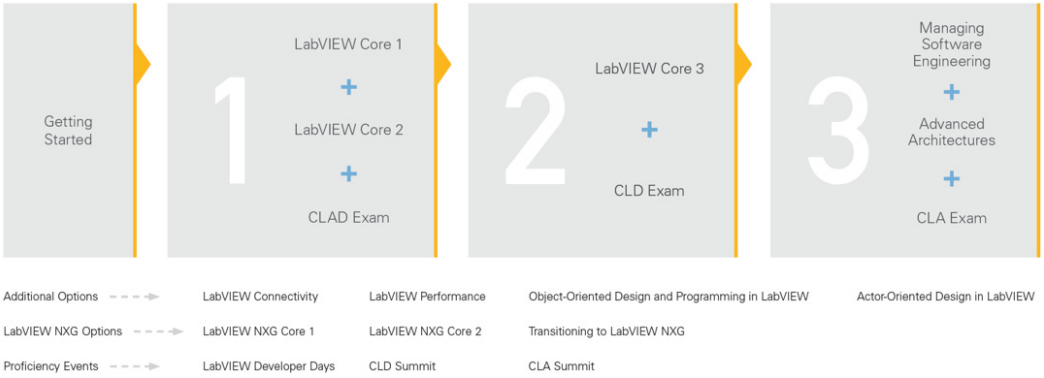
In this student guide, you will learn about the LabVIEW Learning Path, the course description, and the items you need to get started in the LabVIEW Core 1 course.

Topics

+ NI Certification
+ Course Description
+ What You Need to Get Started
+ Installing the Course Software
+ Course Goals

A. NI Certification

The *LabVIEW Core 1* course is part of a series of courses designed to build your proficiency with LabVIEW and help you prepare for the NI Certified LabVIEW Associate Developer exam. The following illustration shows the courses that are part of the LabVIEW training series. Refer to `ni.com/training` for more information about NI Certification.



B. Course Description

The *LabVIEW Core 1* course teaches you programming concepts, techniques, features, VIs, and functions you can use to create test and measurement, data acquisition, instrument control, datalogging, measurement analysis, and report generation applications. This course assumes that you are familiar with Windows and that you have experience writing algorithms in the form of flowcharts or block diagrams.

The Participant Guide is divided into lessons. Each lesson contains the following:

- An introduction with the lesson objective and a list of topics and exercises.
- Slide images with additional descriptions of topics, activities, demonstrations, and multimedia segments.
- A set of exercises to reinforce topics. Some lessons include optional and challenge exercises.
- A lesson review that tests and reinforces important concepts and skills taught in the lesson.

If you do not have hardware, you still can complete the exercises. Alternate instructions are provided for completing the exercises without hardware. You also can substitute other hardware for those previously mentioned. For example, you can use another National Instruments DAQ device connected to a signal source, such as a function generator.

C. What You Need to Get Started

Before you use this course manual, make sure you have all of the following items:

- Computer running Windows 7 or later
- LabVIEW 2019 or later
- NI-DAQmx 19 or later
- NI-VISA 18.5 or later
- NI-488.2 18.5 or later
- NI Instrument Simulator and power supply
- Real or simulated NI PCI 6221 or NI USB 6212, with BNC-2120.
- Downloaded course materials.

D. File Locations

After extracting the exercises and solutions folders we recommend placing them in the Local Disk (C:).

Note Folder names in angle brackets, such as `<Exercises>` and `<Solutions>`, refer to `C:\Exercises\` and `C:\Solutions\` respectively, assuming that you have copied the files to the root directory of your computer.

E. Course Goals

This course prepares you to do the following:

- Acquire data from hardware
- Analyze and process data
- Visualize and log data
- Develop, debug, and test LabVIEW programs
- Use best practices for reusability, readability, and error management
- Implement a sequencer using a state machine

2 First Measurement (NI DAQ Device)

In this lesson, you acquire and validate your first measurement from an NI DAQ device.

Exercises

Exercise 2-1:Using NI MAX to Examine a DAQ Device

Goal

Use NI Measurement & Automation Explorer to examine, configure, and simulate NI hardware.

Description

You want to create an application using NI Hardware.

In this exercise, you will explore how to configure the NI DAQ device, or simulate the NI DAQ device if you don't have the physical hardware.

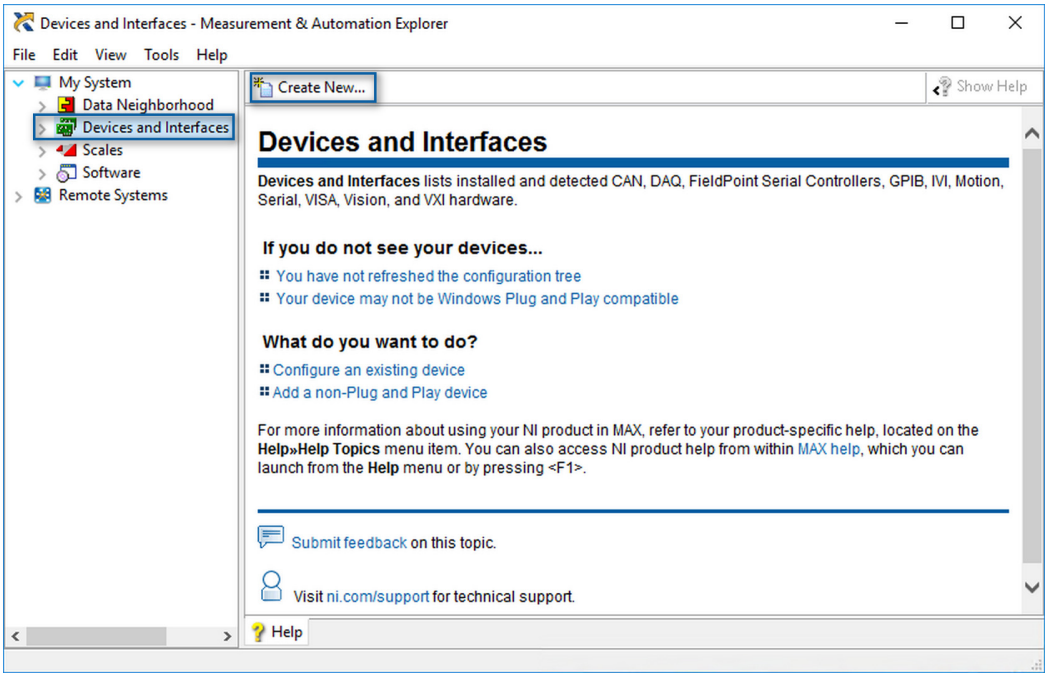
Implementation

Complete the steps in this section to examine the configuration for the DAQ device using NI MAX. If you do not have a DAQ device, simulate the NI PCI-6221 multifunction I/O device using the instructions in step 3. You will use this device to complete the course exercises.

Note Portions of this exercise can only be completed with the use of a real device and a BNC-2120, shown in the figure. Some of these steps have alternative instructions for simulated devices.

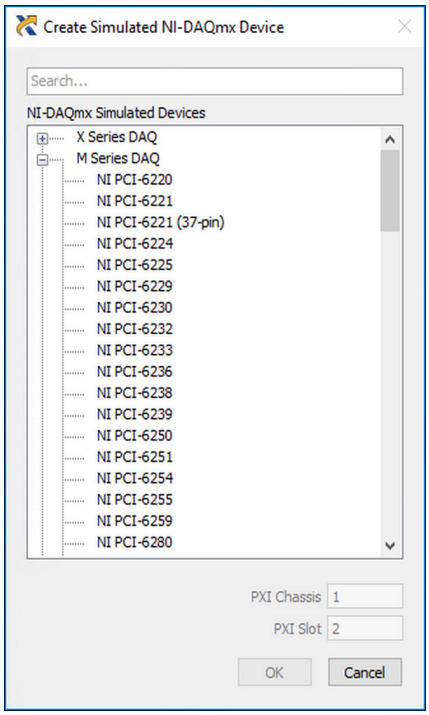
1. Launch **NI MAX** from the Windows **Start** menu.
2. If you have a DAQ device installed, skip step 3 and go to the Examine the DAQ Device Settings section.
3. Create simulated hardware.

   ☐ In the NI MAX, select **Devices and Interfaces**, and click the **Create New** button.



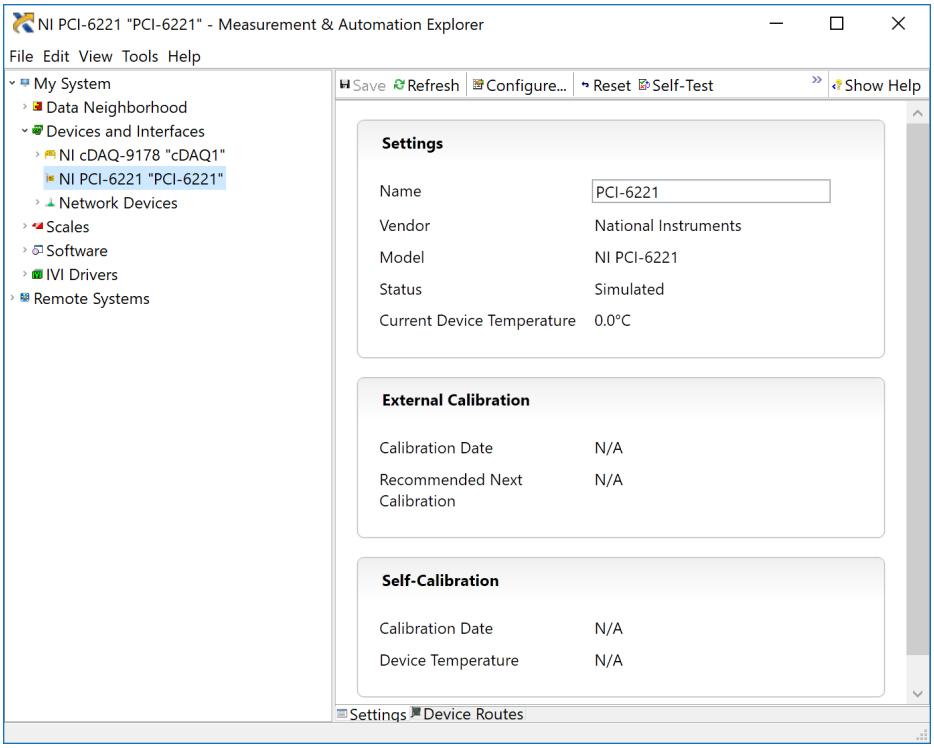
☐ In the **Create New** dialog box, select **Simulated NI-DAQmx Device or Modular Instrument**, and click **Finish**.

☐ Expand the **M Series DAQ** tree, and select **NI PCI-6221**.



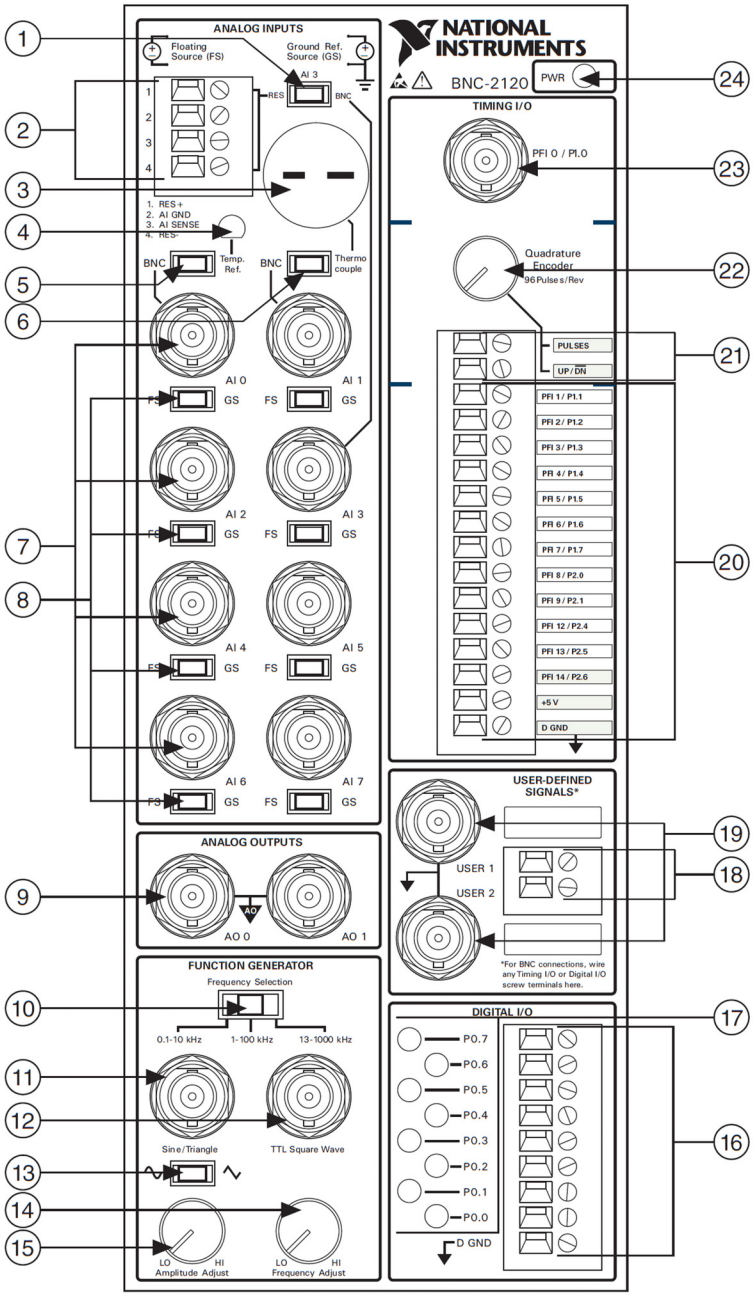
☐ Click OK.

☐ Select the simulated device and name it `PCI-6221` under the **Settings** section.


NI PCI-6221 "PCI-6221" - Measurement & Automation Explorer
File Edit View Tools Help
My System
Data Neighborhood
Devices and Interfaces
NI cDAQ-9178 "cDAQ1"
NI PCI-6221 "PCI-6221"
Network Devices
Scales
Software
IVI Drivers
Remote Systems
Save Refresh Configure... Reset Self-Test Show Help
Settings
Name PCI-6221
Vendor National Instruments
Model NI PCI-6221
Status Simulated
Current Device Temperature 0.0°C
External Calibration
Calibration Date N/A
Recommended Next Calibration N/A
Self-Calibration
Calibration Date N/A
Device Temperature N/A
Settings Device Routes

BNC-2120 Terminal Block

A terminal block consists of screw or spring terminals for connecting signals or other sensors. A cable transports the signal from the terminal block to the DAQ device.



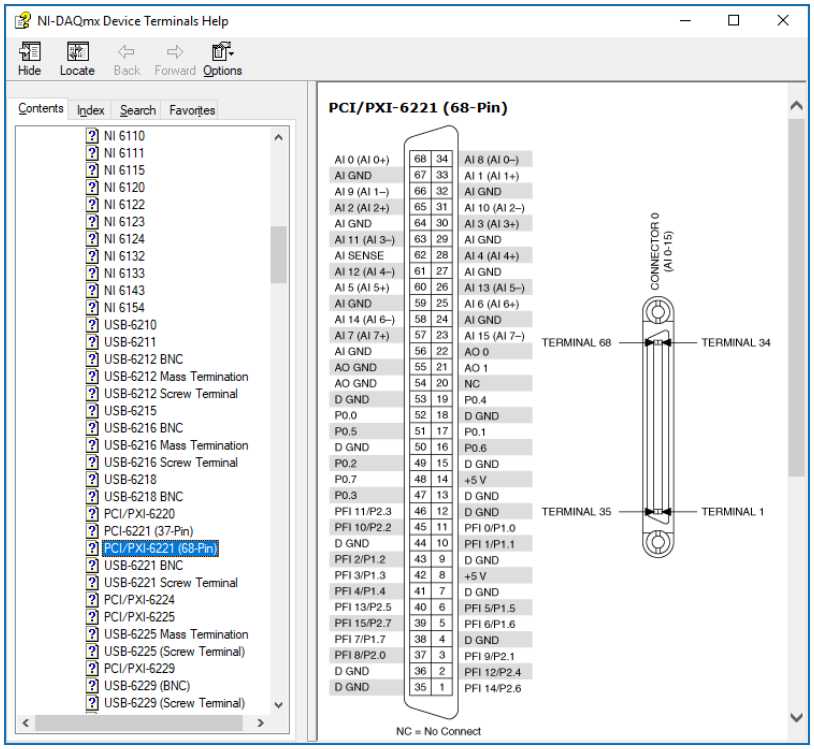
1	RES/BNC Switch (AI 3)	13	Sine/Triangle Waveform Switch
2	Resistor Measurement Screw Terminals	14	Frequency Adjust Knob
3	Thermocouple Input Connector	15	Amplitude Adjust Knob
4	Temperature Reference	16	Digital I/O Screw Terminals
5	BNC/Temp. Ref. Switch (AI 0)	17	Digital I/O LEDs
6	BNC/Thermocouple Switch (AI 1)	18	User-Defined Screw Terminals
7	Analog Input BNC Connectors	19	User-Defined BNC Connectors
8	FS/GS Switches	20	Timing I/O Screw Terminals
9	Analog Output BNC Connector	21	Quadrature Encoder Screw Terminals
10	Frequency Range Selection Switch	22	Quadrature Encoder Knob
11	Sine/Triangle BNC Connector	23	Timing I/O BNC Connector
12	TTL Square Wave BNC Connector	24	Power Indicator LED

Examining the DAQ Device Settings

1. Expand the **Devices and Interfaces** section.
2. Select the device that is connected to your machine. Green icons represent real devices and yellow icons represent simulated devices. You might have a different device installed, and some of the options shown might be different.

   MAX displays National Instruments hardware and software in the computer. The device alias appears in quotes following the device type. The Data Acquisition VIs use this device alias to determine which device performs DAQ operations. MAX also displays the attributes of the device such as the system resources that the device uses.

   Make sure the device you use is named `PCI-6221`. To rename a device, right-click the device and select **Rename** from the shortcut menu.
3. Click **Device Pinouts** and explore the pinout information.



   ☐ Close the dialog box when you are finished.
4. Select the **Settings** tab to see the information about the last time the device was calibrated both internally and externally. Not all devices contain calibration information.
5. If you are using a physical device, right-click the NI-DAQmx device in the configuration tree and select **Self-Calibrate** to update the built-in calibration constants and calibrate the DAQ device using a precision voltage reference source. When the device has been calibrated, information in the **Self-Calibration** section updates. Skip this step if you are using a simulated device.
6. Close NI MAX when finished.

End of Exercise 2-1

Exercise 2-2: Validating I/O (DAQ)

Goal

Verify that the I/O of your DAQ device behaves correctly.

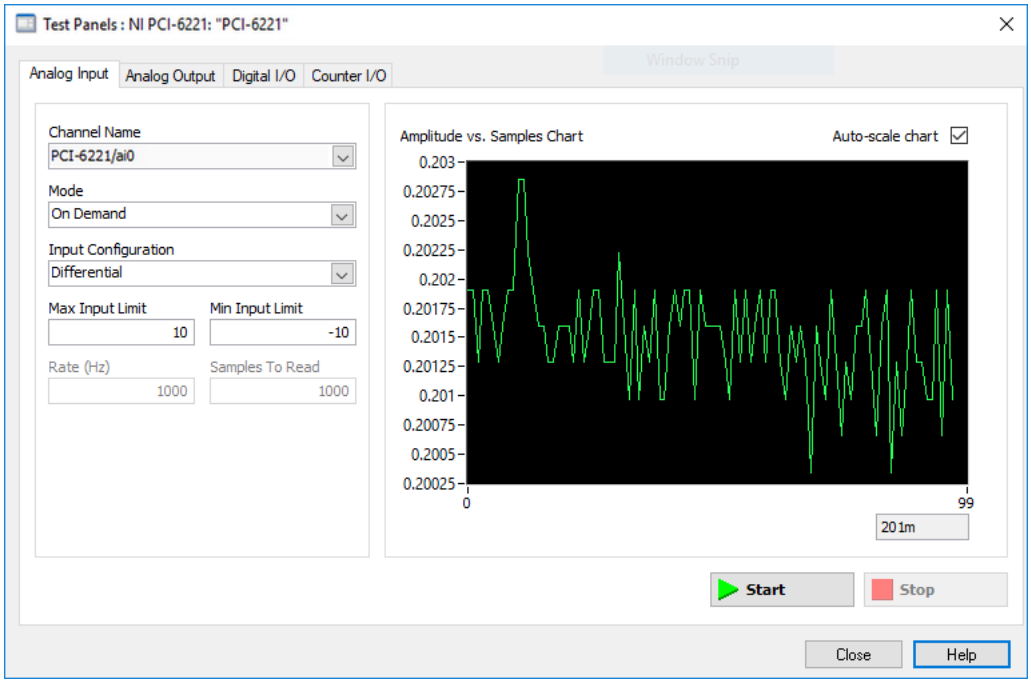
Scenario

You want to validate that your DAQ system acquires/generates the following data correctly:

- Acquired analog input voltage measurement from the temperature sensor.
- Generated analog output voltage.
- Generated digital output voltage.

Implementation

1. Launch **NI MAX** from the Windows **Start** menu.
2. Expand **Devices and Interfaces** and select the **PCI-6221**.
3. Click **Test Panels…**
4. Validate the analog input of your device.
   - ☐ Make sure that the **Analog Input** tab is selected in the Test Panels dialog box.
   - ☐ Select **PCI-6221/ai0** from the **Channel Name** control, which corresponds to the ai0 pin of the device. The temperature sensor of BNC 2120 is connected to ai0 of PCI-6221.
   - ☐ Change **Mode** to **Continuous** enables the measurement-specific settings, e.g. **Rate (Hz)**, and **Samples to Read.**
   - ☐ Click the **Start** button to start the acquisition.
   - ☐ Explore data on the chart, then click **Stop.**
   - ☐ Change **Samples To Read** to other values, e.g. 5000, 10000, etc.
   - ☐ Start the acquisition, and explore how it affects the acquired signal.



☐ Click the **Stop** button.

☐ You can disable the **Auto-scale chart** option, and change the scaling of the x-scale to adjust the graph by double-clicking the maximum and minimum values of the scale and typing the appropriate ones.

5. Validate the analog output of your device.

   ☐ Switch to the **Analog Output** tab.

   ☐ Set the **Channel Name** control, e.g. **PCI-6221/ao0**.

   ☐ You can either change **Mode** to **Voltage Sinewave Generation** or leave **Voltage DC.**

   ☐ Set **Output Value (V)** to the constant voltage you want to output.

   ☐ Click the **Update** button to start the generation.

   ☐ Click **Stop.**

6. Validate the digital I/O of your device.

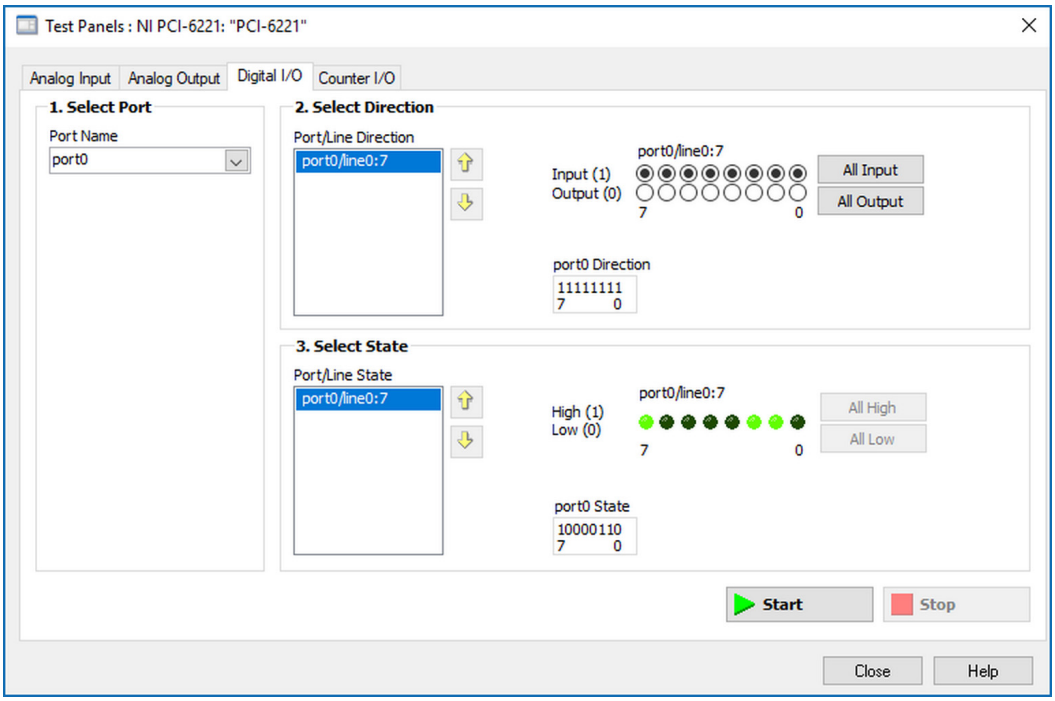
   ☐ Switch to the **Digital I/O** tab.

   ☐ Select the port from the **Port Name** control. Port is a collection of lines, and each line can output a high or low voltage.

   ☐ Select **All Output** in the **Select Direction** section.

   ☐ Configure the output voltage in the **Select State** section by changing the line state.

- ☐ Click **Start.**
- ☐ Select **All Low** and click **Stop.**
- ☐ Select **All Input** and click **Start.**



- ☐ Review the acquired data and click **Stop.**
- ☐ Close the **Test Panels** dialog box.

7. Close NI MAX.

End of Exercise 2-2

3 First Measurement (Non-NI Instrument)

In this lesson, you use LabVIEW to connect to non-NI instruments and validate the results.

Exercises

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

Exercise 3-1: Connecting Hardware (Non-NI Instrument)

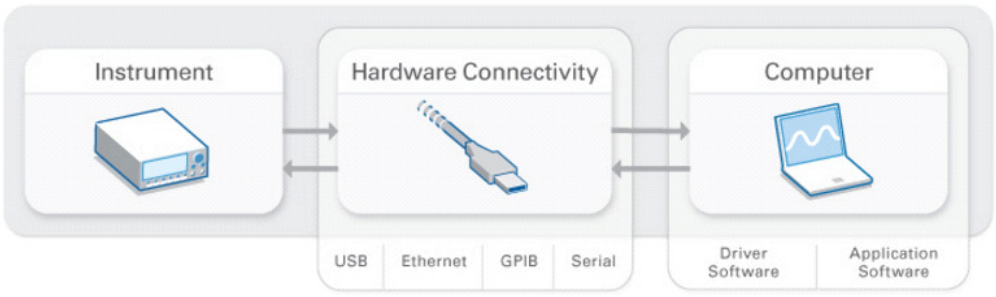
Goal

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

Scenario

You are creating an application that acquires data from a non-NI instrument (for example, a non-NI oscilloscope, or 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

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

Furthermore, imagine that your computer has a GPIB instrument control device, such as the GPIB-USB-HS, which allows your computer to communicate with a GPIB instrument.

Complete the following 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 to your computer GPIB interface with a GPIB cable. For example, connect the USB end of the GPIB-USB-HS to your computer and the GPIB end of the GPIB-USB-HS to the GPIB port of the NI Instrument Simulator.
3. Now you can programmatically control (send commands to, receive data from) the NI Instrument Simulator (that is a non-NI instrument).

Set Up the NI Instrument Simulator

This exercise uses the NI Instrument Simulator to simulate an instrument. Before NI Measurement & Automation Explorer (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 the figure below.

   ☐ 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**.

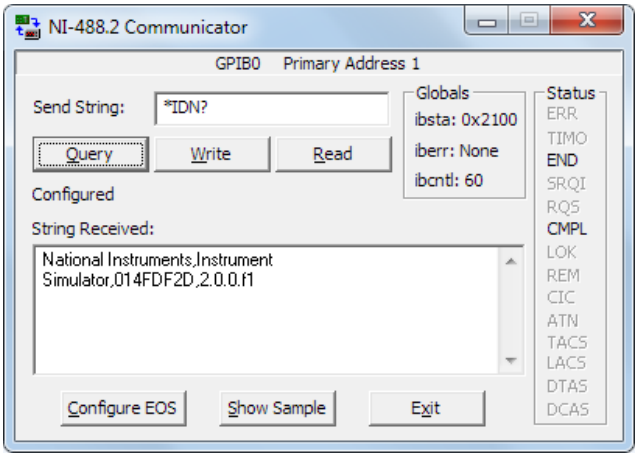
   ☐ Set **GPIB Primary Address** to `1`.

- ☐ 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 **NI MAX** from the Windows **Start** menu.
   - ☐ In NI MAX, expand the **Devices and Interfaces** 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.
2. Communicate with the GPIB instrument.
   - ☐ Make sure the GPIB interface is still selected in the **Devices and Interfaces**.
   - ☐ Click the **Scan for Instruments** button on the toolbar.
   - ☐ Expand the GPIB interface selected in the **Devices and Interfaces**. An instrument named `Instrument Simulator` appears.
   - ☐ Click `Instrument Simulator` to display information about it in the right pane of NI MAX.
   - ☐ Click the **Settings** tab at 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 manufacturer and model number in the **String Received** indicator as shown in the figure. You can use this communicator

window to debug instrument problems or to verify that specific commands work as described in the instrument documentation.



- ☐ 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.

3. 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 NI 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**.
4. Exit NI MAX.
5. Click **Yes** if prompted to save the instrument.

End of Exercise 3-1

4 Exploring an Existing Application

In this lesson, you explore an existing application and predict its behavior.

Exercises

Exercise 4-1: Exploring an Existing Project and a VI

Goal

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

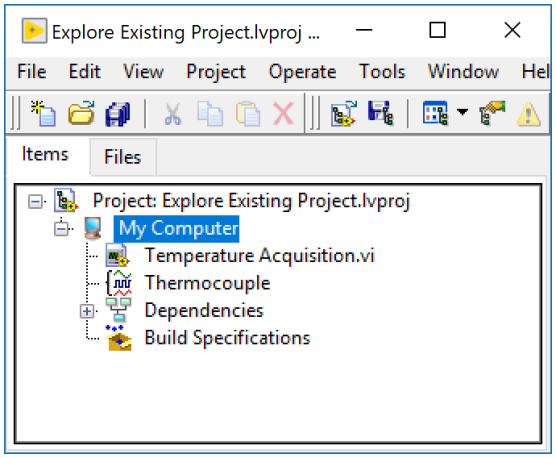
Explore an existing project

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

2. Notice that the **Project Explorer** window contains the following two items.

   ☐ Thermocouple task

   ☐ Temperature Acquisition VI

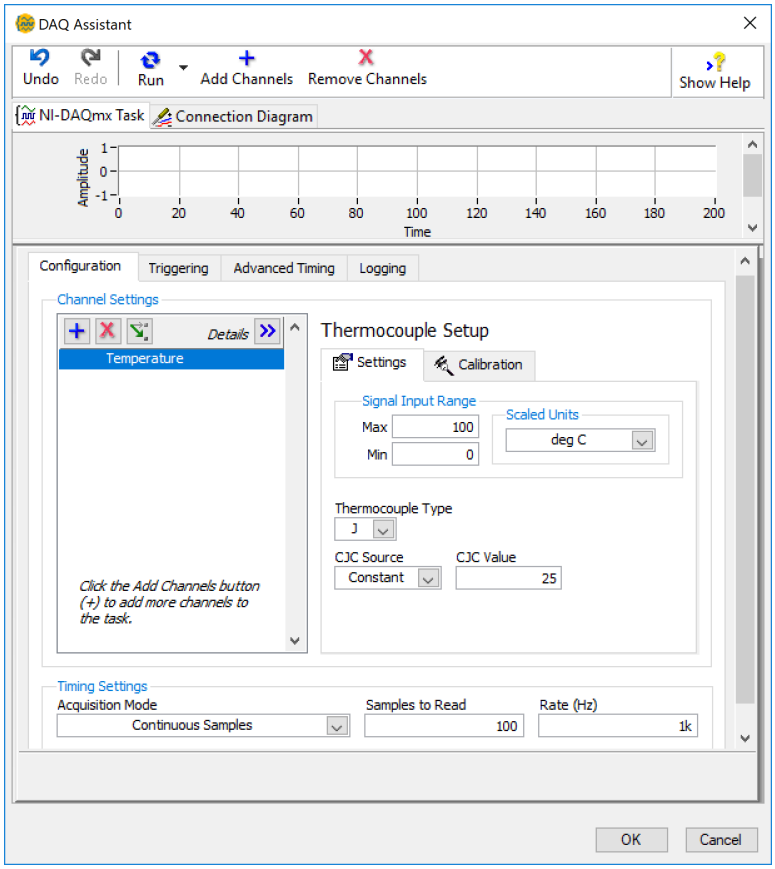


3. Explore the Thermocouple task.

   ☐ Double-click the Thermocouple task to launch DAQ Assistant.

   ☐ In the opened window you can configure the DAQmx task by changing different values of sample rate, samples to read, signal input range, and by adding physical and virtual channels to the task.

   ☐ Configure the Thermocouple task as shown in the figure below.



Explore and run the VI in the project

1. Open the VI in the project.

   ☐ In the **Project Explorer** window, double-click the Temperature Acquisition VI.

2. Run the VI.

   ☐ Notice that **Current Temperature** displays the temperature.

   ☐ Run the VI a couple more times.

   ☐ Set **Max Temperature Threshold** to values above and below the current temperature, and observe how this affects the **Threshold Exceeded?** and **Message** indicators.

Explore the VI Front Panel

The front panel is the user interface of your VI.

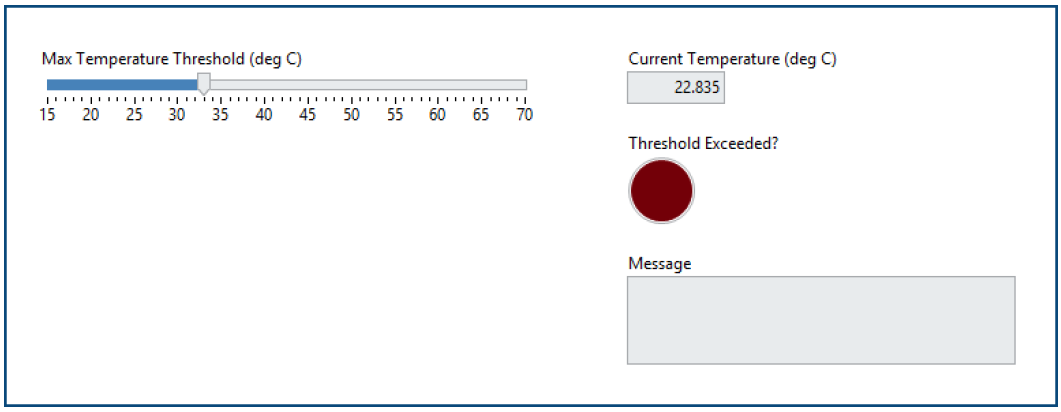
1. Examine the VI front panel.

2. Determine whether each of the following front panel objects is a control (front panel input) or indicator (front panel output), and circle the correct answer.

- ☐ **Max Temperature Threshold** Control | Indicator
- ☐ **Current Temperature** Control | Indicator
- ☐ **Threshold Exceeded?** Control | Indicator
- ☐ **Message** Control | Indicator

3. Determine whether each of the following front panel objects is a numeric, boolean, or string, and circle the correct answer.

- ☐ **Max Temperature Threshold** Numeric | Boolean | String
- ☐ **Current Temperature** Numeric | Boolean | String
- ☐ **Threshold Exceeded?** Numeric | Boolean | String
- ☐ **Message** Numeric | Boolean | String



Explore the Block Diagram

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

1. Select **Window»Show Block Diagram** in the front panel toolbar to display the block diagram.

   - ☐ On the block diagram, identify the following items. How many can you find of each item?
     - Controls
     - Indicators
     - Constants
     - Free labels
     - Functions
     - VIs

2. Use the context help to learn more about the items on the block diagram.

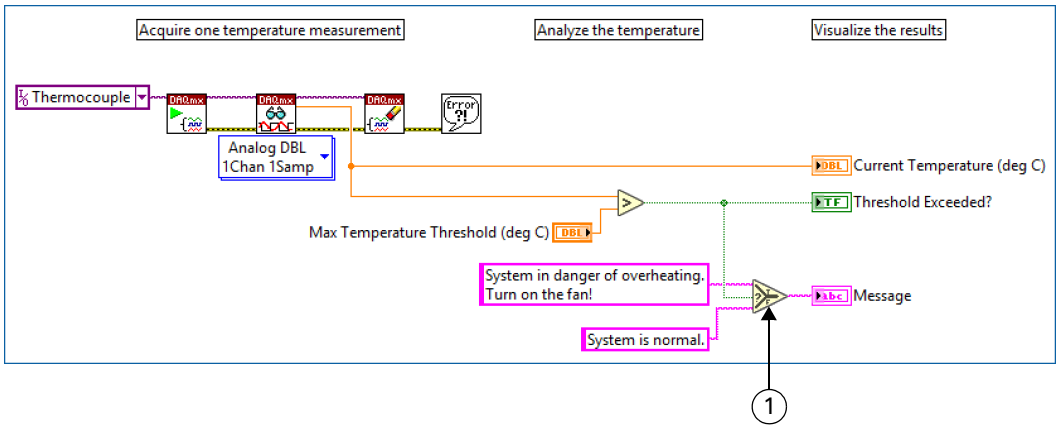
   ☐ Press <Ctrl-H> to open the **Context Help** window or select **Help»Show Context Help**.

   ☐ 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.

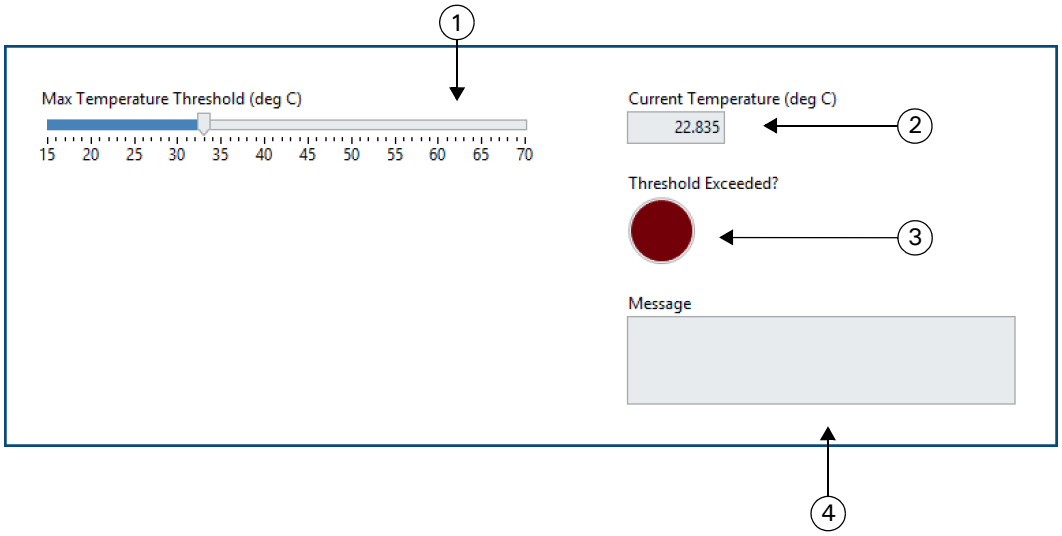
3. Get detailed help for the Select function.



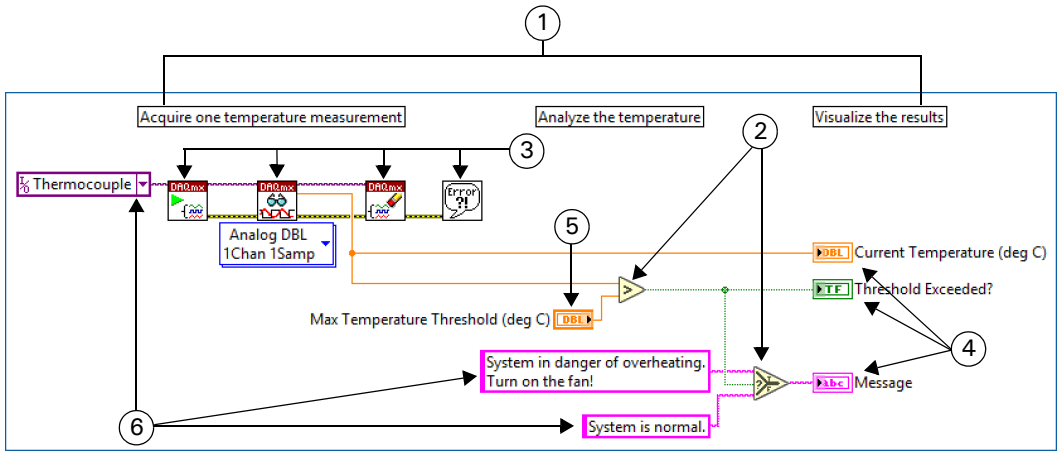
1	**Select** function—Place your cursor over the function. In the **Context Help** window, click the **Detailed help** link to launch the *LabVIEW Help* and learn more about this function.

Check Your Answers

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



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



1	Comments	4	Indicators
2	Functions	5	Control
3	VIs	6	Constants

On the Job

Describe the controls and indicators that you need when you create VIs at work. (name, data type)

End of Exercise 4-1

Exercise 4-2: Exploring Hardware Example Programs

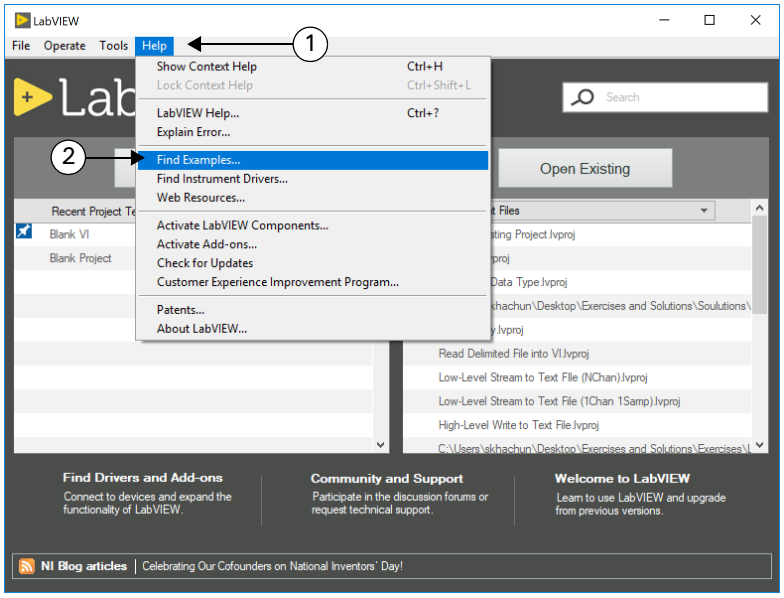
Goal

Explore hardware example programs included in LabVIEW.

Instructions

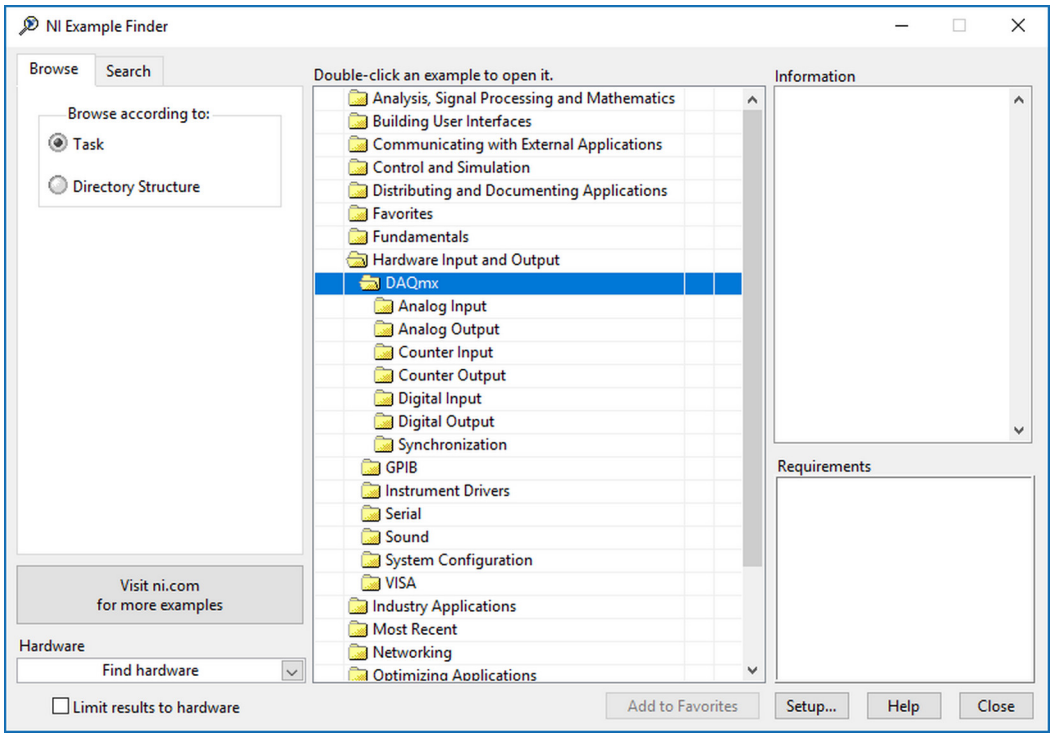
NI-DAQmx Example Programs

1. Explore NI-DAQmx examples.



1 From the Getting Started window, open the **Help** menu.
2 Select **Find Examples**.

2. In the **NI Example Finder** window, double-click `Hardware Input and Output»DAQmx`, double-click the folder for the type of example you want to explore, and then double-click one of the examples to launch an example DAQmx project.



3. Explore the VIs found in the launched example.

   ☐ Open and explore the VIs from the DAQmx subfolders.

   ☐ Explore the front panel.

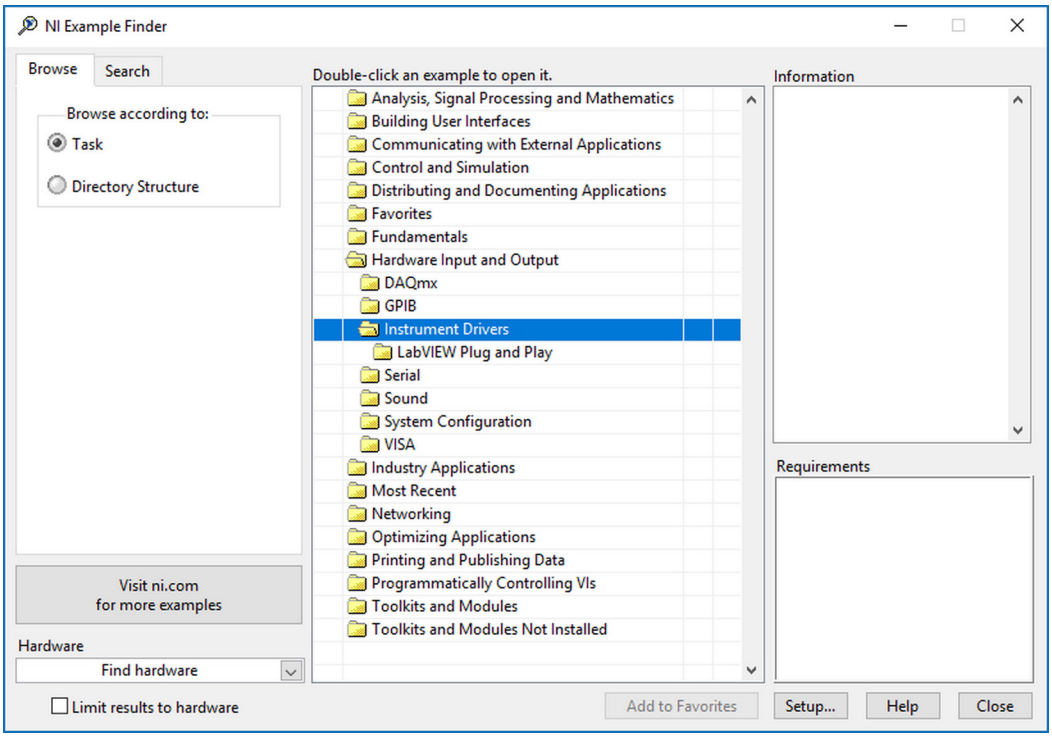
   ☐ Explore the block diagram. You can read a description of a VI by using the **Context Help** window.

4. Close the VI when finished. If you save an example, it will save over the original example. In order to prevent that you should choose “Save as Copy” or don’t save the VI.

Non-NI Instrument Example Programs

1. Explore Non-NI Instrument Examples.

   ☐ Switch to NI Example Finder and navigate to `Hardware Input and Output»Instrument Drivers»LabVIEW Plug and Play` folder.



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

   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 found in the launched example.

   ☐ Open and explore the VIs from the Instrument Drivers subfolder.

   ☐ Explore the front panel.

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

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

End of Exercise 4-2

5 Creating Your First Application

In this lesson, you create your first application.

Exercises

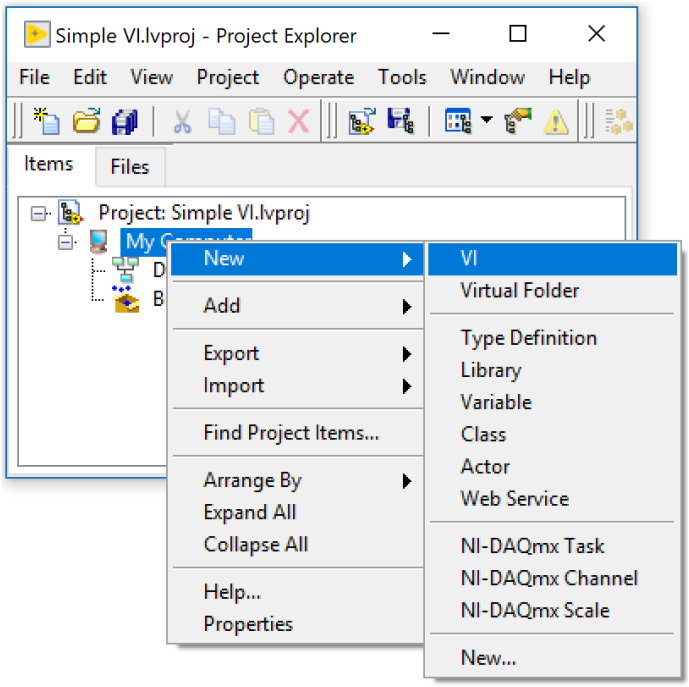
Exercise 5-1: Creating a Simple Project and a VI

Goal

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

Instructions

1. Create a new project.
   - ☐ From the **Getting Started** window, select **Create Project.**
   - ☐ As a starting point, choose **Blank Project**, and click **Finish.**
2. Save the project.
   - ☐ Select **File»Save All.**
   - ☐ Type `Simple VI` in the **File Name** field.
   - ☐ In the **File Explorer** window, browse to the `<Exercises>\LabVIEW Core 1\Simple VI` directory.
   - ☐ Click **OK**.
3. Create a new VI in your project.
   - ☐ In the **Project Explorer** window, select **File»New VI**.



   - ☐ Save the VI as `Straight-Line Equation`.

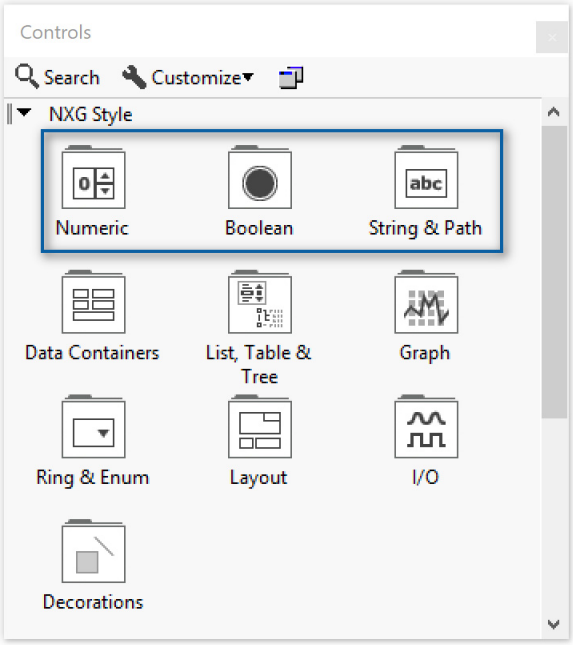
4. Explore the Controls Palette on the front panel.

   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, or indicator you need.

   ☐ From the front panel toolbar, select **View»Controls Palette,** or right-click in the front panel.

   Note You can access the NXG Style controls palette by right-clicking in the front panel and hovering your mouse over NXG Style submenu. Also, if you want for NXG Style controls palette to open automatically when you click in the front panel press the pin button in the upper-left corner and drag the NXG Style controls palette to the top of the stack.

   ☐ Open the **Numeric**, **Boolean**, and **String&Path** palette categories to explore various types of controls and indicators.



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

   ☐ Press the **Search** button on the top right corner of the Controls palette.

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

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

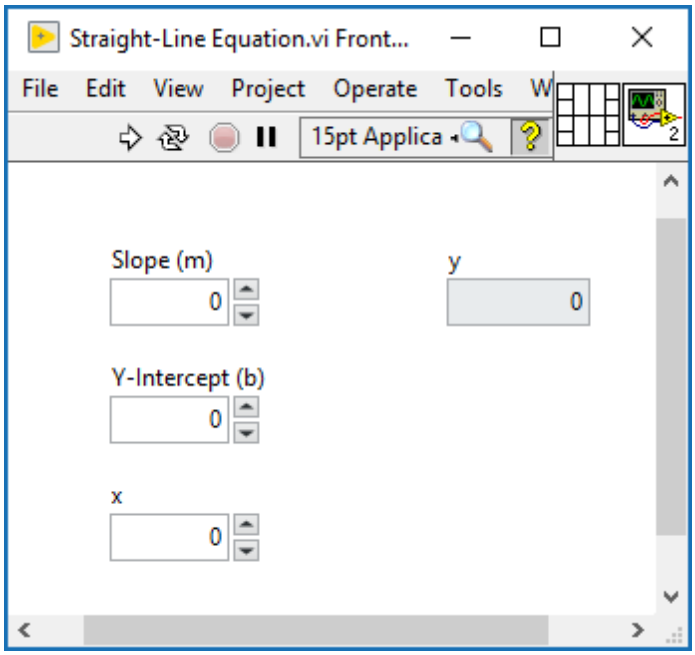
   ☐ Click **Numeric Control (NXG Style)** in the search results.

   ☐ Drag it to the front panel. Do this three times to place three numeric controls on the front panel. Rename them as `Slope (m)`, `Y-Intercept (b)`, and `x`.

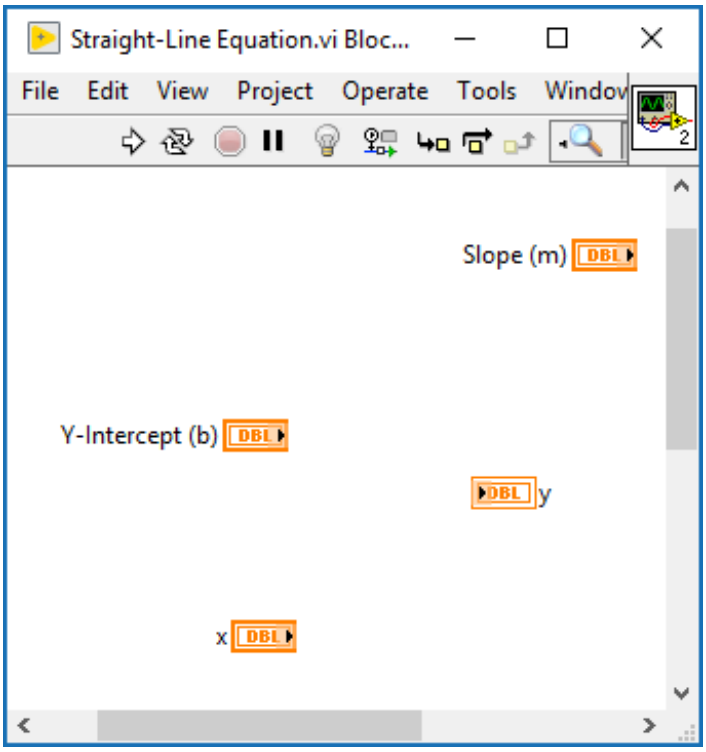
   ☐ Press <Ctrl-Space> to open the **Quick Drop** dialog box.

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

- ☐ Double-click **Numeric Indicator (NXG Style)** in the search results and place it on the front panel. Rename it `y`.
- ☐ Arrange the items on the front panel as shown in the following figure.



6. Develop the code for your VI on the block diagram.
   - ☐ Switch to block diagram by pressing <Ctrl-E>.
   - ☐ The block diagram contains the terminals of the controls and indicators, placed on the front panel. In this case it will contain the **Slope (m)**, **Y-Intercept (b)**, **x**, and **y** terminals as shown in the figure below.



☐ Press <Ctrl-Space> and type `multiply` in the search text box. Double-click **Multiply** in the search results and place the Multiply function on the diagram.

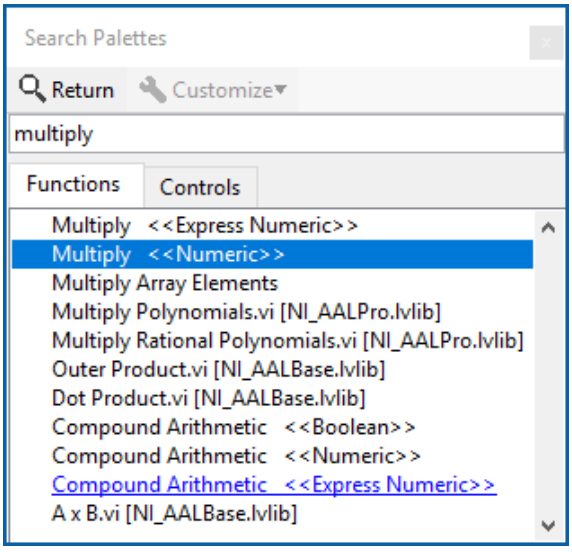
☐ Place an Add function to the block diagram.

☐ Wire the block diagram as shown in the following figure.

7. Locate the palette that contains a particular VI or function using the palette search.

☐ Right-click in the block diagram, press the **Search** button in the upper right hand corner of the **Functions** palette and search for the term `Multiply`.

☐ In the search results, double-click the desired function to display the palette category that contains the function. This allows you to browse related functions.

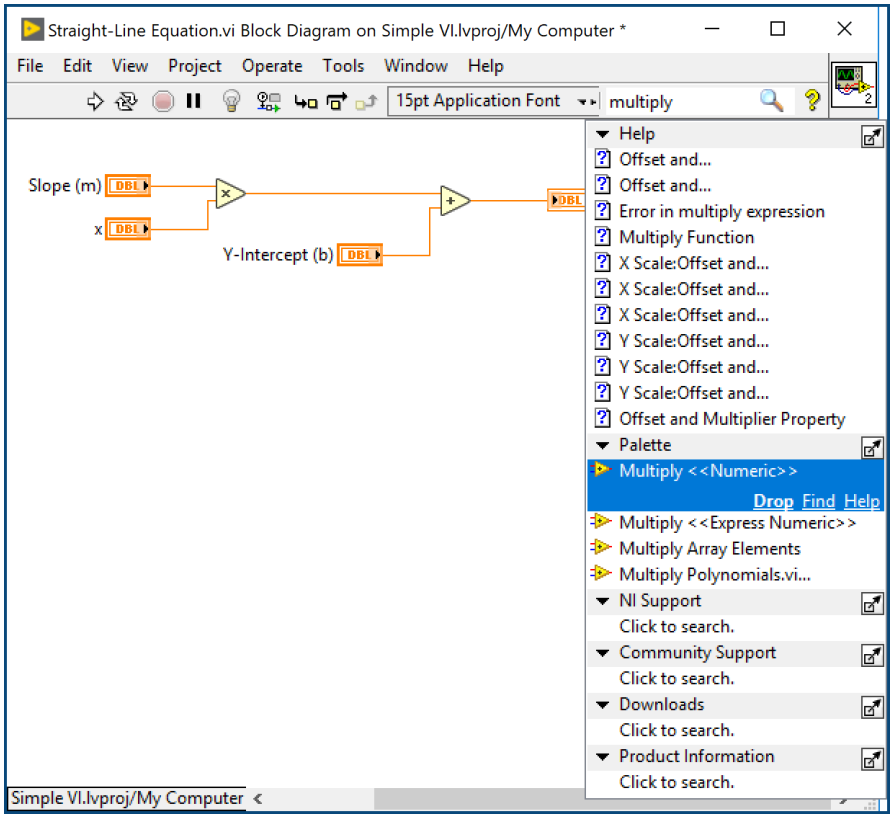


8. Practice using the global search feature.

   ☐ Type `Multiply` in the **Search** bar in the upper right hand corner of either front panel or block diagram.

   Note As you type, the global search automatically looks for matches in 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 **Run**.

   ☐ 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 (b)	5		
Slope (m)	-3	y	-16
x	2		
Y-Intercept (b)	-10		

Your Turn

1. Create a new project and name it `Average.lvproj`.
2. Add a new VI named **Average.vi** 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: Creating 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.

   ☐ From the **Getting Started** window, select **Create Project»All»Blank Project** and click **Finish**.

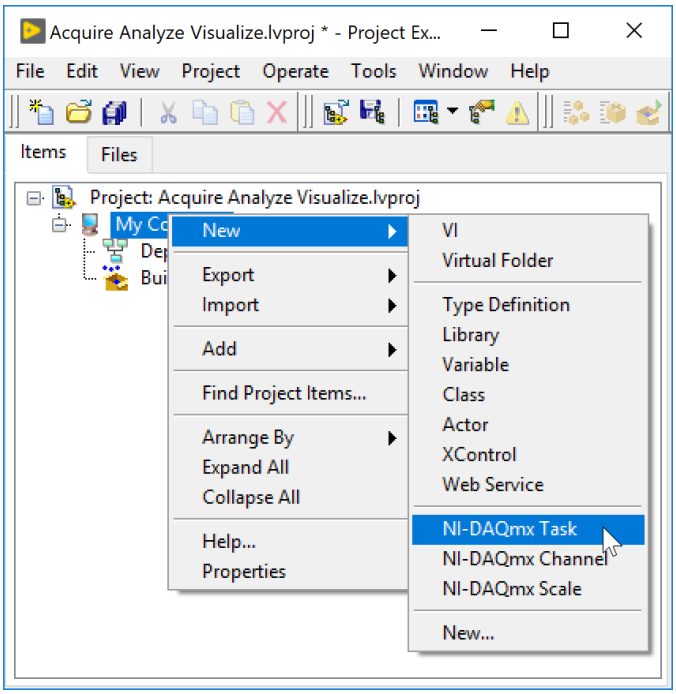
   ☐ Select **File»Save All** and enter `Acquire Analyze Visualize.lvproj` in the **File name** field.

   ☐ Browse to the `<Exercises>\LabVIEW Core 1\Acquire Analyze Visualize` directory and click **OK**.

Create an NI-DAQmx Task

1. Create a DAQmx task.

   ☐ In the **Project Explorer** window right-click **My Computer** and select **New» NI-DAQmx Task.**



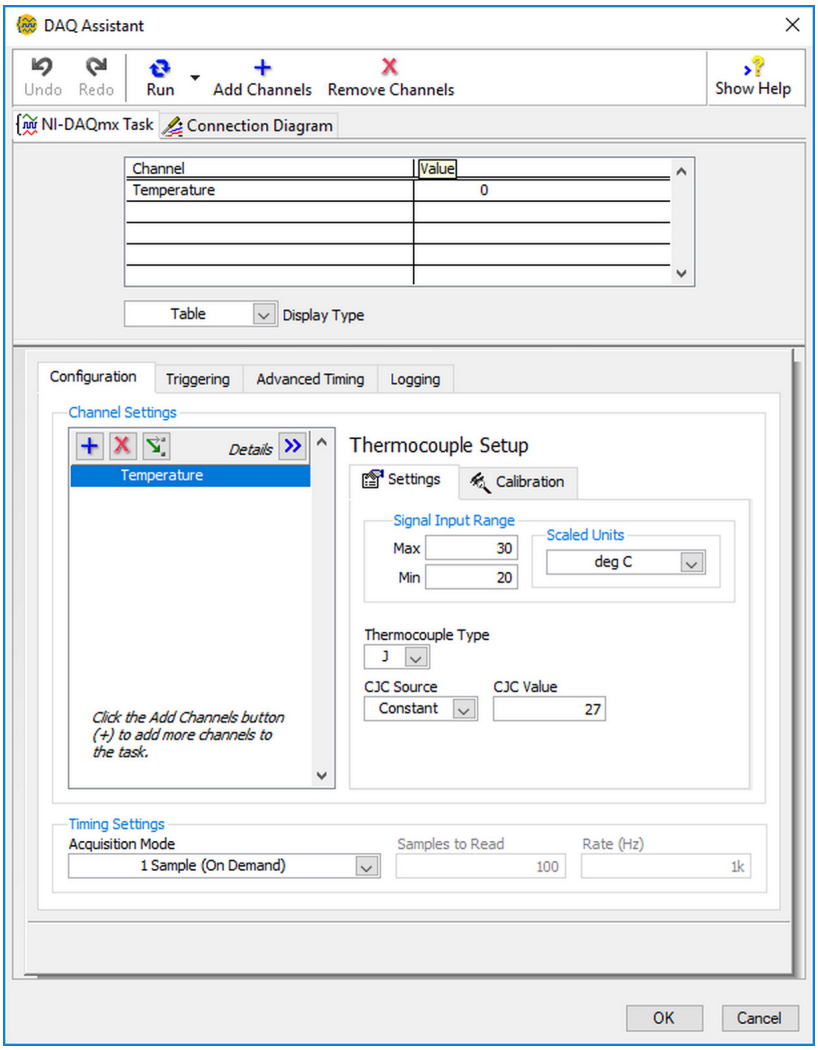
2. Configure the measurement.

   ☐ In the Create New dialog box that appears, select **Acquire Signals»Analog Input»Temperature»Thermocouple**.

☐ The next dialog box that appears prompts you to select the physical channel to use with your new task. Under **PCI-6221**, select **ai1** and click **Next**.

☐ Name the task `Thermocouple`, and click **Finish**.

☐ In the **DAQ Assistant** dialog box, configure the task as shown in the following figure.



Note You should configure the task according to your specific hardware configuration, e.g. if you have a thermocouple connected of a different type and with different signal input range connected, then you should configure the parameters accordingly to that thermocouple characteristics.

3. Validate the signal is correct using the **DAQ Assistant** dialog box.

   ☐ On the top pane of the dialog box, switch the **Display Type** to **Chart** and run the task to verify that the chart displays the expected temperature.

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

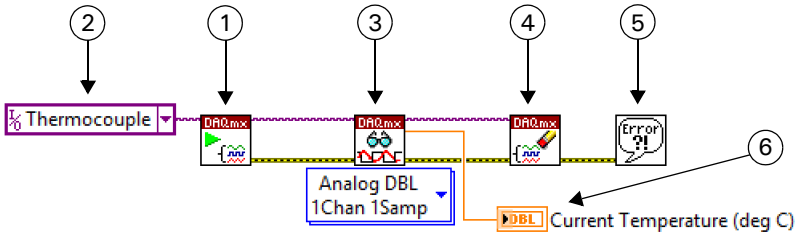
Acquire Data

1. Create a new VI in your project.

   ☐ In the **Project Explorer** window, add a VI.

   ☐ Click <Ctrl-S> to save the VI and name it `Acquire Analyze Visualize`.

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



1 **DAQmx Start Task** VI—Transitions the task to the running state to begin the measurement or generation.
2 **Task** constant—Right-click the **task in** input of the DAQmx Start Task VI and select **Create Constant**. Set the constant to **Thermocouple**.
3 **DAQmx Read** VI—In the Polymorphic VI selector, set the values shown in the table below.
4 **DAQmx Clear Task** VI—Clears the task. Before clearing, this VI aborts the task, if necessary, and releases any resources the task reserved.
5 **Simple Error Handler** VI—Indicates whether an error occurred. If an error occurred, this VI returns a description of the error and optionally displays a dialog box.
6 **Current Temperature (deg C)** indicator—Right-click the **data** output of the DAQmx Read VI and select **Create Indicator**. Rename the indicator `Current Temperature (deg C)`.

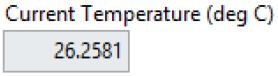
Note Task constant populates with any DAQmx tasks that you have created in this project.

DAQmx Read Configuration Settings

Channel Type	Analog Input
Channel Count	Single Channel
Sample Count	Single Sample
Data Format	DBL (Floating Point)

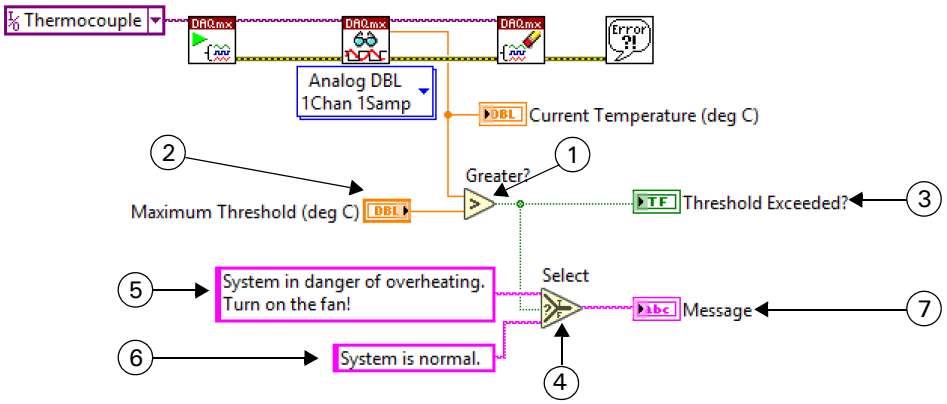
3. Switch to Front Panel.

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



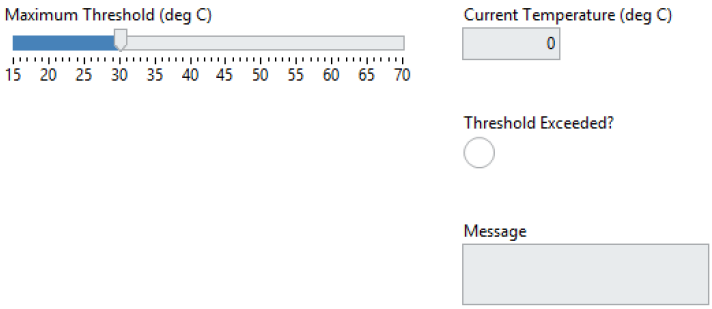
Analyze and Visualize Data

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



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

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

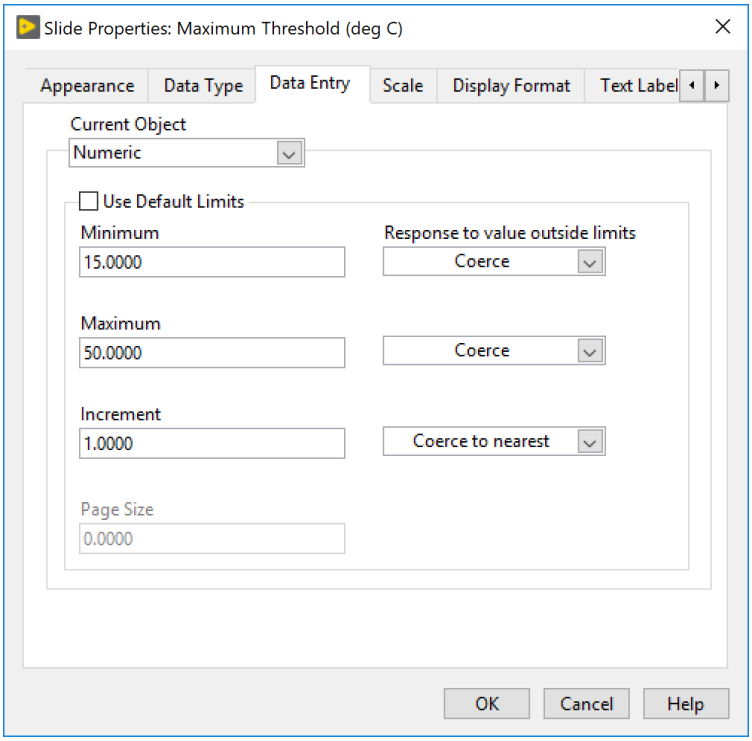


☐ Right-click the **Maximum Threshold (deg C)** control and select **Replace» Numeric»Horizontal Pointer Slide (NXG Style)**. Resize the control as desired.

☐ Double-click the minimum and maximum values on the **Horizontal Pointer Slide** control and set them according to the table below.

Minimum	15
Maximum	20

- ☐ Right-click the **Horizontal Pointer Slide** control and select **Data Entry** from the shortcut menu. Set the properties on the **Data Entry** tab as shown in the following figure.



- ☐ Set the current value of the **Horizontal Pointer Slide** control to `30`. After that, right-click the control and select **Data Operations»Make Current Value Default** from the shortcut menu.

- ☐ Right-click the **Threshold Exceeded?** indicator and select **Replace»Boolean» LED (NXG Style)**. 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 (deg C)**, and compare the indicator results.

Your Turn

1. Add a new VI named `Celsius and Fahrenheit.vi` to the project.

2. Modify the VI to acquire a temperature, display the temperature in degrees Celsius, convert the temperature to display the temperature in degrees Fahrenheit. The VI should have the following indicators.

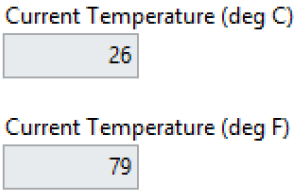
   - ☐ Current Temperature (deg C)

   - ☐ 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: DAQmx Task vs Full DAQmx API

Goal

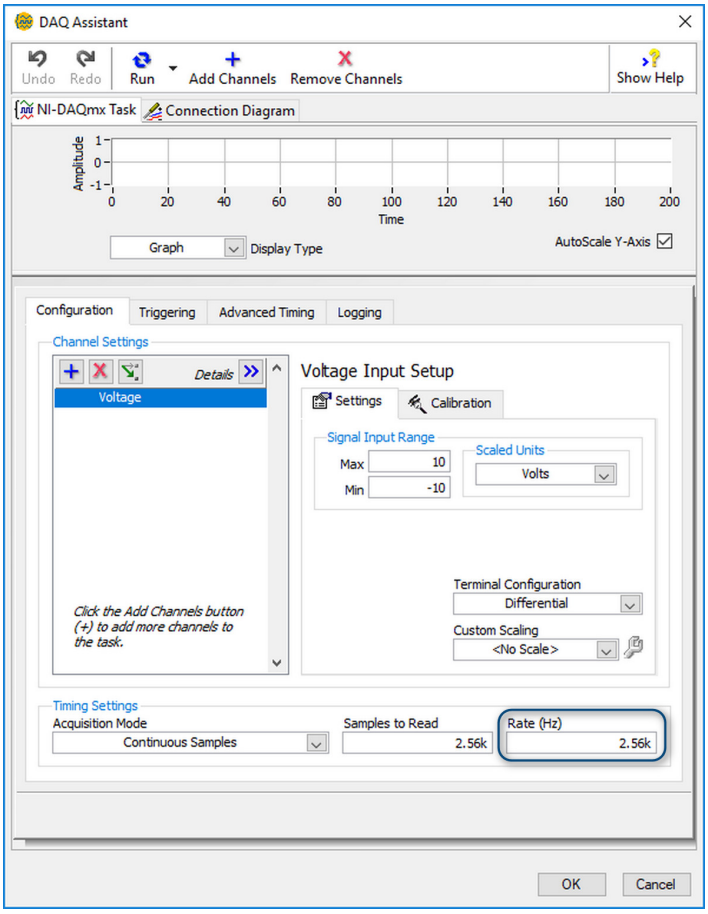
Use and compare the DAQmx task with full DAQmx API approaches of acquiring data from a DAQmx device.

Instructions

Approach 1: Using the DAQmx Task

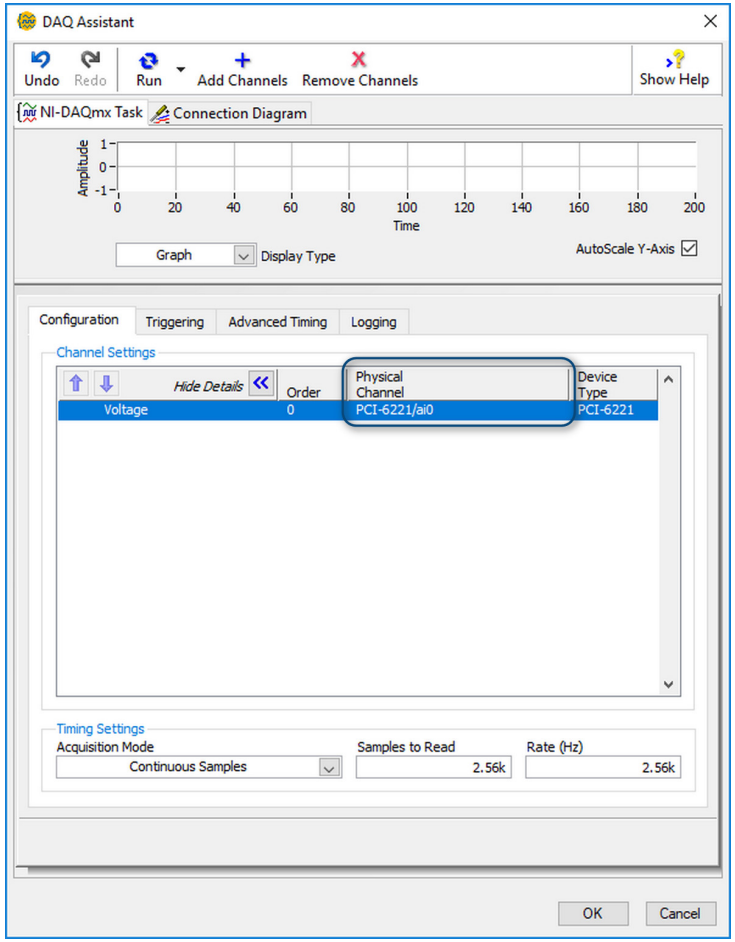
This approach helps you get started quickly if you have already created a DAQmx task.

1. Open `<Exercises>\LabVIEW Core 1\DAQmx Task vs Full DAQmx API\DAQmx Task vs Full DAQmx API.lvproj`.
2. Examine the task presented in the project.
   - ☐ Open the My Analog Input Voltage task from the **Project Explorer** window.
   - ☐ Examine the settings in the **DAQ Assistant** dialog box. Notice **Rate (Hz)** is set to 2.56 kHz.



Note Adjust **Rate (Hz)** to a rate that is supported by your DAQ device, if necessary.

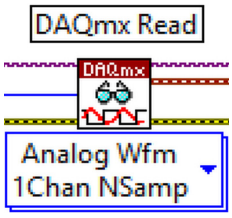
☐ Click the **Details** button, and note that **Physical Channel** is set to **PCI-6221/ai1**.



☐ Close the **DAQ Assistant** dialog box.

3. Open DAQmx Task Method VI from the **Project Explorer** window.
4. Tile the front panel and block diagram to see both at the same time by pressing <Ctrl-T>.
5. Examine the block diagram.

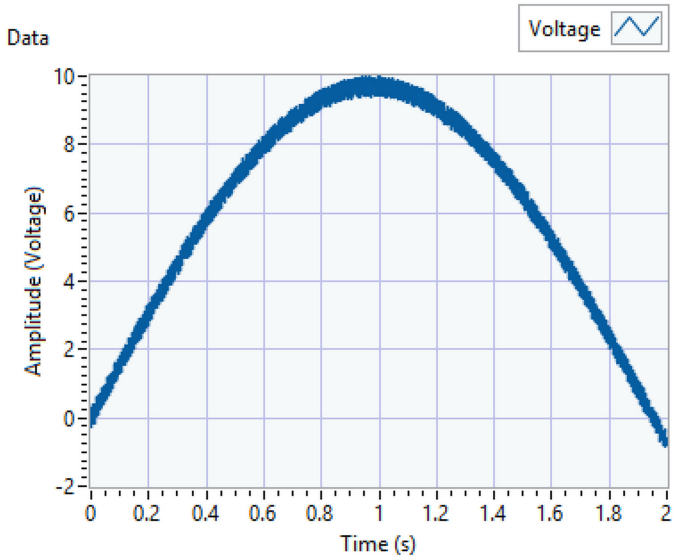
- ☐ Notice the task constant, which references the My Analog Input Voltage task in the project. This task defines the channel and timing settings.
- ☐ Pay attention to the configuration of the DAQmx Read VI shown under the Polymorphic VI selector.



6. Examine the behavior of this VI.
   - ☐ On the front panel, set the value of the **Number of Samples** control to `5120`.

Note Because the sample rate defined in the **My Analog Input Voltage** task is 2.56 kHz (samples per second) and **Number of Samples** is set to 5,120, the DAQmx Read VI will read 5,120 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 5,120 samples. Notice that the **Time (s)** x-axis shows that graph contains 2 seconds of data.



Approach 2: Using the Full DAQmx API

This approach allows all the configuration settings 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 DAQmx Task Method VI.

   ☐ In the **Project Explorer** window, right-click the DAQmx Task Method VI and select **Save As**.

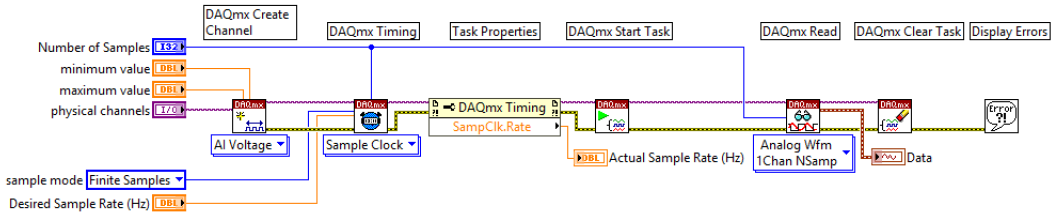
   ☐ In the appeared dialog box, select **Open additional copy**, check the **Add copy to DAQmx Task vs Full DAQmx API.lvproj** box, and press **Continue**.

   ☐ Name the VI as `Full DAQmx API Method.vi`, then click **OK**.

   ☐ Close the DAQmx Task Method VI.

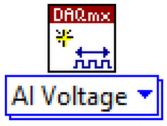
2. Delete the **task in** constant from the block diagram.

3. Programmatically create and configure the DAQmx task, as shown in the figure below by following the instructions in the next steps.



4. Add a DAQmx Create Channel VI to the block diagram.

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



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

   ☐ Right-click both the **minimum** and **maximum value** inputs, and select **Create Control**.

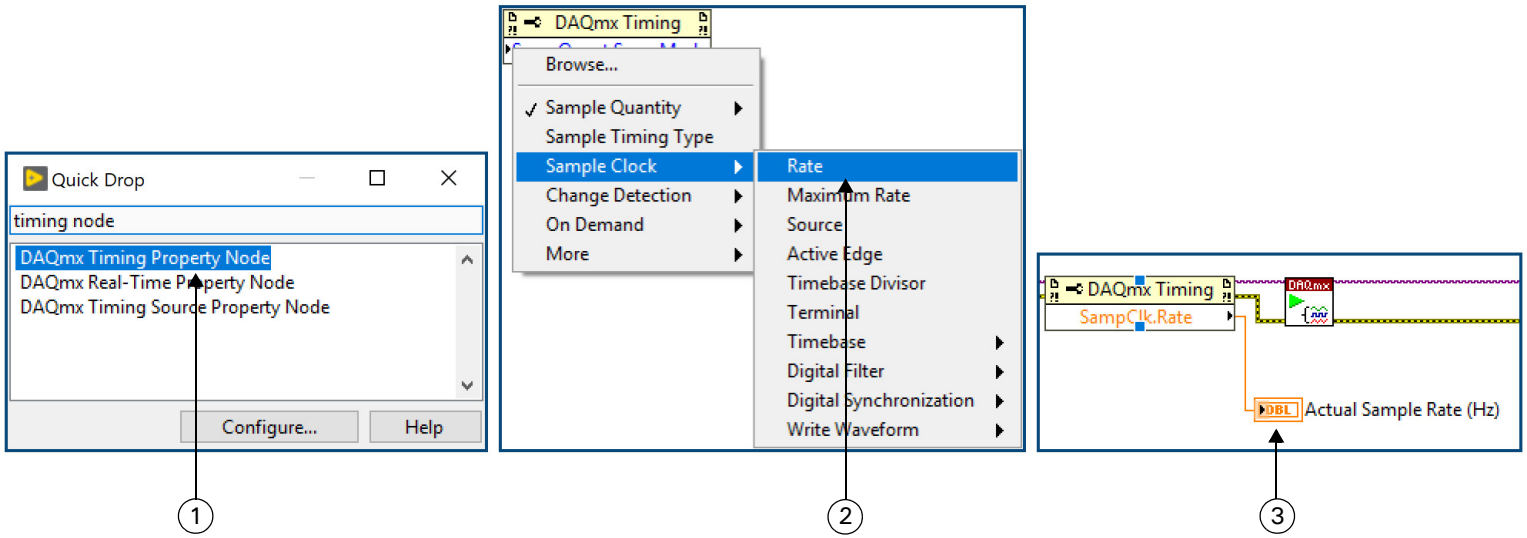
5. Add a DAQmx Timing VI to set the sample rate and sample mode.

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

- ☐ Right-click the **rate** input and select **Create Control**. Rename it `Desired Sample Rate (Hz)`.
- ☐ Right-click the **sample mode** input and select **Create Constant**. Set the value of 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 VI.

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

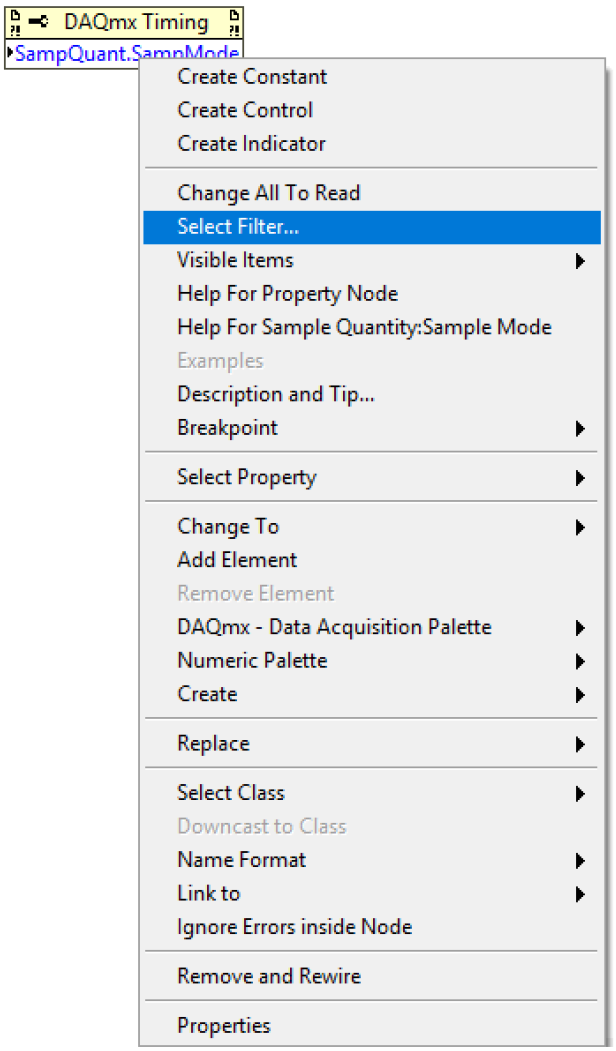
6. Add a DAQmx Timing Property Node to get the actual sample rate.



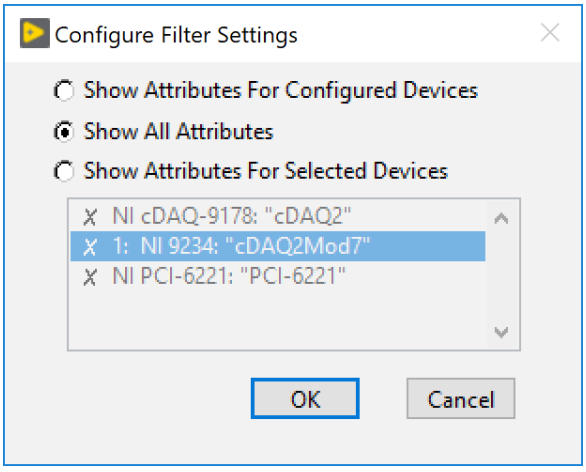
1 **Timing**—Select and place the DAQmx Timing Property Node from the **Quick Drop** menu.
2 **Sample Clock Rate**—In the pull-down menu, select **Sample Clock»Rate**.
3 **Actual Sample Rate (Hz)**—Right-click the DAQmx Timing Property Node and select **Change All To Read**. Then right-click the **SampleClock:Rate** output and select **Create Indicator**. Rename the indicator as `Actual Sample Rate (Hz)`.

Note If you can't locate the appropriate Property Node setting follow the instructions below.

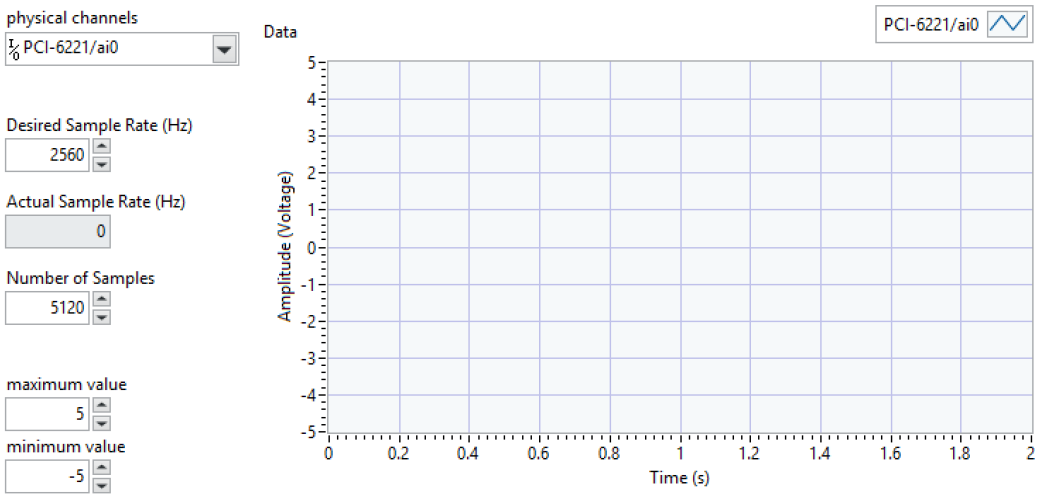
- Right-click on the property node and click **Select Filter…**.



- The **Configure Filter Settings** dialog box should appear similar to the following picture. Selecting **Show All Attributes** will cause LabVIEW to display all properties regardless of which devices are configured on that computer.



7. Complete the wiring as shown in the figure under step 3.
8. Switch to the front panel and arrange the controls as shown in the following figure.



9. Set the **Number of Samples** control to `5,120`.
10. Set the values of the added controls to match the values in the **DAQ Assistant** window.
11. You can find the values for **DAQ Assistant** in the locations shown in the following .

Control Name	DAQ Assistant Value Location
physical channels	**Configuration»Details»Physical Channel**
Desired Sample Rate (Hz)	**Configuration»Timing Settings»Rate (Hz)** **Note** Enter this value into the control in units of Hz, not kHz.
maximum value	**Configuration»Signal Input Range»Max**
minimum value	**Configuration»Signal Input Range»Min**

12. Run the VI.
13. Because you are using the DAQmx Create Channel and DAQmx Timing VIs, you can modify the physical channel and sample rate directly from your VI. This VI is no longer directly connected to the **My Analog Input Voltage** task. All the configuration for this DAQmx task is contained in this VI.
    - ☐ Change the **Desired Sample Rate (Hz)** control value from `2,560` to `5,120` to update the sample rate.
    - ☐ Run the VI, and notice that the **Time (s)** x-axis now shows that only 1 second of data was acquired.

- ☐ Change the **Desired Sample Rate (Hz)** control value to `3,333`. Run the VI, and notice that the **Actual Sample Rate** indicator value is different from **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 the 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 **physical channels** control to a different channel, such as PCI-6221/ai3.
- ☐ Run the VI, and notice that you are now acquiring data from a different channel.

14. Select **File»Save**.
15. Close the project when finished.

On the Job

If your application will use NI-DAQmx, which approach will you use? Circle the correct answer.

a. DAQmx Task Approach

OR

b. Full DAQmx API Approach

End of Exercise 5-3

6 Debugging and Troubleshooting

In this lesson, you learn about tools that help you debug and troubleshoot a VI.

Exercises

Exercise 6-1 Debugging

Exercise 6-1: Debugging

Goal

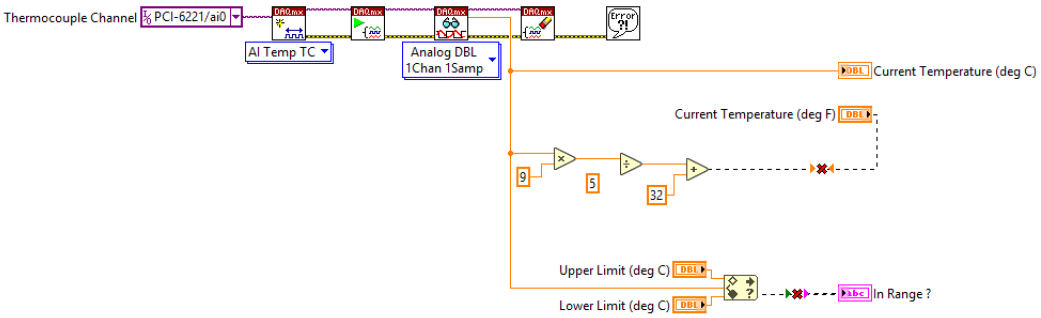
Use debugging tools and troubleshooting techniques to fix a broken VI that returns unexpected data.

Instructions

Edit-Time Errors

Locate and correct errors that prevent the VI from running.

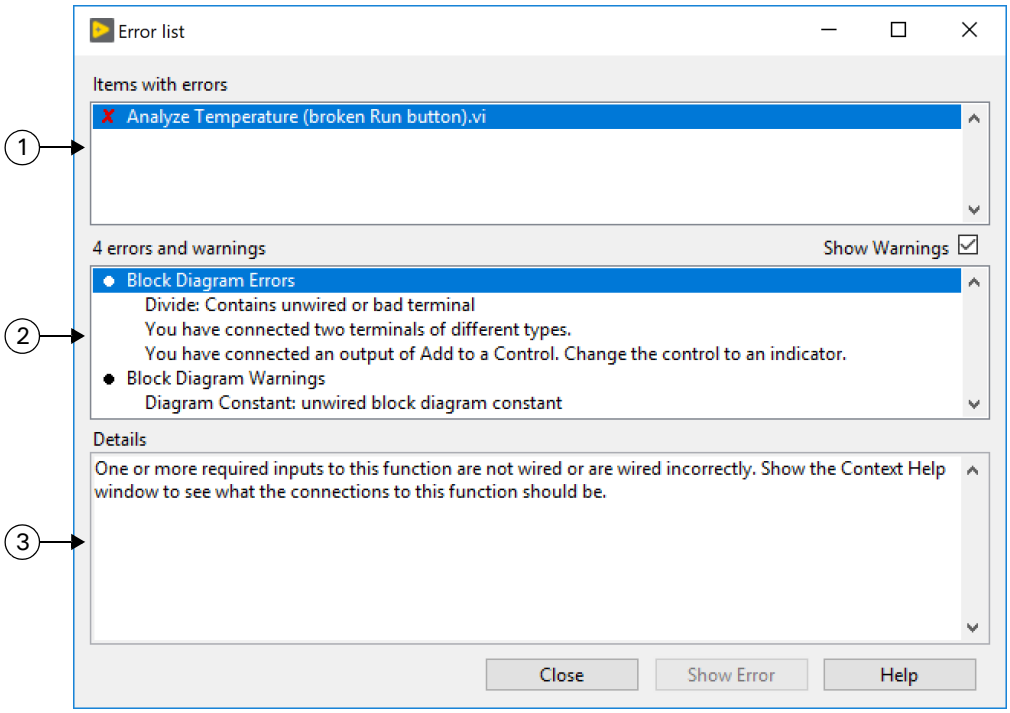
1. Open and examine the Analyze Temperature VI.
   - ☐ Open the `Debug.lvproj` in the `<Exercises>\LabVIEW Core 1\Debug` directory.
   - ☐ Open the Analyze Temperature (broken Run button) VI from the **Project Explorer** window.
   - ☐ Notice the **Run** button on the toolbar appears broken, indicating that the VI is broken and cannot run.
2. Examine the block 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 specified range.

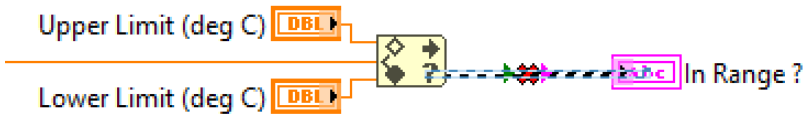
3. Find and fix each error.

   ☐ Click the broken **Run** button to display the **Error list** dialog box. Examine the errors and warnings listed.



1 **Items with errors**—Lists all LabVIEW items affected by errors and warnings in the current VI. If two or more items have the same name, this list shows the specific application instance for each ambiguous item.
2 **Errors and warnings**—Lists all errors and warnings associated with the VI, if a VI is selected.
3 **Details**—Indicates specific details of the selected error or warning.

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



   ☐ Use the information in the **Error list** dialog box to fix each error.

   ☐ Notice that after you fix all the errors, 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 Explorer** window, open Analyze Temperature (incorrect behavior) VI.

   ☐ On the block diagram, set T**hermocouple Channel** to **PCI-6221/ai1**.

   ☐ Switch to the front panel and run the VI.

   The DAQmx Read VI returns 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) x 9/5 + 32
   = _____ x 9/5 + 32
   = _____

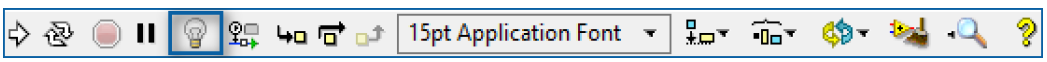
   Does the **Temperature (deg F)** indicator value match your calculation? _____

   ☐ Notice that even though the VI runs, the VI returns an incorrect result in the **Temperature (deg F)** indicator.

   Complete the following steps to identify the source of this error using the debugging tools and correct the error.

2. Animate the flow of data through the block diagram.

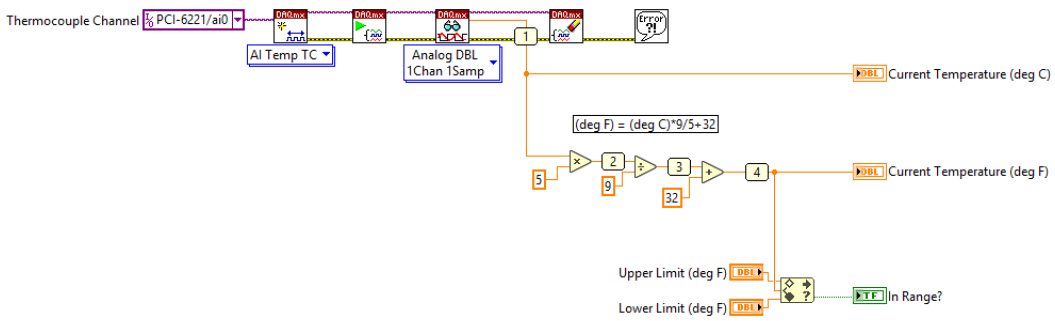
   ☐ Switch to the block diagram and click the **Highlight Execution** button to enable execution highlighting and then run the VI.



   ☐ 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.

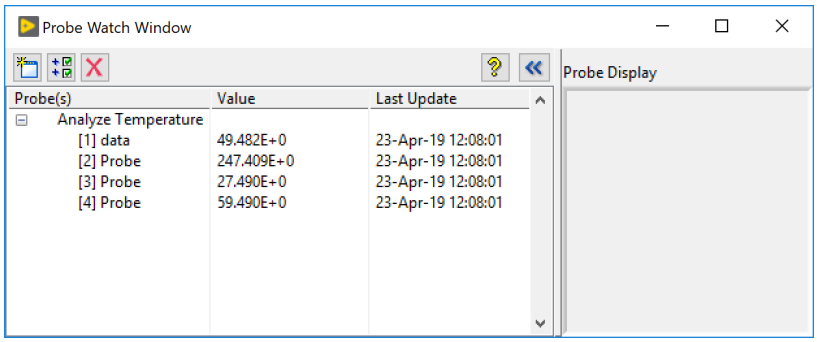
   ☐ Add probes to the wires, as shown in the following figure, by right-clicking each wire and selecting **Probe**.



   ☐ The **Probe Watch Window** opens after you set a probe, which shows an item for each probe you placed.

   ☐ Run the VI.

   ☐ Notice that the **Probe Watch Window** updates with the latest data values on the corresponding wires. The row marked **[1] data** shows the data value for the wire labeled with **Probe 1**.



   ☐ Double-click any probe item in the **Probe Watch Window** to highlight the corresponding probe on the block diagram.

   ☐ You can view the latest data values of a probe by looking at the values in the **Probe Watch Window**.

   ☐ By examining these probes, you determine that Probe 1 shows the correct temperature in degrees 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.

   Question 1 - What is the mistake?

   ☐ Remove all the probes by right-clicking each probe in the **Probe Watch Window** and selecting **Remove**.

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.

   ☐ Turn on Highlight Execution, and on the front panel toolbar, click the **Step Over** button to start single-stepping through the VI. Execution highlighting shows the flow of data on the block 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 block 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 block diagram, it flashes to indicate that all the nodes in the block diagram have finished executing. Click the **Finish VI "Analyze Temperature (incorrect behavior).vi"** button to finish running the VI.

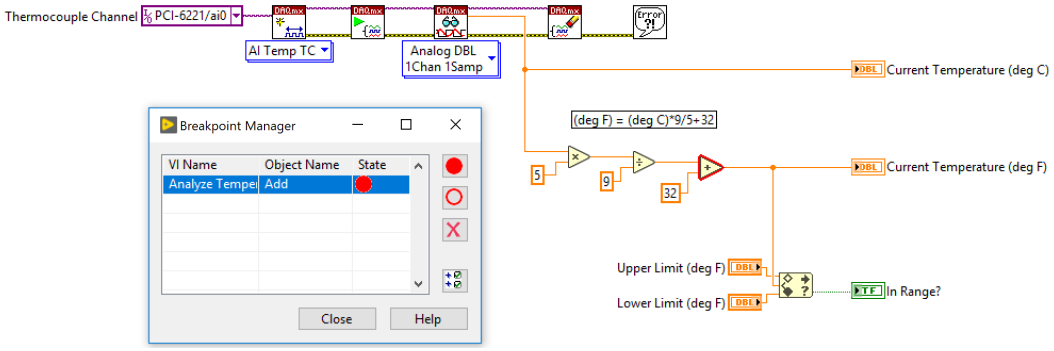
5. Set a breakpoint to pause the VI when execution reaches a specified point in the program.

   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 function executes.

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

   ☐ Notice that the Add function is highlighted red now and that the highlighted function corresponds to a breakpoint item in the **Breakpoint Manager** (Right-click on the highlighted function, then select **Breakpoint»Breakpoint Manager**).



   ☐ Turn off the **Highlight Execution** 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 the **Step Over** button to start single-stepping through the VI.
- ☐ When finished, remove the breakpoint by right-clicking the breakpoint on the block diagram and selecting **Breakpoint»Clear Breakpoint**.

Note If you select **Disable Breakpoint** instead, the breakpoint will remain on the block 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.

Answers

Question 1 - **Answer:** The values obtained from those probes show that the algorithm used to convert from ºC to ºF is incorrect.

End of Exercise 6-1

7 Executing Code Repeatedly Using Loops

In this lesson, you learn how to execute code repeatedly using loops.

Exercises

Exercise 7-1: Introduction to While Loops

Goal

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

Instructions

Explore the behavior of a VI without a While Loop

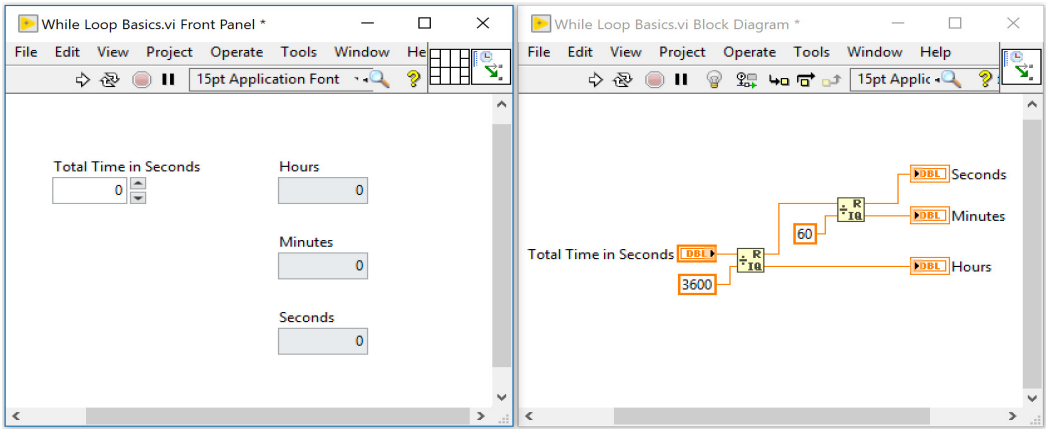
1. Explore the While Loop Basics VI.

   ☐ Open `<Exercises>\LabVIEW Core 1\While Loop Introduction\While Loop Introduction.lvproj`.

   ☐ From the **Project Explorer** window, open the While Loop Basics VI.

   ☐ Press <Ctrl-T> to display both the front panel and the block diagram at the same time.

2. Explore the front panel and block diagram.



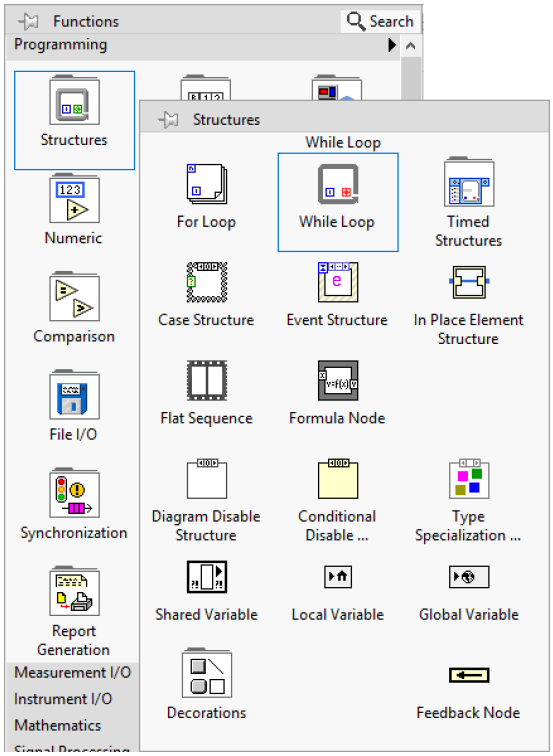
   ☐ Run the VI several times with different values in the **Total Time in Seconds** control. (e.g. 10, 70, 3750).

   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?

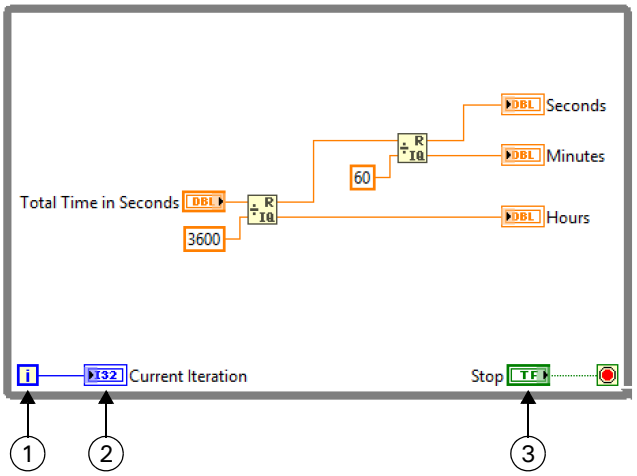
Execute Code Repeatedly

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

   ☐ Select **While Loop** from the **Programming»Structures** menu in the **Functions** palette.



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

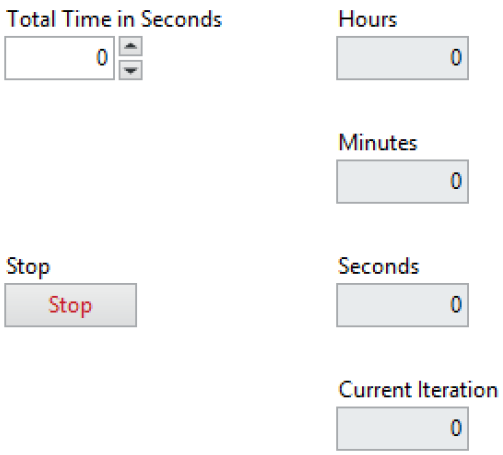


1 **Iteration terminal**—Provides the current loop iteration count. The first iteration of a loop is 0.
2 **Current iteration** indicator—Right-click the iteration terminal and select **Create Indicator**. Rename the indicator `Current Iteration`.
3 **Conditional terminal**—Right-click the terminal and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

2. Update the front panel.

   ☐ Switch to the front panel.

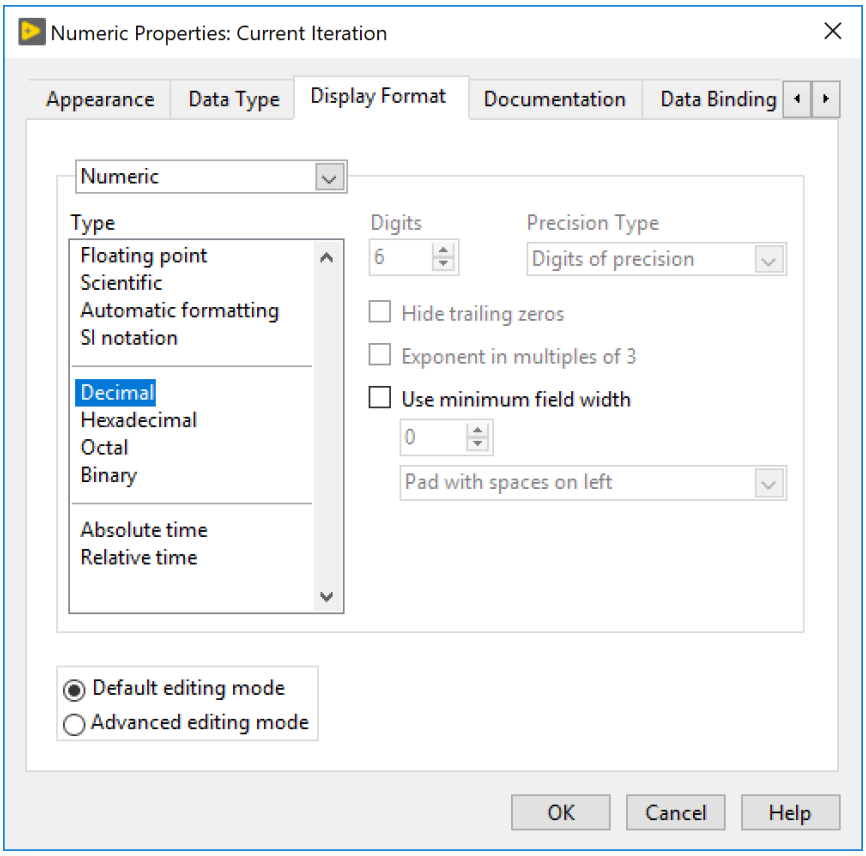
   ☐ Organize the front panel as shown in the following figure.



3. Increase the significant digits on the **Current Iteration** indicator so the iteration number displays correctly.

   ☐ Right-click the **Current Iteration** indicator and select **Display Format**.

   ☐ On the **Display Format** tab, select **Decimal** under the **Type** section and press **OK**.



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 execution highlighting.

   ☐ Press Ctrl-T to display the front panel and the block diagram at the same time.

   ☐ Turn on execution highlighting by clicking the **Highlight Execution** button.

   ☐ Run the VI.

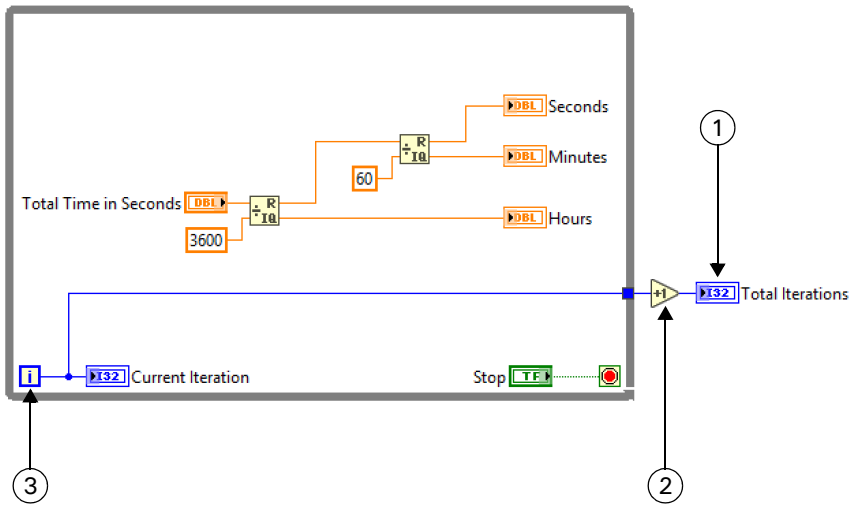
☐ On the block diagram, notice that the **Current Iteration** indicator returns a 0 on the first iteration and increases by 1 for each subsequent iteration.

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

6. Click the **Highlight Execution** button again to disable execution highlighting. Save the VI.

Explore the Functionality of While Loop Output Tunnels

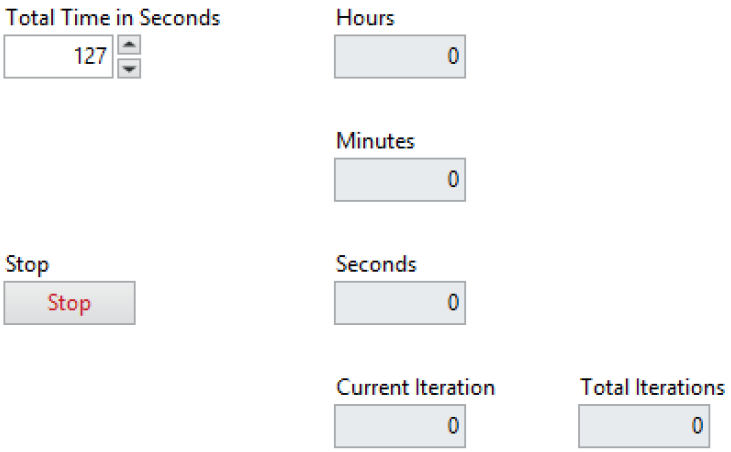
1. Count the total number of iterations.



1 **Numeric** indicator—Add a numeric indicator to the front panel.
2 **Increment** function—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 function.

2. Update the front panel.

☐ Arrange the front panel as shown in the following figure.



- ☐ Select the **Total Iterations** indicator and configure the display format to **Decimal** as you did with **Current Iteration** indicator.

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.

   - ☐ The While Loop does not execute until all of its inputs, which are input tunnels, are available. A While Loop does not return anything through an output tunnel until it finishes executing, when it finishes its last iteration due to an input to the conditional terminal.

   - ☐ Click the **Highlight Execution** button and run the VI again.

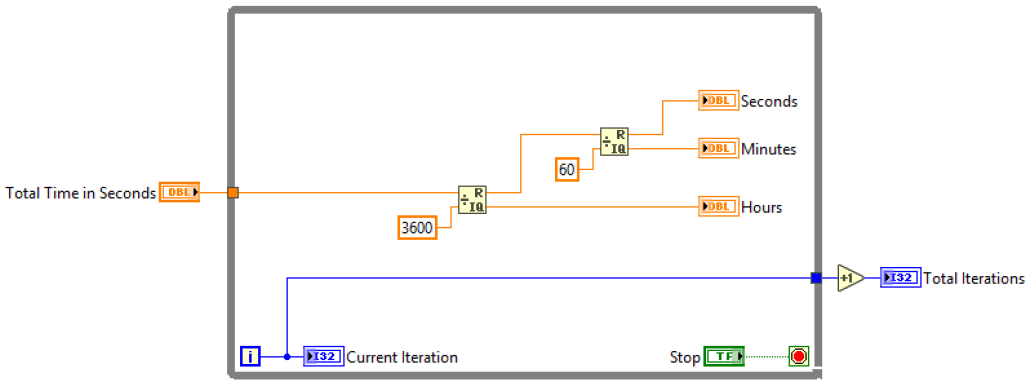
     Observe the flow of data on the block diagram at the output tunnel when you click the **Run** button. Observe the flow of data when you click the **Stop** button.

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 block diagram looks like the following figure.



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

   ☐ Press <Ctrl-T> to display the front panel and the block diagram at the same time.

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

   ☐ Turn on execution highlighting, so you can watch the dataflow for this block 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 returns 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 return?

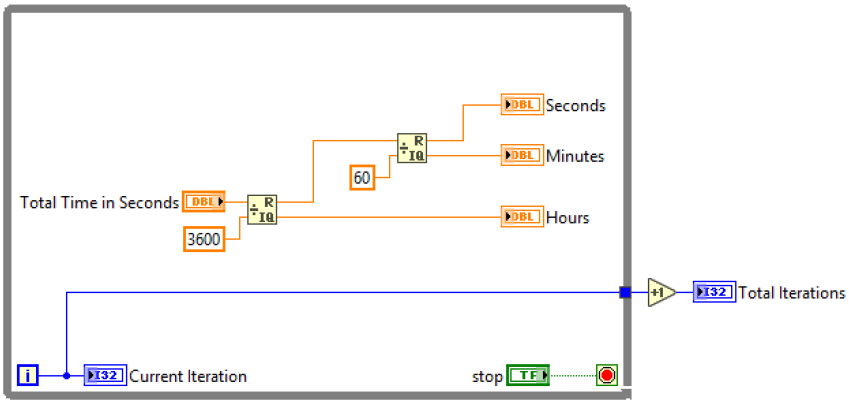
   _____

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

   Notice that the input tunnel still returns 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 return 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 Functions and VIs in a Loop

Goal

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

Instructions

Set Loop Period

In this part of exercise, you set a specific loop period for your While Loop.

1. Explore the Set Loop Period VI.

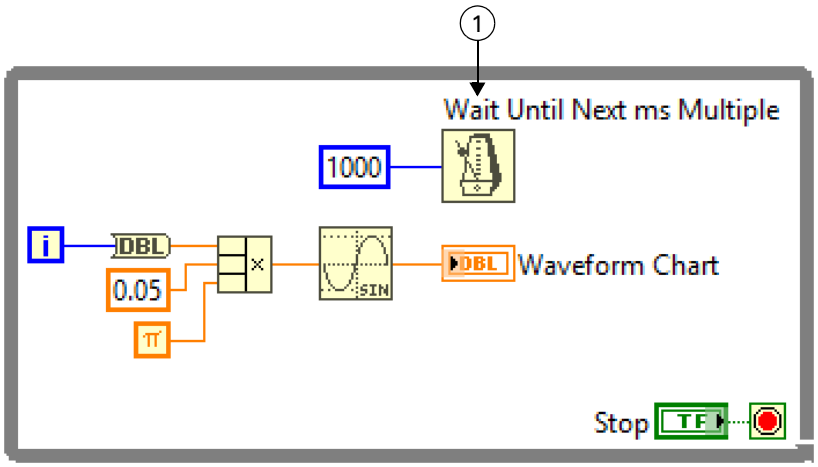
   ☐ Open `<Exercises>\LabVIEW Core 1\Using Timing Functions in a Loop\Using Timing Functions in a Loop.lvproj`.

   ☐ From the **Project Explorer** window, open the Set Loop Period VI.

2. Run the VI.

   Notice that the loop runs as fast as possible.

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



1 **Wait Until Next ms Multiple** function—Right-click the **millisecond multiple** input and select **Create Constant**. Set the constant to `1000`.

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

Note that in the first iteration, the Wait Until Next ms Multiple function will probably wait less than 1,000 milliseconds. For example, if the OS timer is at 5800 when the first iteration starts, then the Wait Until Next ms Multiple function will only wait 200ms on the first iteration because the next multiple

of 1,000 is 6,000. However, the Wait Until Next ms Multiple function 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.
   - ☐ Press <Ctrl-T> to display the block diagram and front 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 block diagram, change the **millisecond multiple** input 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 ms 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 to Complete Other Tasks

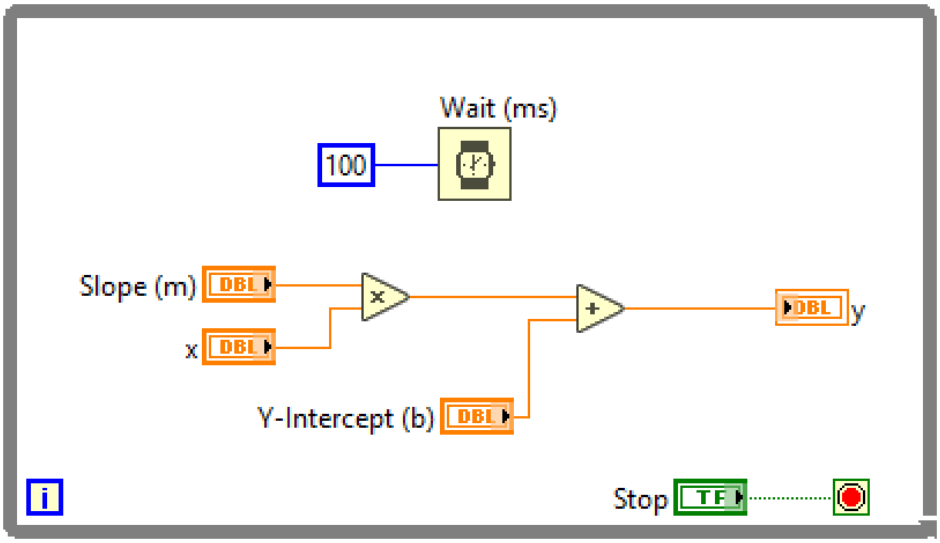
In this section, you want to ensure that your loop allows the CPU to complete other tasks. In other words, you want to make sure that your VI does not needlessly have a high CPU usage.

1. Explore the CPU Usage of a Loop VI.
   - ☐ From the **Project Explorer** window, open CPU Usage of Loop VI.
   - ☐ Examine the block diagram.
     Notice that there is no Wait function 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 high CPU usage.

CPU Usage of LabVIEW.exe: _____

Note **(Windows 10)** Right-click the Windows taskbar and select **Task Manager**.

4. Stop the VI when finished.
5. Modify the block diagram, as shown in the following figure, to add a **Wait (ms)** function 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 functions that return the current date/time and elapsed time.

1. From the **Project Explorer** window, open Using Timestamp and Elapsed Time Functions VI and examine the block diagram.
   - ☐ Notice that the While Loop continuously performs a slope calculation.
   - ☐ The Get Date/Time in Seconds function returns the current date and time. This function does not wait.

- ☐ The Elapsed Time Express VI returns how much time has elapsed since the first time the function executed in the loop. This VI does not wait.
  When the elapsed time is greater than or equal to the target time, then the Elapsed Time Express VI will return True from the **Target Time Has Elapsed** terminal. Since this output is wired to the conditional 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 function and Elapsed Time VI work.

  Question 1 - Because none of the functions or VIs in this While Loop wait, how many iterations per second does this While Loop execute?

2. Examine the behavior of the Using Timestamp and Elapsed Time Functions VI.
   - ☐ On the front 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?

_____

_____

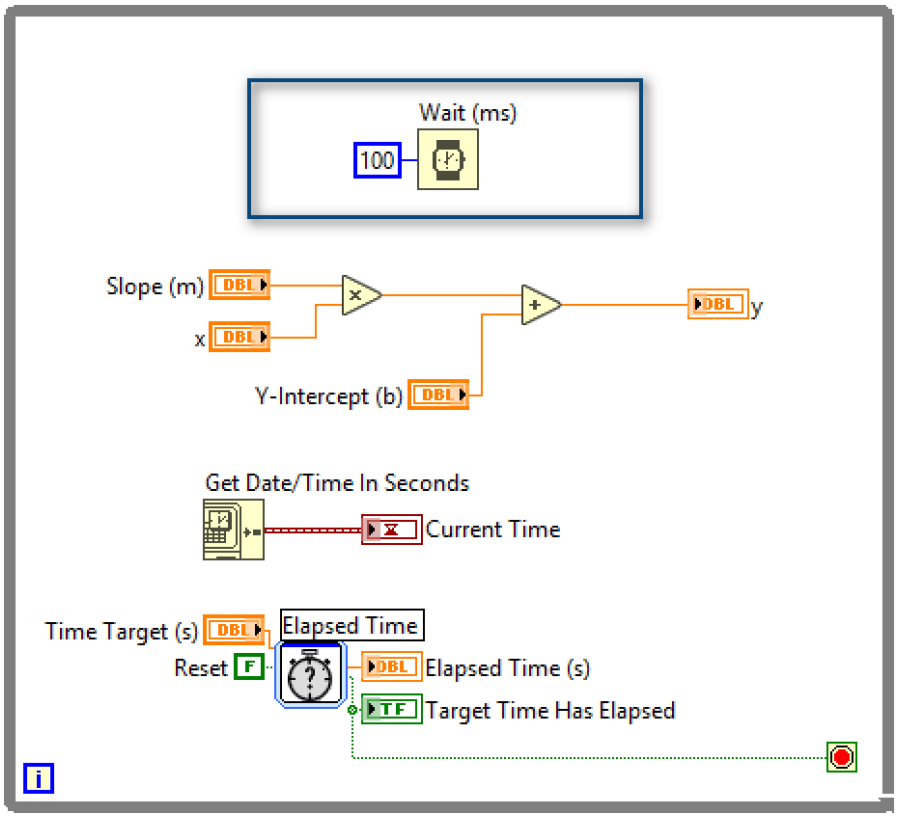
   - ☐ Use your ideas to modify the VI to reduce the CPU usage.
     Did your modification work? What is the CPU usage of the LabVIEW.exe process now?

_____

See the next page for the solution.

Answer

To reduce the CPU usage of the Using Timestamp and Elapsed Time Functions VI, you can add a Wait (ms) function 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.



Answers

Question 1 - **Answer**: The While Loop executes its iterations as fast as possible because the While Loop does not contain a function or a VI that waits.

End of Exercise 7-2

Exercise 7-3: Continuously Acquiring Data using DAQmx API Timing

Goal

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

Predict the time to wait before reading data from the DAQmx device and the While Loop frequency based on the sample rate and number of samples you specify.

Instructions

Examine the Finite Acquisition VI

1. Explore the Finite Acquisition VI.

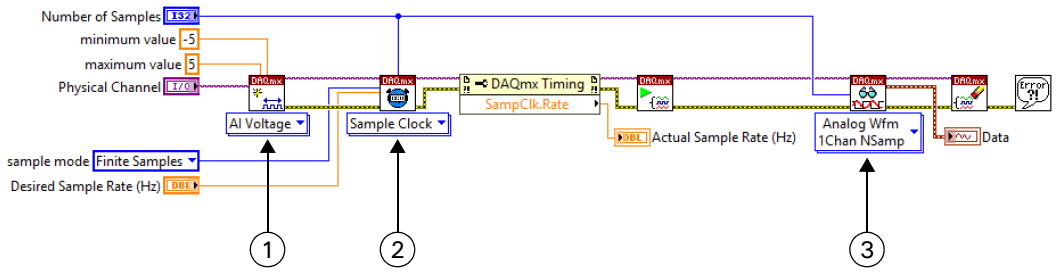
   ☐ Open `<Exercises>\LabVIEW Core 1\While Loop with Hardware API\While Loop with Hardware Timing.lvproj`.

   ☐ Open the Finite Acquisition VI from the **Project Explorer** window.

   ☐ Notice the controls and waveform graph indicator on the front panel.

   ☐ Switch to the block diagram and notice that the code performs a finite acquisition.

   ☐ Examine the configuration of the following three VIs.



1 **DAQmx Create Channel** VI—Note the Polymorphic VI selector configuration.
2 **DAQmx Timing** VI—This VI sets the sample rate of the acquisition. The sample mode must be set to **Finite Samples** for a finite acquisition.
3 **DAQmx Read** VI—This VI reads data from the DAQ device. Notice that the VI is configured to read multiple samples from single channel. As a result, the DAQmx Read VI has a number of samples per channel input, which tells this VI 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 front panel, specify the following values to the controls:

   – **Physical Channel:** PCI-6221/ai1

   – **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 VI waits until 1,000 samples of data is available to read from the DAQ device. Then the DAQmx Read VI returns 1,000 samples of data to the waveform graph indicator.
- ☐ Set the **Number of Samples** control to 5,000, and run the VI again.

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

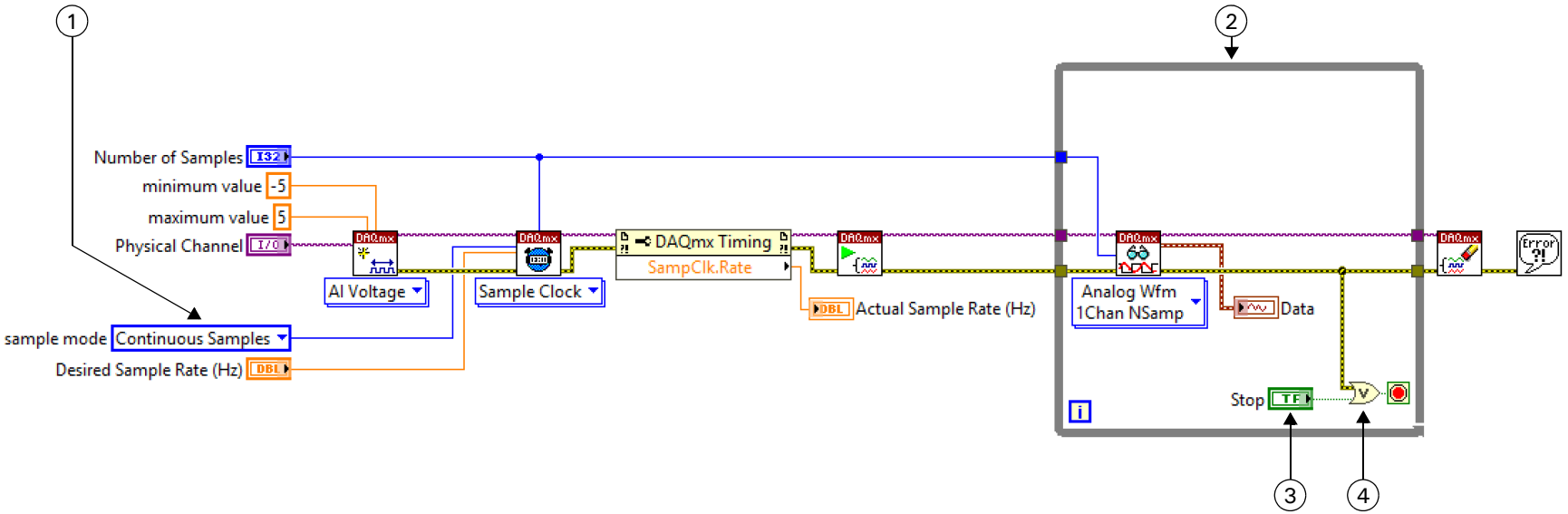
Note When the sample rate is 1,000 Hz (1,000 samples per second), 5,000 samples of data represents 5 seconds of data.

3. Close the VI when you are finished.

Creating a VI to Acquire Data Continuously

1. Create a copy of the Finite Acquisition VI.
   - ☐ In the **Project Explorer** window, right-click the Finite Acquisition VI and select **File»Save As**.
   - ☐ In the appeared dialog box, select the **Open additional copy** option and place a checkmark in the **Add copy to While Loop with Hardware Timing.lvproj** checkbox, then press **Continue**.
   - ☐ Name the VI as `Continuous Acquisition`, click **OK**.

2. 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 VI. Because the **Number of Samples** control is outside the While Loop, the DAQmx Read VI 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** control—Right-click the condition?? terminal and select **Create Control.**
4 **OR** function—Add an OR function between the **Stop** control and conditional terminal and wire the error wire to input x. This stops the While Loop when the user clicks the **Stop** button or when an error occurs.

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

   ☐ On the front panel, specify the following values to the controls:

   - **Physical Channel:** PCI-6221/ai1
   - **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 waveform data graph updates about once per second.

   This is because the DAQmx Read VI 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 VI must wait approximately 1 second for 1,000 samples to be available to read and then returns 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 VI waits, which allows the CPU to execute other tasks.

   Note **(Windows 10)** Right-click the Windows taskbar and select **Task Manager** to launch the Task Manager.

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

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

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

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

5. Test the combinations of different sample rate and number of samples. 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 waveform graph on the front panel update? _____

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

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

6. Save the VI and close the project.

End of Exercise 7-3

Exercise 7-4: Using Shift Registers

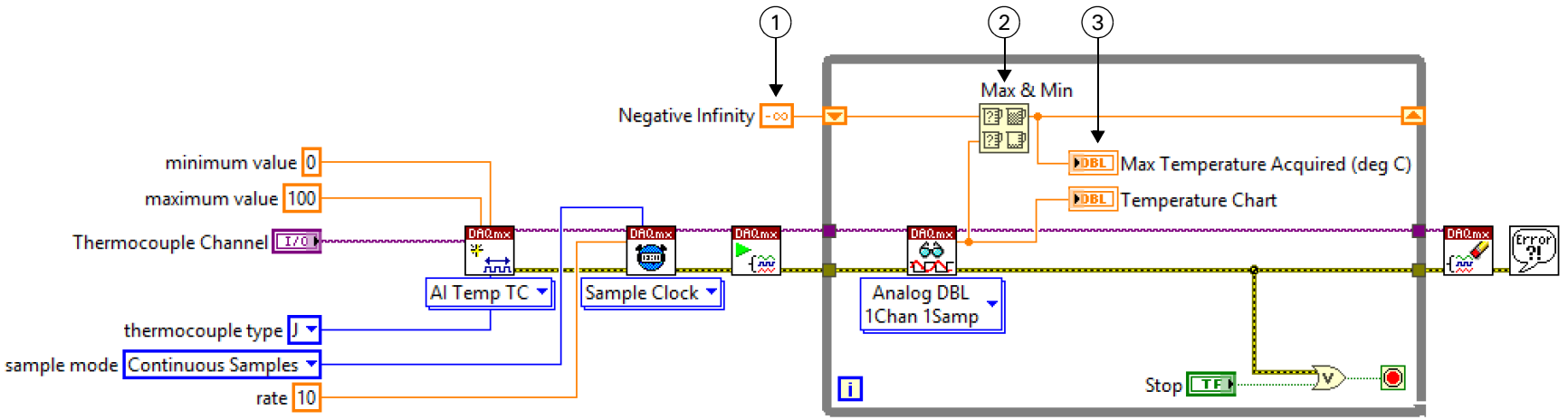
Goal

Use a shift register to pass values from previous iterations through a While Loop to the next iteration. This way, you can keep track of the maximum acquired value.

Instructions

1. Open `<Exercises>\LabVIEW Core 1\Using Shift Registers\Using Shift Register.lvproj`.
2. From the **Project Explorer** window, open Track the Maximum Temperature VI.
3. Explore and run the VI.
   - ☐ On the block diagram, modify the **minimum value**, **maximum value**, and **thermocouple type** constant values to match your hardware.
   - ☐ On the front panel, set the **Thermocouple Channel** control accordingly.
   - ☐ Run the VI.
   - ☐ Notice that this VI performs a continuous acquisition at a rate of 10 Hz. The DAQmx Read VI reads one temperature value per loop iteration.
   - ☐ Click the **Stop** button on the front 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 **Add Shift Register**.

5. Modify the block 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 VI 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 & Min** function—Use this function to compare the maximum acquired value with the current value and return the higher value of the two.
3 **Max Temperature Acquired** indicator—Right-click the **max(x,y)** output of the **Max & Min** function and select **Create Indicator**. Rename the indicator as `Max Temperature Acquired (deg C)`.

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

   ☐ Turn on execution highlighting by clicking the **Highlight Execution** button.

   ☐ Run the VI. Touch the thermocouple to make the temperature increase and decrease, and observe how the logic of the shift register, Max & Min function, and DAQmx Read VI 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.

   ☐ Click the **Highlight Execution** button again to disable execution highlighting. The VI runs at regular speed.

Your Turn

1. From the **Project Explorer** window, open the Threshold Exceeded VI.
2. Run and explore the VI. **The Threshold Currently Exceeded?** indicator turns on if the DAQmx Read VI output exceeds the **Max Threshold** control value.
3. Add a Boolean indicator named `Threshold Ever Exceeded?` to the front panel.
4. Modify the VI so that the **Threshold Ever Exceeded?** indicator turns on and stays on if the DAQmx Read VI 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 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) Using Stacked Shift Registers

Goal

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

Instructions

1. Open `<Exercises>\LabVIEW Core 1\Stacked Shift Registers\Using Stacked Shift Registers.lvproj`.
2. From the **Project Explorer** window, open **R**unning Average Using Stacked Shift Register VI.
3. Explore the stacked shift register on the block diagram.
   - ☐ Notice the stacked shift register.
   - ☐ The first (top) left shift register returns data from the previous iteration.
   - ☐ The second left shift register returns data from the second-most recent iteration. If there was a third left shift register, it would return data from the third most recent iteration.
4. Examine the functionality of the block 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 VI in the While Loop returns the current measurement, the first (top) left shift register returns the previous measurement, and the second left shift register returns the measurement before the previous measurement.
   - ☐ The **Average Temperature Chart** indicator 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** indicators. The **Average Temperature Chart** indicator should be smoother due to the averaging.
   - ☐ Turn on execution highlighting by clicking the **Highlight Execution** button to examine how the stacked shift register stores and returns 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, 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 Element**.

Your Turn

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

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

End of Exercise 7-5

8 Working with Groups of Data

In this lesson, you learn how to work with array and waveform data types and how to work with single-channel and multi-channel acquisition data.

Exercises

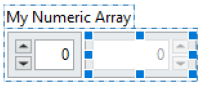
Exercise 8-1: Creating and Viewing an Array

Goal

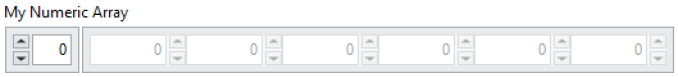
Create and view a numeric array.

Instructions

1. Create a new project. Save the project as `Create Array.lvproj` in the `<Exercises>\LabVIEW Core 1\Create Array` directory.
2. Create and open a new VI. Rename the VI as `Create Array.vi`.
3. Create an array of numeric controls.
   - ☐ Add an **Array** shell to the front 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 so that a blue line outlines the array, as shown below.

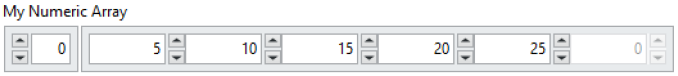


   - ☐ To resize the array horizontally, drag the dot on the far right until 6 elements appear on the front panel.



   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.

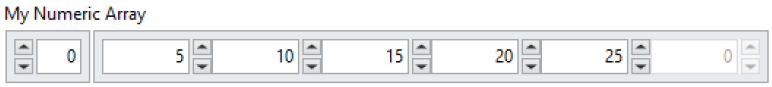


   Question 1 - Check your understanding of the array index.

   - ☐ 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? _____

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

6. 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.

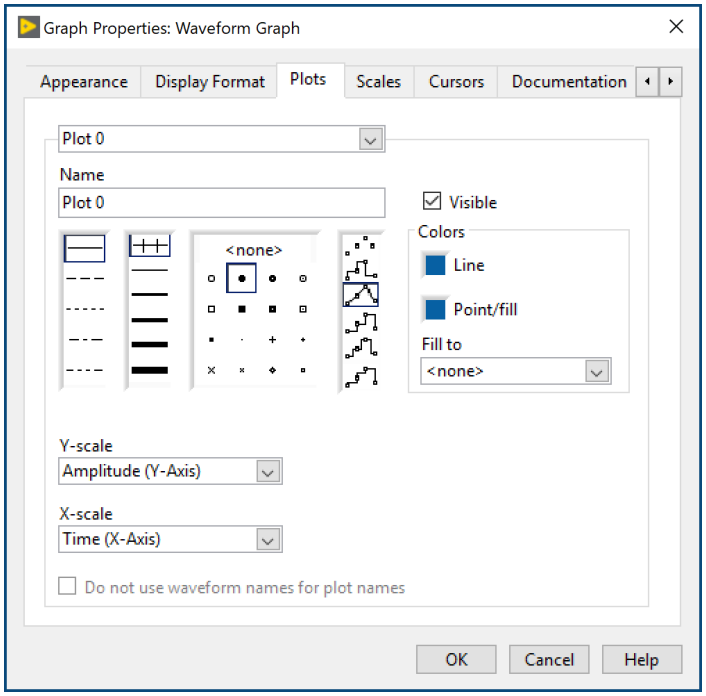
7. View how the array data appears on a graph indicator.

☐ Place a waveform graph indicator on the front panel.

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

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

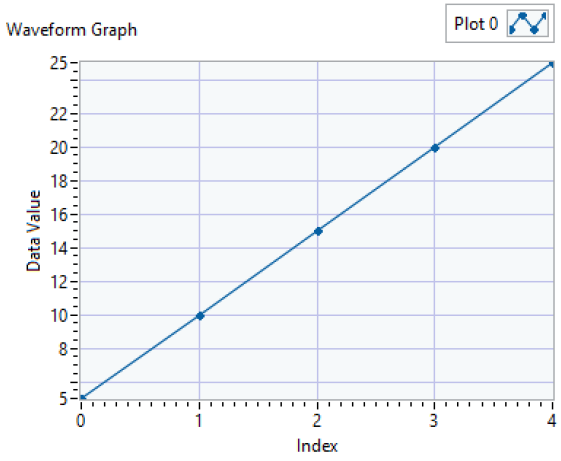
☐ Right-click the waveform graph indicator and select **Properties** to open the **Graph Properties** dialog box. In the **Plots** tab you can configure the visual representation of the plot. For now, configure plot as shown in the picture below for the indicator to display sequentially connected points, which represent the data values in the array control.



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

– On the x-axis of the waveform 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 points at x-axis values of 0, 1, 2, 3, and 4.



Creating a Two-Dimensional (2D) Array

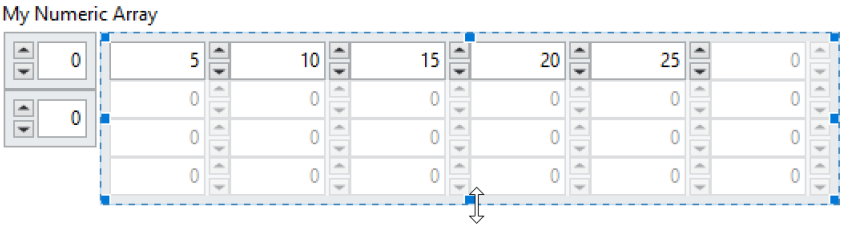
1. Change the number of dimensions from 1 to 2.

   ☐ Right-click index display of My Numeric Array and select **Add Dimension**.

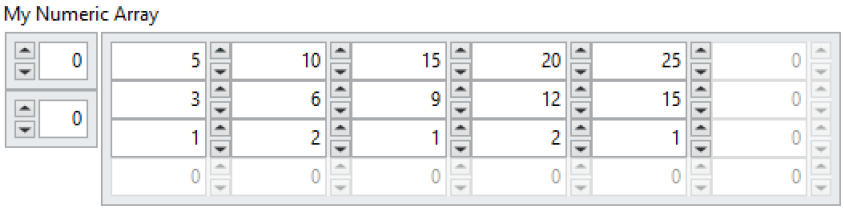
2. View two dimensions (rows and columns) of data.

   ☐ Select the entire array control so that a blue line outlines the array.

   ☐ Drag the middle-bottom dot downwards until 4 rows are displayed on the front 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.

   Question 2 - What is the row index and column index of the element containing 12?

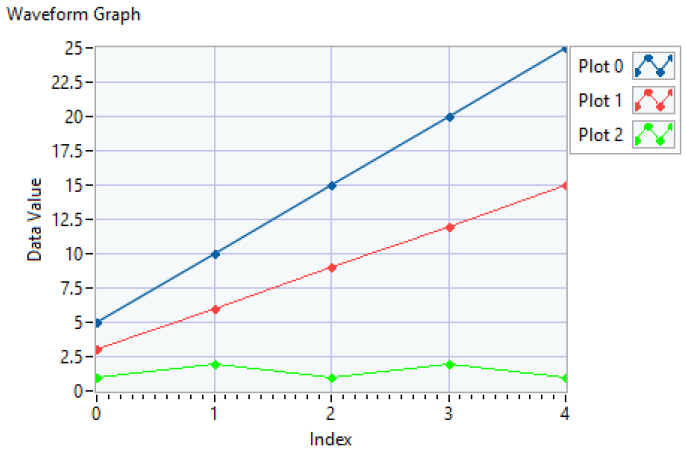
Row Index	
Column Index	

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

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

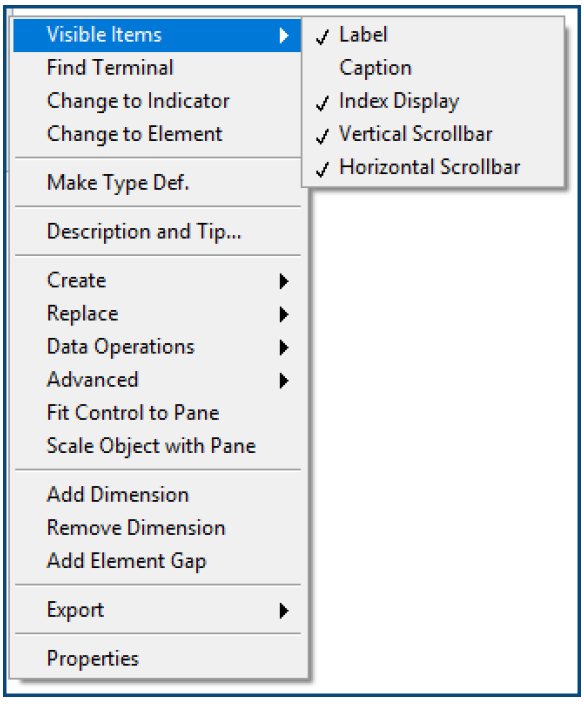
☐ Hover your cursor over the plot legend, and resize it to show three plots.

☐ Right-click the **Waveform Graph** indicator and select **Properties**. Configure each plot as the first one in the **Plots** tab of the **Properties** dialog box. The Waveform 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. Right-click the body of **My Numeric Array** control.
2. In the **Visible Items** section of the drop down menu, you can configure whether to show the label, caption, 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.

Answers

Question 1 - **Answers:** 15, 25, 5

Question 2 - **Answer**: row index is 1; column index is 3

End of Exercise 8-1

Exercise 8-2: Examining the Waveform Data Type

Goal

Recognize the components and visualization of the waveform data type.

Instructions

1. Open `<Exercises>\LabVIEW Core 1\Waveform Data Type\Waveform Data Type.lvproj`.
2. Open the Examine Waveform Data Type VI from the **Project Explorer** window.
3. Examine the VI.
   - ☐ Examine the front panel, which consists of a waveform control and waveform graph indicator.
   - ☐ Examine the block diagram. Notice that the waveform control passes its data directly to the graph indicator.
   - ☐ Examine the waveform control, which has a waveform data type. Take notes 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. The x-axis of the waveform 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 waveform 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).

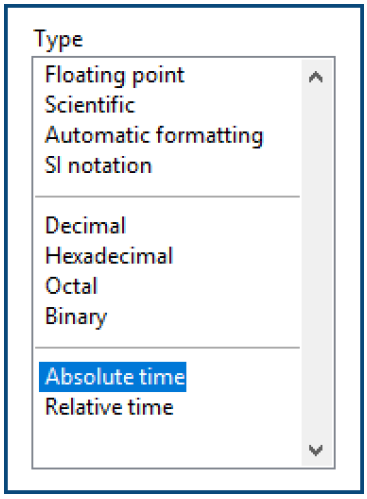
   Currently the x-axis of the waveform graph indicator shows relative time in units of seconds, where the initial time is always displayed as 0.

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

   – Right-click the **Waveform Graph** indicator and select **Properties**.

   – In the **Display Format** tab select **Default editing mode.**

   – Set Type 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 waveform 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, right-click the waveform graph indicator, select **Properties**, and set the type back to **Automatic formatting** in the **Display Format** tab. You can also try experimenting with the other **Display Format Type** options.

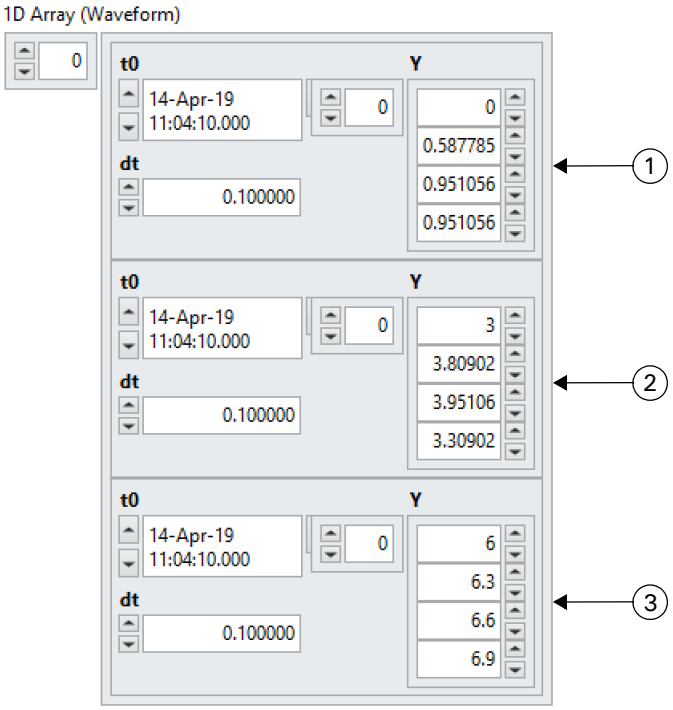
7. Try making additional modifications to t0, dt, and Y array, and examine the effects on the graph indicator.

1D Array of Waveform

1. Open the Examine 1D Array (Waveform) Data Type VI from the **Project Explorer** window.

2. Examine the 1D Array (Waveform) control on the front 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 **Waveform Chart** indicator.

4. Observe the behavior of the VI.

   ☐ Run the VI.

   ☐ Notice that when you pass a 1D array of waveforms to a chart, the chart 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, e.g. 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 Functions and VIs to Analyze Data

Goal

Use analysis tools on your VI block diagram to analyze data.

Instructions

Analysis tools with the Waveform Data Type

1. Open `<Exercises>\LabVIEW Core 1\Analysis tools\Analysis tools.lvproj`.
2. From the **Project Explorer** window, open the Using Analysis tools (Waveform data type) VI.
3. Examine the front panel and the block diagram.
   - ☐ Set **Physical Channel**, **Desired Sample Rate (Hz)**, and **Number of Samples** controls appropriately to acquire a signal from your DAQ device.

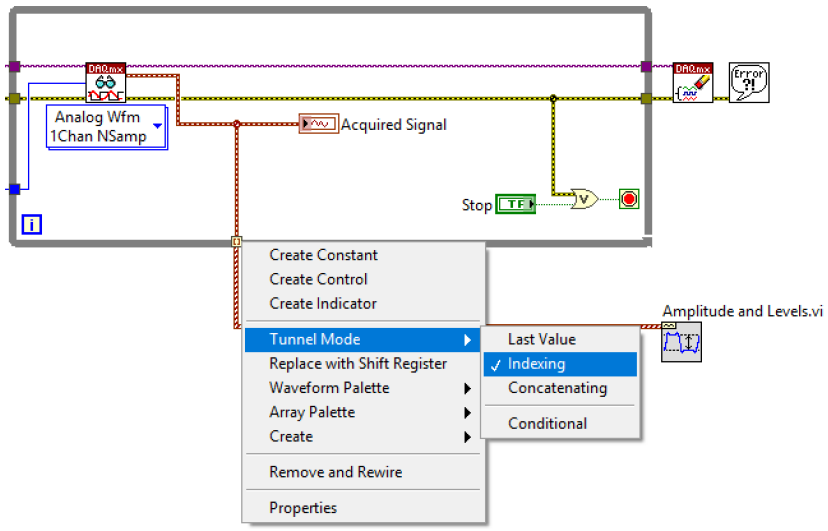
Note Make sure that you have your thermocouple connected to the Thermocouple Input Connector, and that the BNC/Thermocouple Switch is in the right position.

	Simulated Hardware	BNC-2120
Physical Channel	PCI-6221/ai1	PCI-6221/ai1 (Thermocouple Input)
Desired Sample Rate (Hz)	1000	2560
Number of Samples	1000	256

   - ☐ Notice that this VI performs a continuous acquisition.
   - ☐ Notice that the **Acquired Signal** indicator is a waveform data type.
4. Observe the behavior of the VI.
   - ☐ Run the VI.
   - ☐ On the front panel, observe the behavior or the **Acquired Signal** waveform chart.
   - ☐ Stop the VI.
5. Analyze your data using analysis tools form the Signal Processing palette.
   - ☐ Right-click in block diagram, find the Amplitude and Levels.vi under **Signal Processing»Waveform Measurements**, and drag it to the block diagram. Place it outside the loop.

☐ Wire the **data** output of DAQmx Read VI to the **signal(s) in** input of the Amplitude and Levels VI.

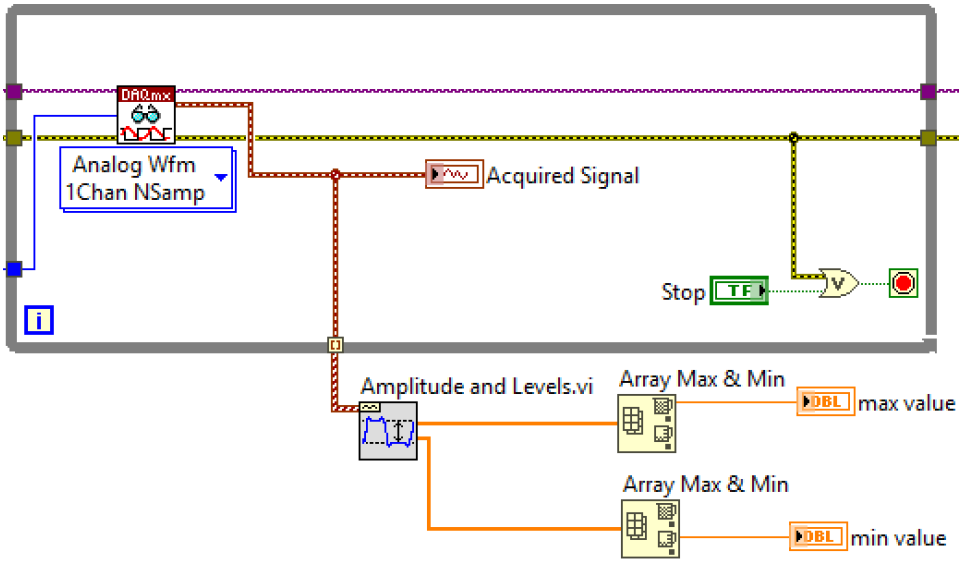
☐ Make sure that the Amplitude and Levels VI is outside the While Loop and that the mode of the tunnel connecting the DAQmx Read VI to the Amplitude and Levels VI is set to Indexing. This way the VI will analyze all acquired data.



☐ Notice as we set the tunnel mode to Indexing, the **high state level** and **low state level** outputs of the Amplitude and Levels VI are 1D DBL arrays. This will allow us to find the maximum and minimum values among all the acquisition.

☐ Place two copies of the Array Max & Min function to the block diagram. These functions return the maximum and minimum values of the high state level and low state level data.

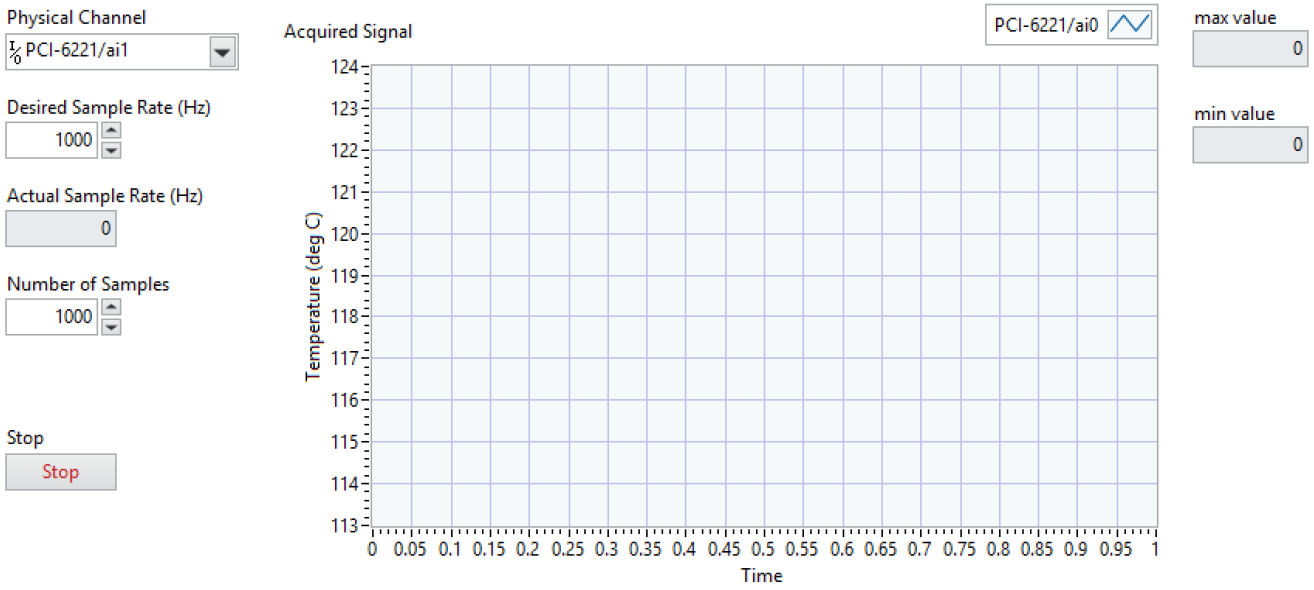
☐ Wire the **high state level** output of the Amplitude and Levels VI to the input of the first Array Max & Min function and the **low state level** output to the input of the second Array Max&Min function.



☐ Right-click the **max value** output of the first Array Max & Min function and select **Create Indicator**. Do the same for the **min value** output of the second Array Max & Min function. Your block diagram should match the figure above.

6. View the results of the analysis tool.

☐ Switch to the front panel and arrange the items as shown in the following figure.



☐ Run the VI and keep it running for a couple of seconds. Notice that the **max value** and **min value** indicators are not showing any data.

☐ Stop the VI.

☐ Now the **max value** and **min value** indicators display the maximum and the minimum values of the entire dataset recorded.

☐ Save the VI.

Your Turn

Experiment with using other analysis tools.

End of Exercise 8-3

Exercise 8-4: Visualizing *N*-Channel Data

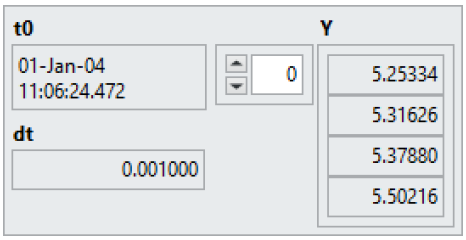
Goal

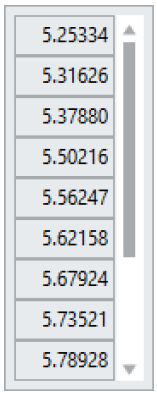
- Become familiar with the different data types that represent N-channel data.
- View multiple channels of data in chart, graph, and array indicators.

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 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

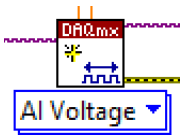
1. Open `<Exercises>\LabVIEW Core 1\Visualize N-Channel Data\Visualize N-Channel Data.lvproj`.
2. Follow the instructions for the scenarios you selected in the **Scenario** section.

N Channel, 1 Sample (1D DBL Array)

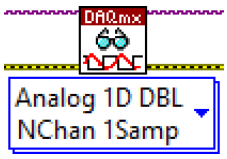
In this section, you create a VI that acquires and visualizes one data sample from each of the three different data sources at a rate of 1 Hz.

(BNC-2120) and **(Simulated Hardware)** Modify this VI to acquire 3 channels of voltage (Sine Wave Function, TTL Square Wave Function and Temperature Reference) from PCI-6221/ai0, PCI-6221/ai2:3.

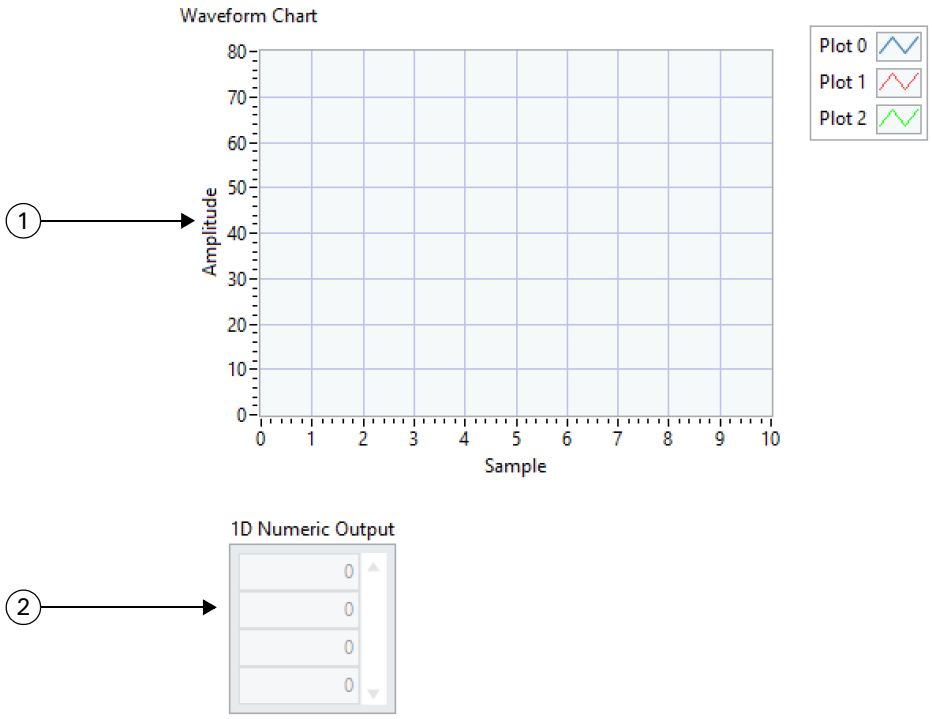
1. From the **Project Explorer** window, open the N-Channel 1-Sample (1D DBL) VI.
2. Examine the properties of the DAQmx Create Channel VI.
   - ☐ Click polymorphic VI selector of the DAQmx Create Channel VI and explore the selected instance.



   - ☐ Notice that this VI is configured for a Voltage measurement. You can modify this configuration if you want to acquire from a different type of measurement.
3. Examine the DAQmx Timing VI.
   - ☐ Notice that the sample rate is set to1 Hz. This sets the DAQ device to acquire 1 sample per channel per second.
4. Examine the DAQmx Read VI.
   - ☐ Click polymorphic VI selector of the DAQmx Read VI and explore the selected instance.
   - ☐ Notice that this VI is set to **Multiple Channels** and **Single Sample**. This VI will return a 1D DBL array that contains a single sample from multiple channels.

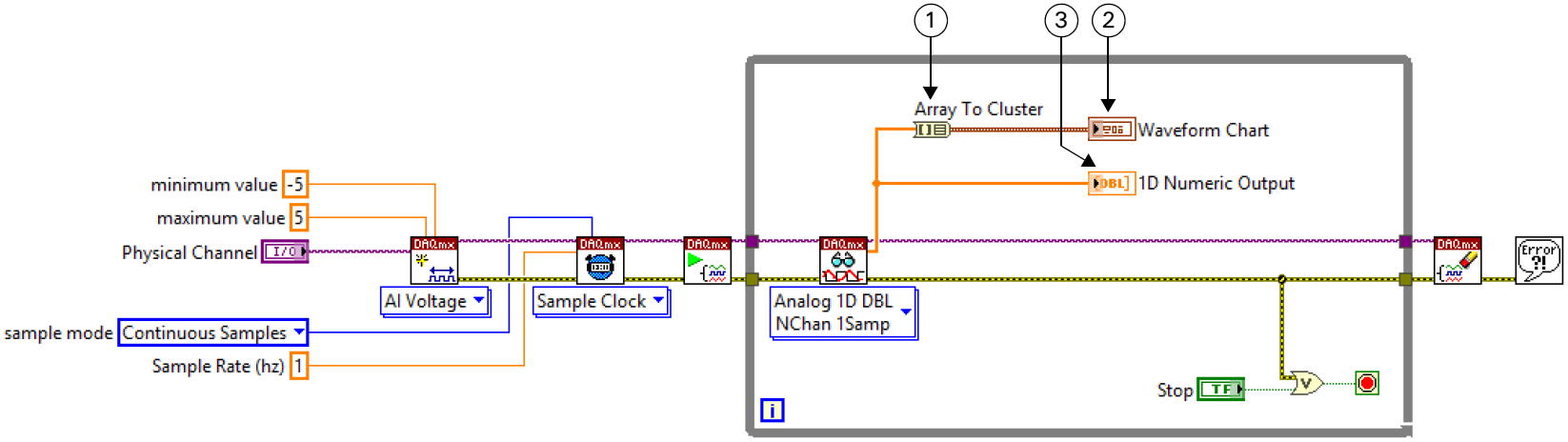


5. Create the front panel.



1 **Waveform Chart**—Add a Waveform Chart to the front panel. Expand the plot legend to display three plots.
2 **Numeric Array Output**—Add a Numeric Array control to the front panel, right-click the control and press **Change to Indicator**. Resize this indicator to show 3 or more elements and rename it `1D Numeric Output`.

6. Modify the block diagram, as shown in the following figure, to visualize the multiple channels single samples data.



1 **Array to Cluster**—This function converts the 1D array data type to a cluster data type. To plot a single sample from multiple channels on a waveform chart, you must pass the waveform chart a cluster data type containing the samples.
2 **Waveform Chart**—Chart that can be used to show historical data and append new data.
3 **1D Numeric Output**—This array indicator displays only the current array data in a specific iteration.

7. Configure the **Waveform Chart** properties.

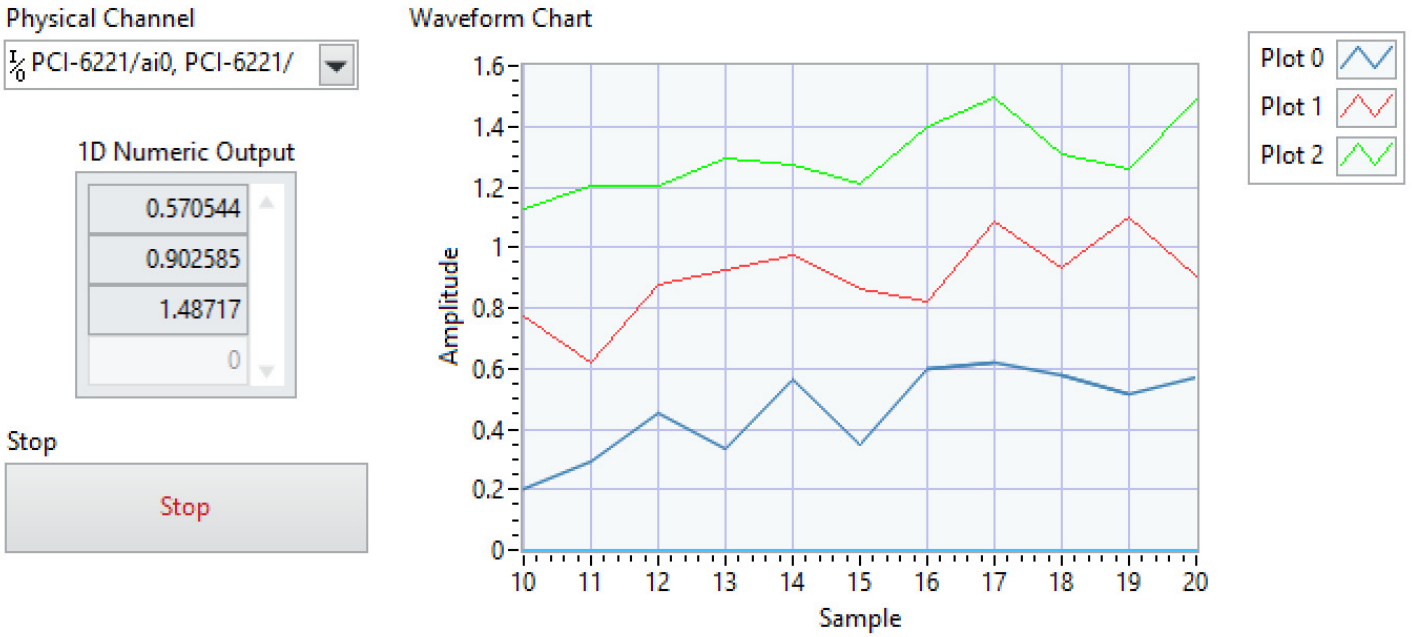
   ☐ Resize the plot legend to show 3 plots.

   ☐ Double-click **Time** on the x-axis and rename it as `Sample`. The 1D array (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 waveform chart will show the most recent 10 samples per channel acquired.

   ☐ Right-click the waveform chart and select **Chart History Length**. 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 front panel should look similar to the following figure when the VI is running.



9. Test the VI.

   ☐ On the front panel, set the **Physical Channel** control to **PCI-6221/ai0, PCI-6221/ai2:3**. This creates a DAQmx task that includes PCI-6221 channels 0, 2, and 3.

   ☐ Run the VI and notice the following things.

   - The rate input (1 Hz) of the DAQmx Read VI controls the loop period.
   - Every iteration, one sample from 3 channels are updated on the waveform chart.
   - Every iteration, the 1D DBL Array indicator only shows the data for the current iteration.
   - The x-axis of the waveform chart is in units of Sample number (not seconds).

- The waveform 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 waveform chart will show the most recent 5 samples per channel acquired.

☐ Stop the VI when finished.

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

N Channel, *N* Samples (1D Waveform Array)

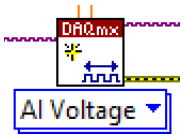
In this section, you create a VI that acquires and visualizes 3 channels of analog input voltage. Each loop iteration acquires/visualizes 1000 samples per channel. The samples are continuously acquired at a hardware timed rate of 1000 Hz. The waveform data type contains the sample values (Y), the start time for the samples (t0), and the time interval between each sample (dt).

(BNC-2120) and **(Simulated Hardware)** Modify this VI to acquire 3 channels of voltage (Sine Wave Function, TTL Square Wave Function and Temperature Reference) from PCI-6221/ai0, PCI-6221/ai2:3, at a sample rate of 2,560 Hz and number of samples of 256.

1. From the **Project Explorer** window, open the N-Channel N-Samples (1D Waveform) VI.

2. Examine the DAQmx Create Channel VI.

   ☐ Click polymorphic VI selector of the DAQmx Create Channel VI and explore the selected instance.

   Notice that this VI 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.

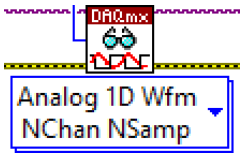
   ☐ The **Desired Sample Rate (Hz)** control sets sample rate of this DAQ task.

   ☐ The **Number of Samples** control sets how many samples per channel the DAQmx Read VI 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 VI acquires from multiple channels at a sample rate of 1000 Hz and outputs 1000 samples per channel.

4. Examine the DAQmx Read VI.

   ☐ Click polymorphic VI selector of the DAQmx Read VI and explore the selected instance.



   Notice that this VI is set to **Analog»Multiple Channels»Multiple Samples»1D Waveform (Samples)**.

5. Create the front panel.

   ☐ Add a waveform graph to the front panel. Resize the plot legend to show 3 plots.

   ☐ Add a waveform chart to the front panel. Resize the plot legend to show 3 plots.

   ☐ Create a 1D waveform array indicator.

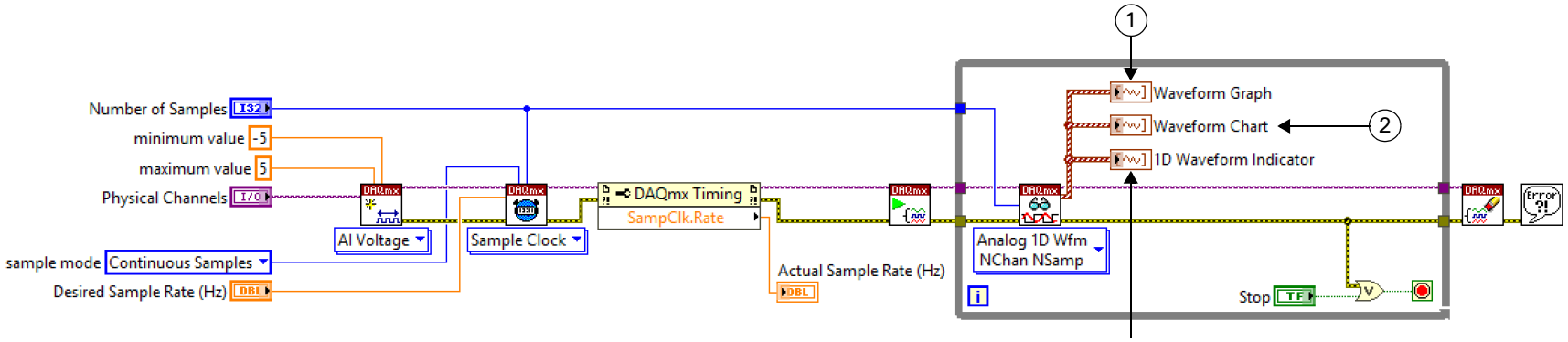
   – Add an array shell on the front panel. Rename it `1D Waveform Indicator`.

   – Add a waveform control inside the unconfigured array.

   – Resize the array to show 4 elements.

   – Right-click the array and select **Change to Indicator**.

6. Modify the block diagram, as shown in the following figure.



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

7. Configure the Waveform Graph properties.

   ☐ Switch to the front 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 waveform chart properties.

   ☐ Right-click the **Waveform Chart** indicator and select **Properties**, then in the **Display Format** tab make sure that the **Default editing mode** is selected and set Type to **Floating point**. Click OK to save changes.

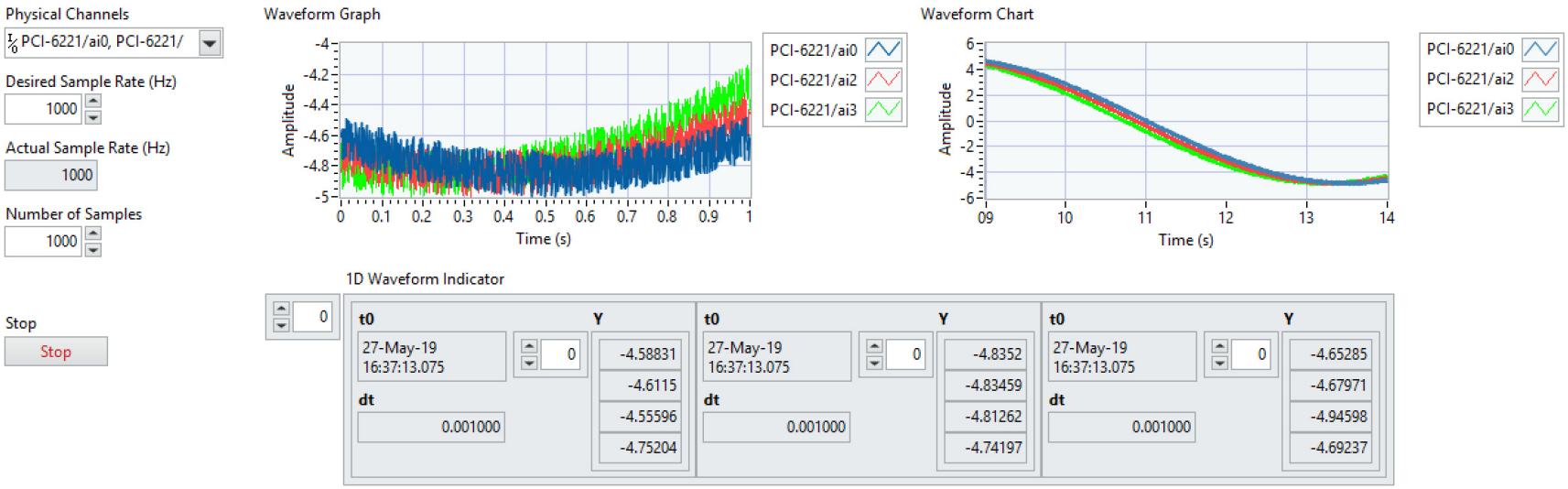
   ☐ Double-click **Time** on the x-axis and rename it `Time (s)`. The Waveform data type contains timing information in units of seconds.

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

   ☐ Right-click the waveform chart then select **Chart History Length** and change the history length from 1,024 to 5,000.
   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 the most recent 5 seconds of data.

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



10. Test the VI.

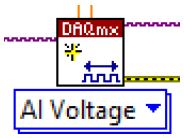
- ☐ On the front panel, set the **Physical Channels** control to **PCI-6221/ai0, PCI-6221/ai2:3**. This creates a DAQmx task that includes PCI-6221 analog input channels 0, 2, and 3.
- ☐ Run the VI.
- ☐ Notice that the following things happen every iteration.
  - Multiple samples from 3 channels are 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 3 channels of analog input voltage. Each loop iteration acquires/visualizes 256 samples per channel. The samples are continuously acquired at a HW-timed rate of 2,560 Hz. The 2D DBL array data type contains the sample values.

(BNC-2120) and **(Simulated Hardware)** Modify this VI to acquire 3 channels of voltage (Sine Wave Function, TTL Square Wave Function and Temperature Reference) from PCI-6221/ai0, PCI-6221/ai2:3 at a sample rate of 2,560 Hz and number of samples of 256.

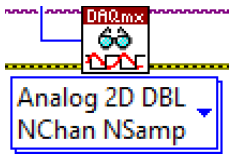
1. From the **Project Explorer** window, open N-Channel N-Samples (2D DBL) VI.
2. Examine the DAQmx Create Channel VI.
   - ☐ Switch to the block diagram, click the polymorphic VI selector of the DAQmx Create Channel VI and explore the selected instance.



   Notice that this VI 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 VI will wait for and output in each loop iteration.
   - ☐ If the sample rate is 2,560 Hz and number of samples is 256, then during each While Loop iteration, the DAQmx Read VI will acquire from multiple channels at a sample rate of 2,560 Hz and output 256 samples per channel.

4. Examine the DAQmx Read VI.

   ☐ Click the polymorphic VI selector of the DAQmx Read VI and explore the selected instance.



   Notice that this VI is set to **Analog»Multiple Channels»Multiple Samples» 2D DBL**.

5. Create the front panel.

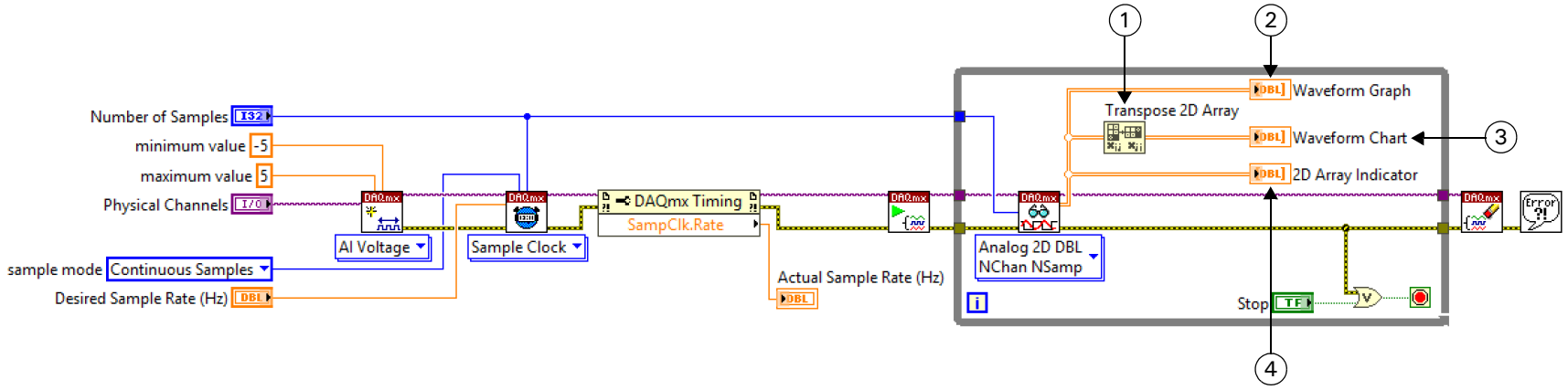
   ☐ Switch to the front panel, add the following indicators.

   – **Waveform Graph** indicator

   – **Waveform Chart** indicator

   – **2D Array** indicator

6. Modify the block diagram, as shown in the following figure, to visualize the multiple channels multiple samples (2D DBL) data.



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 256 samples per channel.
2 **Waveform Graph indicator**—Graph that displays the current array data in a specific iteration.
3 **Waveform Chart indicator**—Chart that can be used to show historical data and append new data..
4 **2D Numeric Array indicator**—This indicator displays only 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 "3 channel, 256 samples per channel" acquisition, the 2D array has 3 rows and 256 columns.

7. Configure the waveform graph properties.

   ☐ Switch to the front panel.

   ☐ Resize the plot legend to show 3 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 waveform chart properties.

   ☐ Resize the plot legend to show 3 plots.

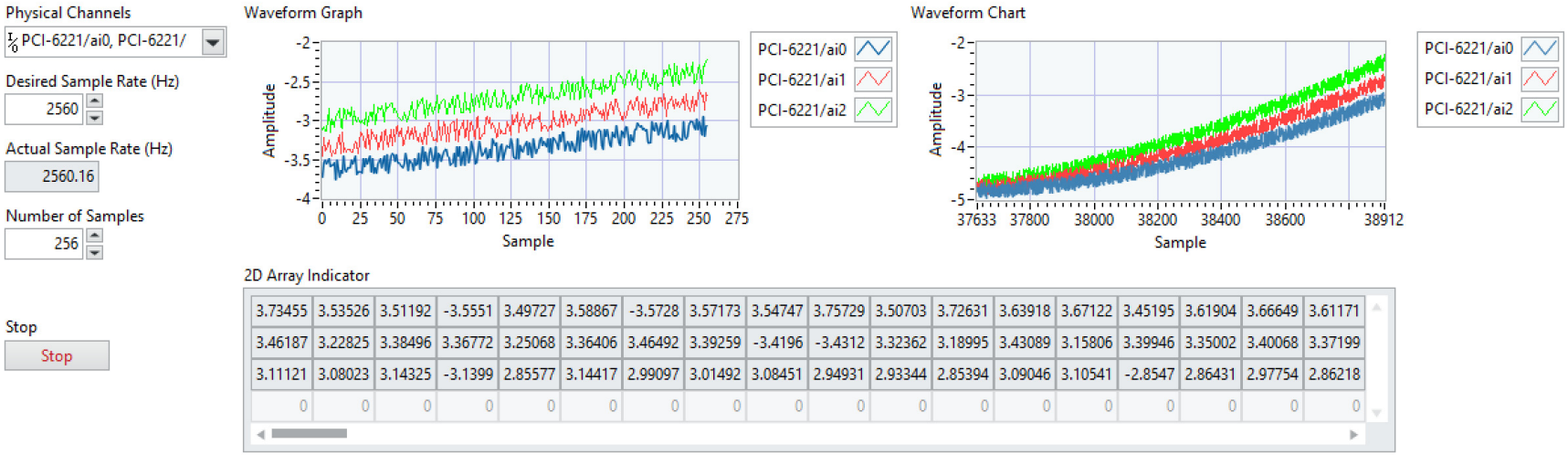
   ☐ Double-click **Time** on the x-axis and rename it as `Sample`. The 2D DBL data type does not contain any timing information, so the x-axis just represents the sample number.

☐ Double-click the minimum and maximum numbers on the x-axis and change them to `0` and `1280`. Now the chart will show the most recent 1,280 samples per channel acquired.

☐ Right-click the waveform chart then select **Chart History Length** and change the history length from 1,024 to 1,280. This setting determines the maximum number of points per plot that the chart can show.

Note Because this VI acquires 256 samples per channel per second, changing the history length to 1,280 configures the chart to show 1,280 samples per channel, which is equivalent to most recent 5 seconds of data.

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



10. Test the VI.

☐ On the front panel, set the **Physical Channels** control to **PCI-6221/ai0, PCI-6221/ai2:3**. This creates a DAQmx task that includes PCI-6221 analog input channels 0, 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 3 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).
  - 2D numeric array indicator shows the data only for the current iteration.
- ☐ Notice that the x-axis of the waveform chart indicates the sample number of a data point (not seconds). This is because the 2D DBL data type does not includes timing information.
- ☐ Stop the VI when finished.

End of Exercise 8-4

Exercise 8-5: Extracting a Subset of an *N*-Channel Data Array

Goal

Extract a single channel of data using the Index Array function.
Extract multiple channels of data using the Array Subset function.

Scenario

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

1D Waveform Array Index	Signal Description
0	Temperature Reference / Simulated Input 1
1	Sine Wave Function / Simulated Input 2
2	TTL Square Wave Function / Simulated Input 3

Hardware Setup

(BNC-2120) Make sure that you have Sine/Triangle BNC Connector connected to the Analog Input 2 and the TTL Square Wave BNC Connector connected to the Analog input 3, also make sure that the Sine/Triangle Waveform Switch is set to Sine.

Explore acquisition data in an existing VI

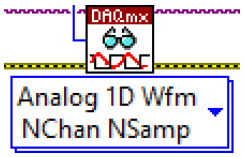
1. Open `<Exercises>\LabVIEW Core 1\Extract Data from N-Channel Array\Extract Data from N-Channel Array.lvproj`.

2. From the **Project Explorer** window, open the Extract Data from N-Channel Array (1D Waveform) VI.

3. Explore the front panel and the block diagram.

   The block diagram code performs a continuous acquisition.

   By clicking polymorphic VI selector of the DAQmx Read VI and exploring the selected instance you can determine that this VI reads *N* samples from *N* channels and returns the data as a 1D waveform array.



4. Examine the behavior of the VI.

   ☐ On the front panel, set the **Physical Channels** control to **PCI-6221/ai0, PCI-6221/ai2:3**.

   This creates a DAQmx task that includes PCI-6221 analog input channels 0, 2, and 3.

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

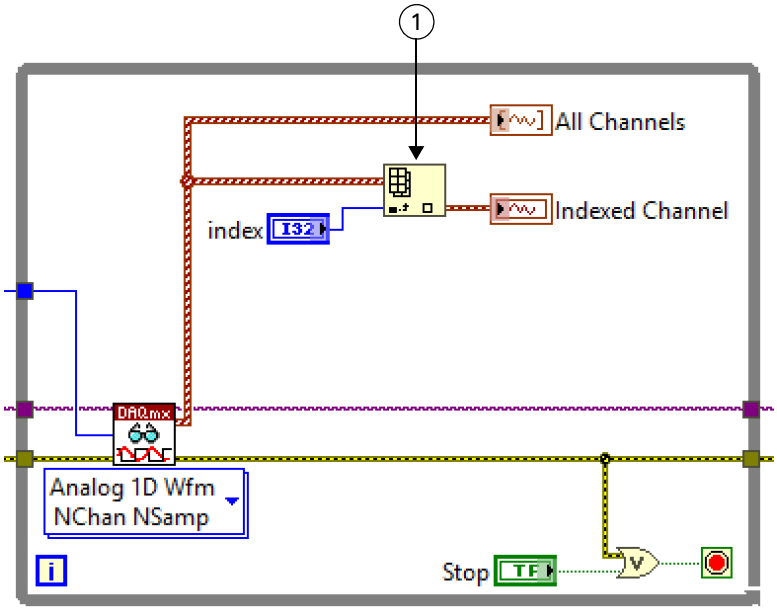
- ☐ Set **Number of Samples** to `256`.

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 Channels** control:
     - AI0 channel
     - AI2 channel
     - AI3 channel
6. In this exercise we are going to separate these three channels into different data sets.

Extract Data for a Single Channel

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



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

2. Use the context help and *LabVIEW Help* to learn about the Index Array function.

3. Test the VI.

   ☐ On the front panel, set the following values.

Physical Channel	`PCI-6221/ai0, PCI-6221/ai2:3`
Rate	`2560`
Number of Samples	`256`

   ☐ Run the VI.

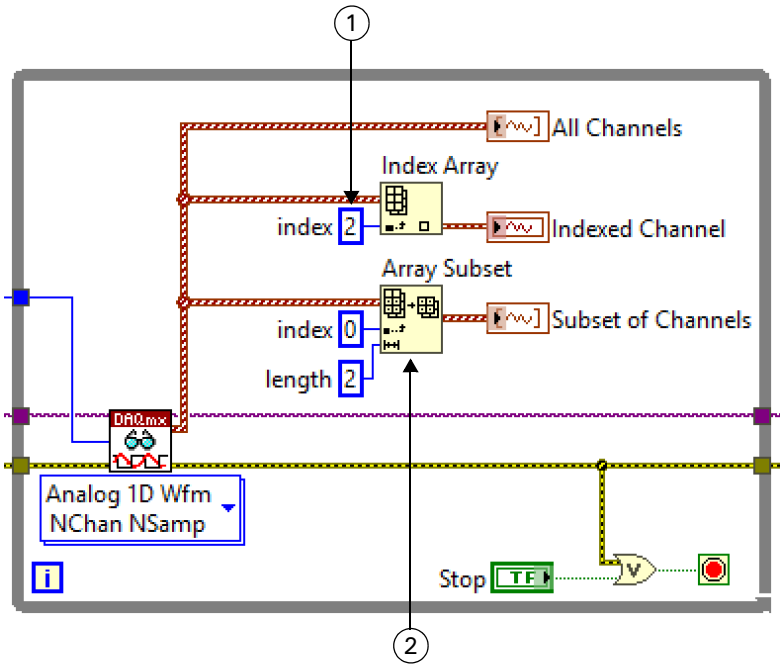
   ☐ Change the value of the index control to `0`, `1`, and `2`. Notice that this control determines which channel is extracted and displayed in the **Indexed Channel** indicator.

   ☐ Stop the VI when finished.

4. Save the VI.

Extract a Subset of Channels

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



1 **Index Array** function—Use this function to extract the third channel (index 2). Right-click the **Index** control, select **Change to Constant**. Set the constant value to `2`.

2 **Array Subset** function—Use this function to extract the first and second channels (indices 0 and 1).

   a. Right-click the **index** input and select **Create Constant.** Set the constant value to `0`.

   b. Right-click the **length** input and select **Create Constant**. Set the constant value to `2`.

   c. Right-click the **subarray** output and select **Create Indicator**. Rename the indicator as `Subset of Channels`.

2. Use the context help and detailed *LabVIEW Help* to learn about the Array Subset function.

3. Test the VI.

   ☐ On the front panel, set the following values.

Physical Channel	`PCI-6221/ai0, PCI-6221/ai2:3`
Rate	`2560`
Number of Samples	`256`

   ☐ Run the VI.

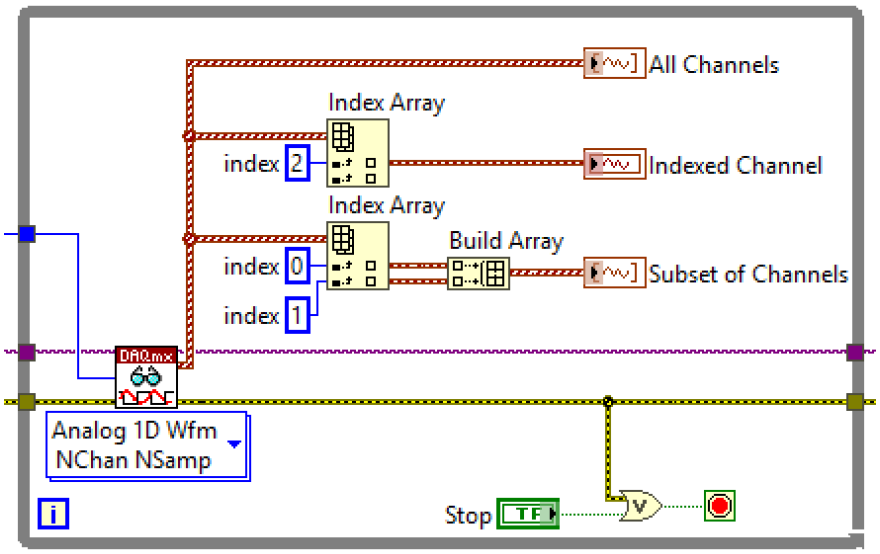
   ☐ Notice that the **Indexed Channel** indicator displays the AI3 channel.

- ☐ Notice that the **Subset of Channels** indicator displays the AI0 and AI2 channels.
- ☐ Stop the VI when finished.

4. Save the VI.

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

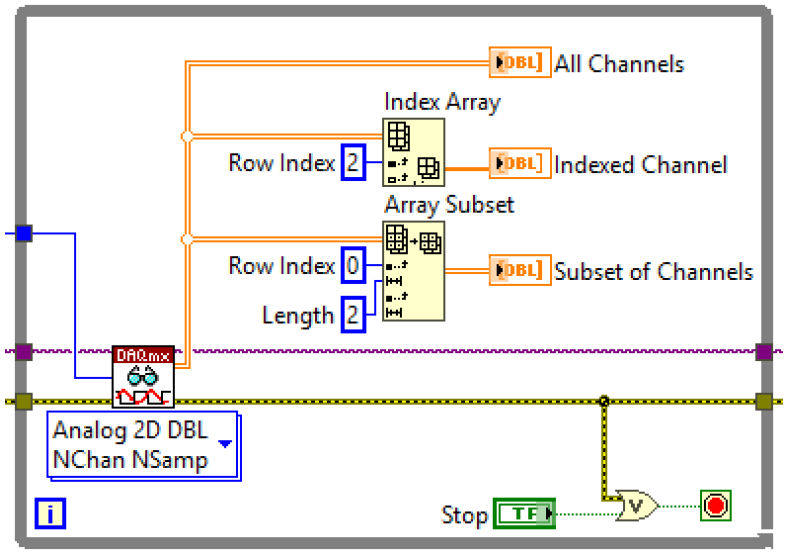
1. From the **Project Explorer** window, open the [Alternate Method] Extract Data from N-Channel Array (1D Waveform) VI.
   - ☐ Notice how this VI uses the Index Array function to extract multiple channels and then group them together into a new array using the Build Array function.



2. Add a new Index Array function and a Build Array function.
   - ☐ Notice how you can resize the Index Array function to extract multiple array elements. Notice how you can resize the Build Array function to take multiple inputs. Use the context help and detailed *LabVIEW Help* to learn about these two functions.

Explore How to Extract Channels From 2D DBL Array Data Type

1. From the **Project Explorer** window, open the Extract Data from N-Channel Array (2D DBL) VI.



2. Examine the behavior of the VI.

   ☐ On the front panel, set the following values.

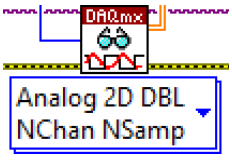
Physical Channel	`PCI-6221/ai0, PCI-6221/ai2:3`
Rate	`2560`
Number of Samples	`256`

   ☐ Run the VI.

   ☐ Stop the VI when finished.

3. On the block diagram, Click polymorphic VI selector of the DAQmx Read VI and explore the selected instance.

   ☐ Notice that the DAQmx Read VI is using the **floating point** data type. Notice that the output data type is 2D DBL array.



4. Notice that the Index Array function 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 function causes the Index Array function 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 disabled 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 disabled 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 disabled index (col) input.

  In this VI, the 2D array wired to the Index Array function 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 function to extract 2 rows of data starting from row index 0.

Your Turn

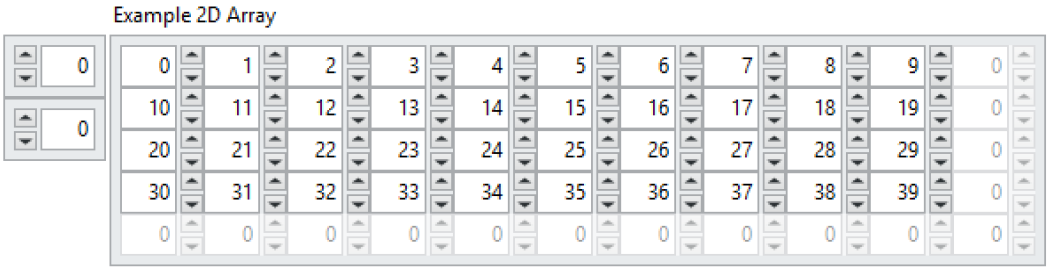
Option 1

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. **(Simulated Hardware)** Try acquiring 4 channels (10 samples per channel) of data. Extract an individual channel or groups of channels.

Option 2: Experiment with Index Array and Array Subset functions

1. From the **Project Explorer** window, open [Unguided] Example 2D Array VI.
2. Familiarize yourself with the functionality of the Index Array and Array Subset functions by wiring the 2D array to these functions, experimenting with how the input terminal values of these functions affect their output values, and exploring the *LabVIEW Help*.

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



End of Exercise 8-5

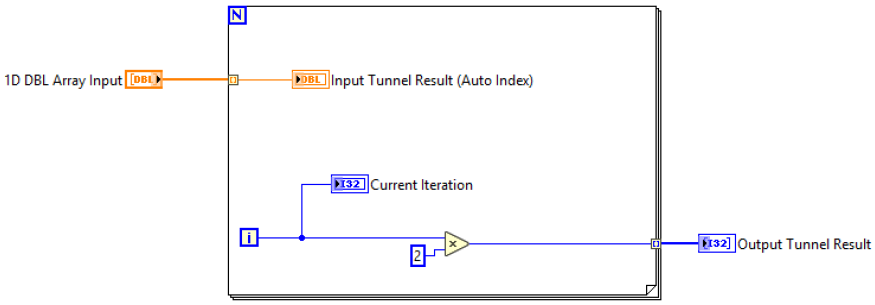
Exercise 8-6: Exploring Auto-Indexing Tunnels

Goal

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

Instructions

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



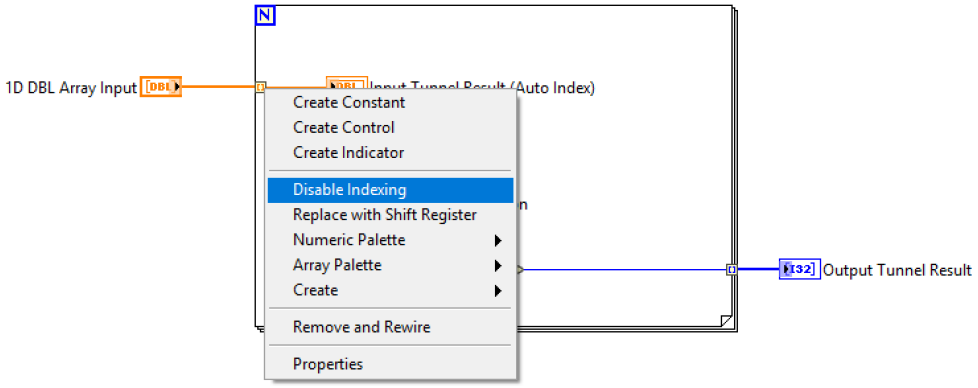
☐ Wire the **1D DBL Array Input** control to the left border of the For Loop, which by default creates an auto-indexing input tunnel. Then, wire the input tunnel to the **Input Tunnel Result** indicator.

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 indexing. When you wire an array from an external node to an input tunnel on the loop border and enable indexing on the input tunnel, elements of that array enter the loop one at a time, starting with the first element.

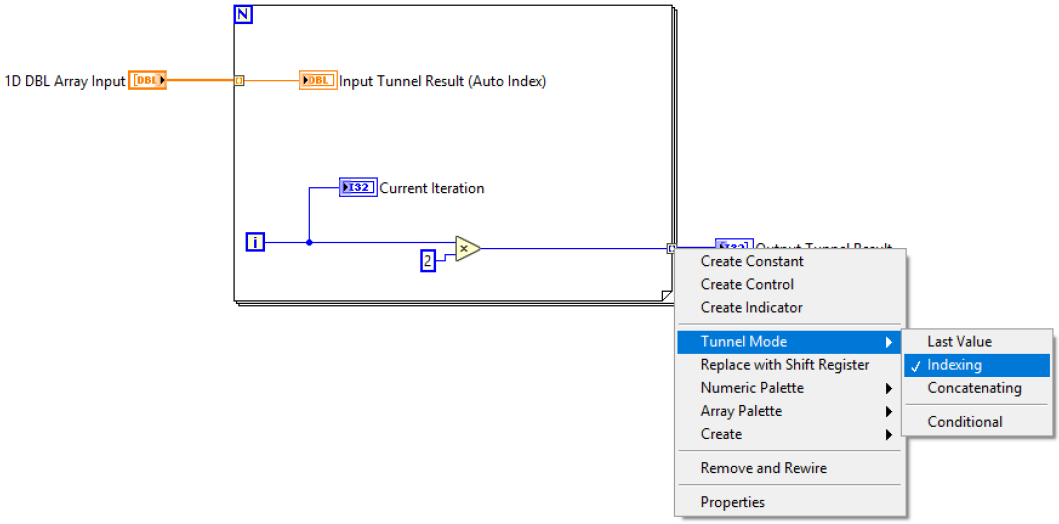
☐ Wire the output of the Multiply function to the right border of the For Loop, which by default creates an indexing output tunnel. Then, wire the output tunnel to the **Output Tunnel Result** indicator.

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

4. Right-click the indexing input tunnel. In the appeared menu you can set the behavior of any loop input tunnel to enable or disable indexing.



5. Right-click the indexing output tunnel. In the **Tunnel Mode** section, notice that **Indexing** is selected. This is where you can set the behavior of any loop output tunnel to **Last Value**, **Indexing**, or **Concatenating**.



6. Turn on Highlight Execution.

7. Run the VI.

   ☐ On the front panel, observe the behavior of each control/indicator.

   ☐ On the block 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 indexing output tunnel while the For Loop executed.

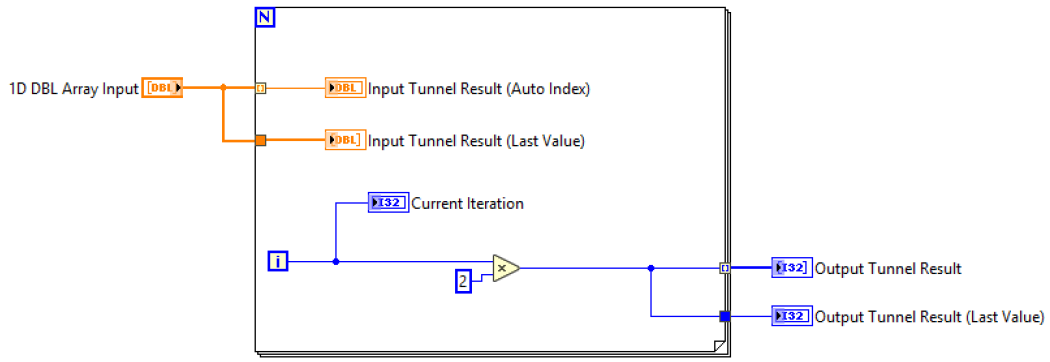
   ☐ Notice how the For Loop output tunnel does not return anything until the entire For Loop has finished executing.

- ☐ Run the VI multiple times with Highlight Execution turned on to make all the above observations.
- ☐ Stop the VI when finished.

8. Save the VI.

Comparing the Behaviour of Auto-Indexing vs. Keeping Last Value

1. Open Compare Auto-Index and Last Value VI.
2. Explore the front panel and the block diagram.
   - ☐ Notice that for each For Loop input tunnel and output tunnel, there is an indexing indicator and a last value indicator.
   - ☐ On the block diagram, select each tunnel. Notice 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.
   - ☐ Stop the VI when finished.
4. Save the VI.

End of Exercise 8-6

Exercise 8-7: Processing Data For Each Channel in an *N*-Channel Data Array

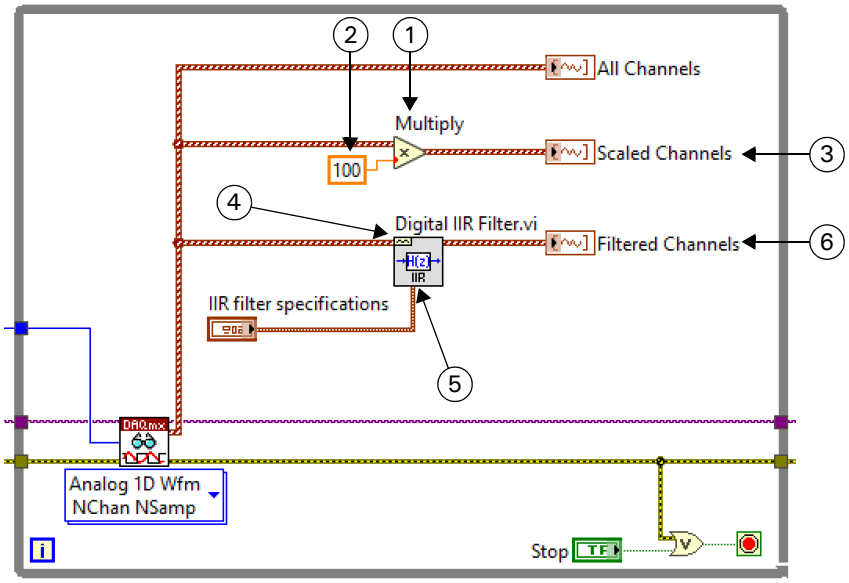
Goal

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

Process Channels Using VIs Compatible with *N*-Channel Array Data Types

1. Open `<Exercises>\LabVIEW Core 1\Process N-Channel Array\Process N-Channel Array.lvproj`.
2. From the **Project Explorer** window, open Process N-Channel Array VI.
3. Examine the VI.
   - ☐ Explore the front panel and the block diagram.
   - ☐ Notice that the **data** output of the DAQmx Read VI is a 1D waveform array data type.
   - ☐ Run the VI.
   - ☐ Stop the VI when finished.

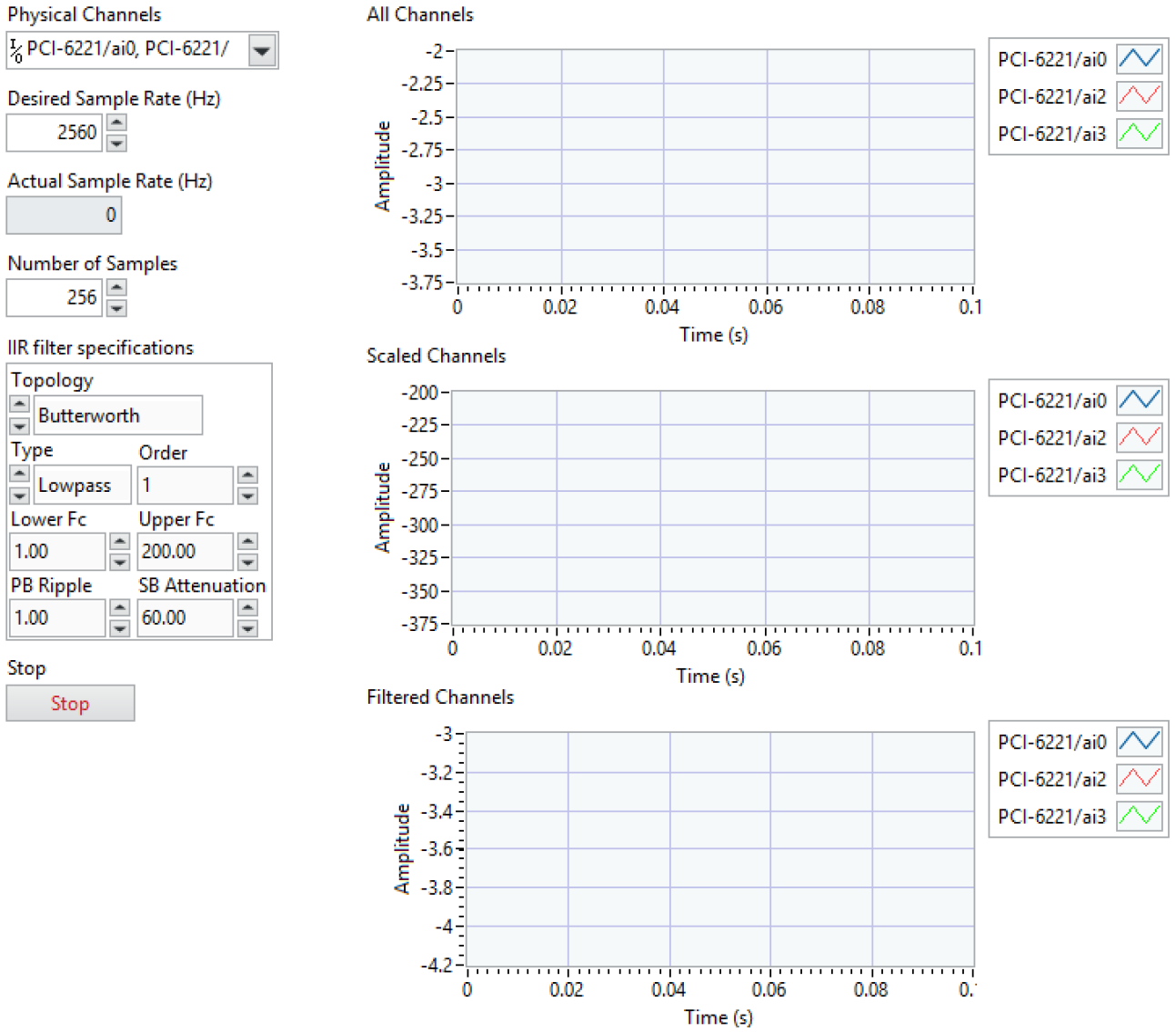
4. Modify the block diagram, as shown in the figure below, to process all channels by scaling their values by 100 and filtering the signals using the following items.



1 **Multiply** function—The input of this function can adapt to accept the 1D waveform array data type.
2 Right-click in the block diagram and search for the DBL Numeric Constant, then wire it to the second input of the Multiply function. Set the value of the constant to `100`. This function will multiply all Y values in every waveform element in the array by 100.
3 Right-click the output of the Multiply function and select **Create Indicator**. Rename the indicator `Scaled Channels`.
4 **Digital IIR Filter VI**—Right-click in the block diagram, click **Search** from the Functions palette and type `Digital IIR Filter`. Drag the first appeared result into the while loop. Wire the 1D waveform array from the DAQmx Read VI to the input of the Digital IIR Filter VI. Notice that the Digital IIR Filter VI will process each waveform contained in the array.
5 Right-click the **IIR filter specifications** input and select **Create Control**.
6 Right-click the **signal out** output of the Digital IIR Filter VI and select **Create Indicator**. Rename the indicator as `Filtered Channels`.

Note Many Math VIs and functions can adapt to accept the 1D waveform array data type. A real-world application can use Math VIs and functions to scale acquired data from units of voltage to the appropriate engineering unit, such as temperature (deg Celsius), acceleration (g), etc.

5. Arrange and modify your front panel similar to the following figure. Make sure that the **IIR filter specifications** cluster control is set as shown in the figure.



6. Examine the behavior of the VI.

   ☐ On the front panel set the **Physical Channels** control to **PCI-6221/ai0, PCI-6221/ai2:3.**

   – **(BNC-2120)** Make sure that you have Sine/Triangle BNC Connector connected to the Analog Input 2 and the TTL Square Wave BNC Connector connected to the Analog input 3, also make sure that the Sine/Triangle Waveform Switch is set to Sine.

   – **(BNC-2120 and Simulated Hardware)** Set the value of **Lower Fc** control to `1`.

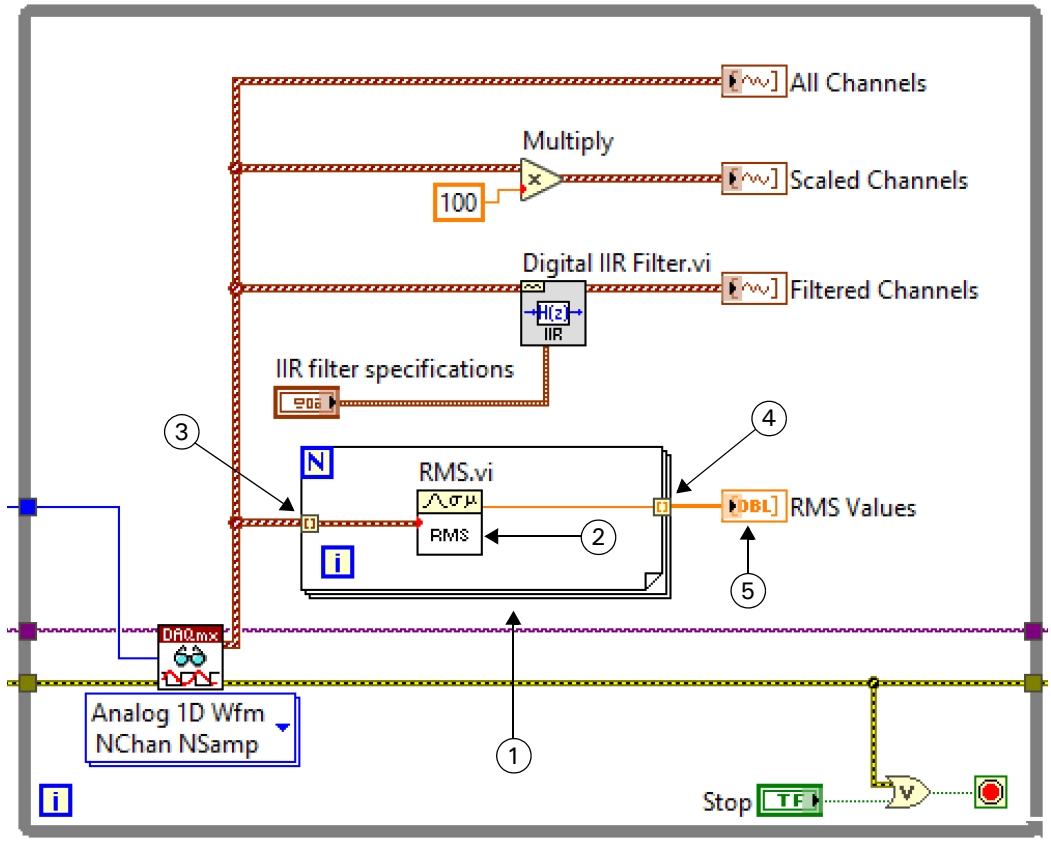
   ☐ Run the VI.

   ☐ Notice that the **Scaled Channels** and **Filtered Channels** waveform graph indicators show that the Multiply function and Digital IIR Filter VI processed all channels in the 1D waveform array.

   ☐ Stop the VI when finished.

Process Channels Using a VI that is not Fully Compatible with *N*-Channel Array Data Types

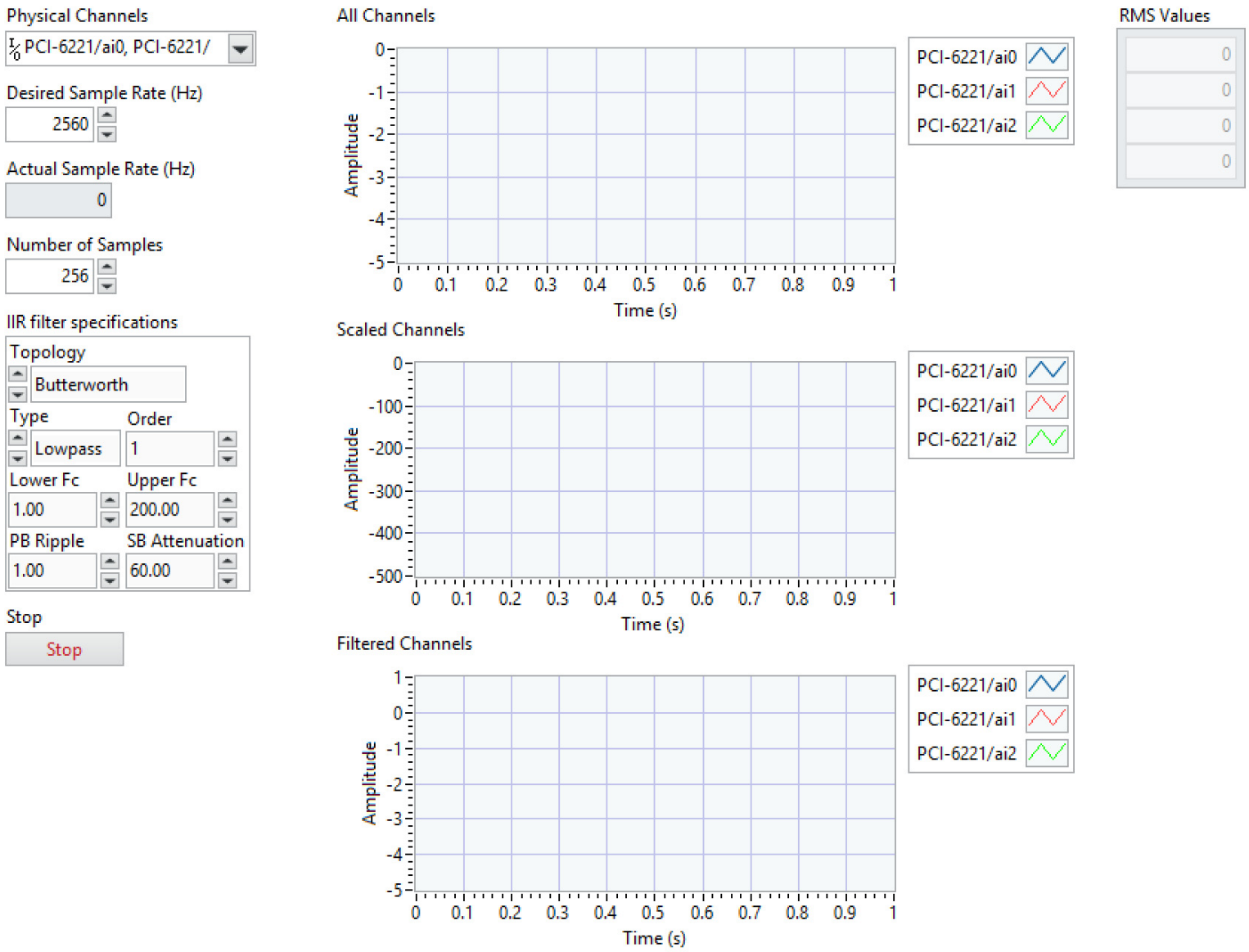
1. Modify the block diagram, as shown in the following figure, to run all channels through a VI that is not fully compatible with the 1D waveform data type.



1 **For Loop**—Add a For Loop to the block diagram.
2 **RMS VI**—Right-click in the block diagram, click **Search** from the Functions palette, type `RMS.vi`. Drag the first appeared result into the while loop. The input of this VI isn't fully compatible with 1D Waveform data type, because if you wire the waveform directly to the input of this VI, the output will show the RMS value for only one channel, instead of four used in this exercise.
3 Wire the **data** output of DAQmx Read VI to the left border of the For Loop and into the RMS VI input. Notice the indexing input tunnel. On the first iteration of the For Loop, this indexing input tunnel will return 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 VI to the right border of the For Loop. This creates an indexing output tunnel. When the For Loop finishes executing, the output of this indexing output tunnel will be a 1D array with its elements containing what the indexing output tunnel received in each For Loop iteration.
5 Right-click the indexing output tunnel and select **Create Indicator**.

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

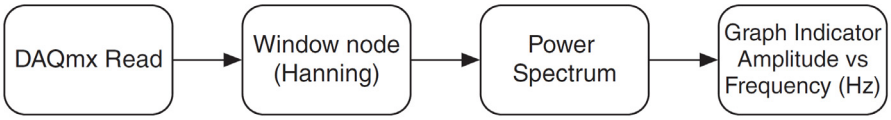
2. Arrange your front panel similar to the following figure.



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, which means 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.
2. Modify the VI to perform the above algorithm on all channels.

Answer

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

End of Exercise 8-7

9 Executing Code Based on a Condition

In this lesson, you learn how to execute code based on a condition by using a Case Structure.

Exercises

Exercise 9-1 Executing Code Based on a Condition

Exercise 9-1: Executing Code Based on a Condition

Goal

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

Hardware Setup/Scenario

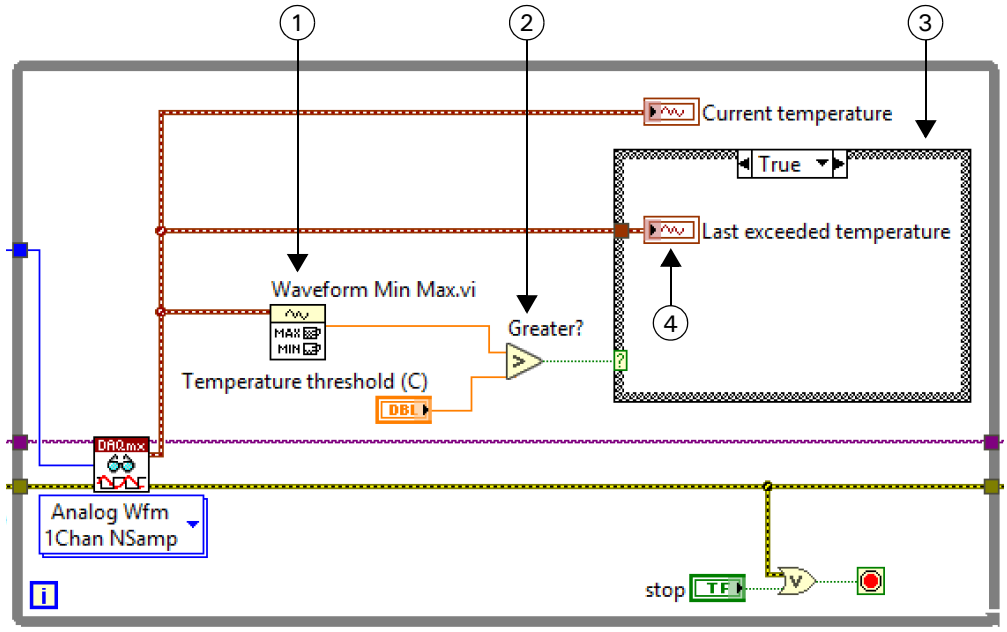
BNC 2120 option

☐ Make sure, that the thermocouple is connected to the Thermocouple Input Connector (AI1) properly.

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

1. Open `<Exercises>\LabVIEW Core 1\Execute Conditional Code\Execute Conditional Code.lvproj`.
2. From the **Project Explorer** window, open the Condition Based on Measurement Result VI.
3. Explore the VI.
   - ☐ Explore the front panel and the block diagram.
   - ☐ Set the DAQmx Physical Channel input to **PCI-6221/ai1**.
   - ☐ Run the VI.
     - Data will appear on the **Current Temperature** graph.
   - ☐ Stop the VI when finished.

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



1 **Waveform Min Max** VI—The Y max output returns the maximum value of the Y array in the waveform.
2 **Greater?** function—Right-click the y input of the Greater? function and select **Create Control**. Rename it `Temperature threshold (C)`.
3 **Case structure**—This VI will use the Case structure to update the **Last exceeded temperature** waveform graph indicator only when the acquired temperature exceeds a threshold.
4 **Last exceeded temperature** indicator—Add a waveform graph indicator to the front panel and wire the corresponding terminal as shown.

5. Configure front panel items.

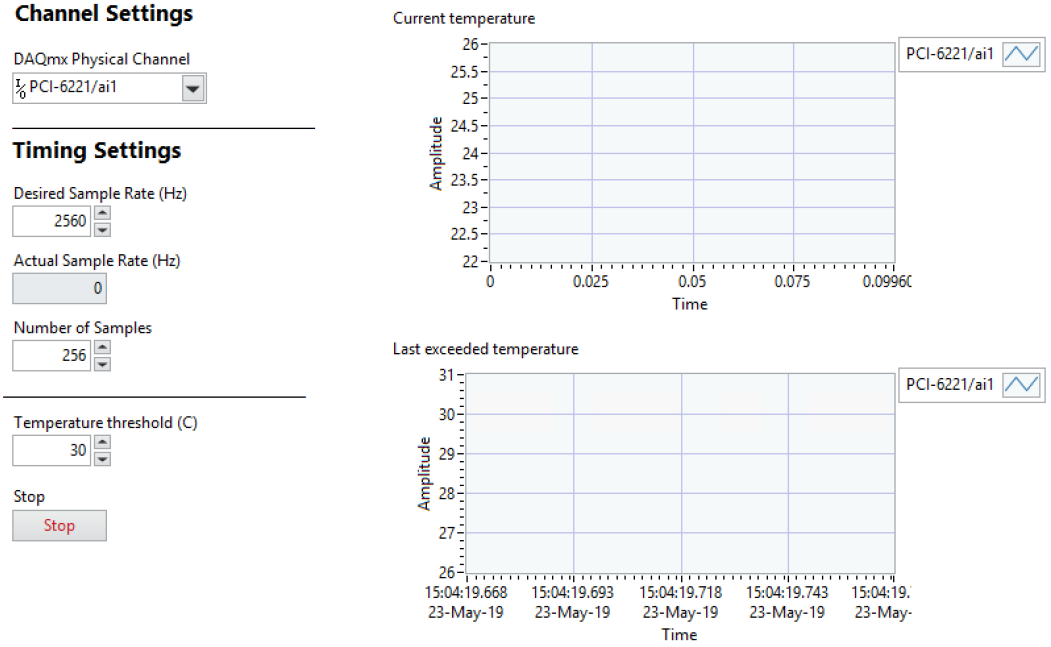
   ☐ Configure the **Last exceeded temperature** waveform graph to show the absolute time, so you can see the exact date and time of when the acquired data last exceeded the threshold.

   – Right-click on the waveform graph, then select **X Scale»Formatting**.

   – In the opened window, make sure that the editing mode is set to **Default** and select **Absolute time** in the **Type** section.

   ☐ Wire this indicator as shown in figure above.

6. Arrange the front panel, as shown in the following figure.



7. Set the control values, as shown in the figure above. Otherwise set the values to whatever is appropriate for your hardware setup.

8. **(BNC 2120)** Lower the Temperature threshold to 30.

   You should be able to exceed the temperature threshold in the next step by firmly pressing the thermocouple.

9. Examine the behavior of the VI.

   - ☐ Run the VI.
   - ☐ Adjust the value of the **Temperature threshold (C)** control and/or the acquired signal, so that **Current temperature** exceeds **Temperature threshold** from time to time.
   - ☐ Notice that every time the **Current temperature** graph contains a value above the **Temperature threshold (C)** control value, that data is passed to the **Last exceeded temperature** graph.
   - ☐ Notice that the x-axis of the **Last exceeded temperature** graph shows the absolute date and time of the last temperature waveform that exceeded the threshold.
   - ☐ Stop the VI when finished.

10. Select **File»Save All** to save the VI.

11. Close the VI.

End of Exercise 9-1

Exercise 9-2: Execute Code Conditionally Based on a User Setting/Parameter/Configuration

Goal

Explore a DAQmx example that sets the triggering behavior based on a user-input.

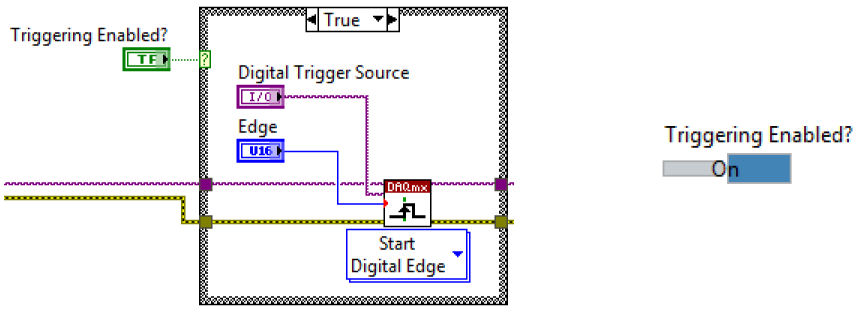
Hardware Setup/Scenario

BNC 2120 option

☐ Make sure, that the **Quadrature Encoder's UP/DN** terminal is connected to the **PFI1** input.

1. From the **Project Explorer** window, open the Condition based on User Input (Triggering) VI.
2. Explore the block diagram.

☐Notice that the Case structure executes code based on the value of the **Triggering Enabled?** control.



3. Test the VI.

☐ Set the following control values.

Physical Channel	PCI-6221/ai1
Digital Trigger Source	PCI-6221/PFI1
Edge	Rising
Triggering Enabled?	On

☐ Run the VI.

The VI is now waiting on the PCI-6221/PFI1 line to detect a rising edge from a digital signal.

☐ **(BNC 2120)** Rotate the Quadrature Encoder clockwise. The VI should immediately return data on the Graph indicator.

If you set the **Triggering Enabled?** control to Off on the front panel and run the VI again, you will see that VI immediately returns data and no longer waits for a trigger.

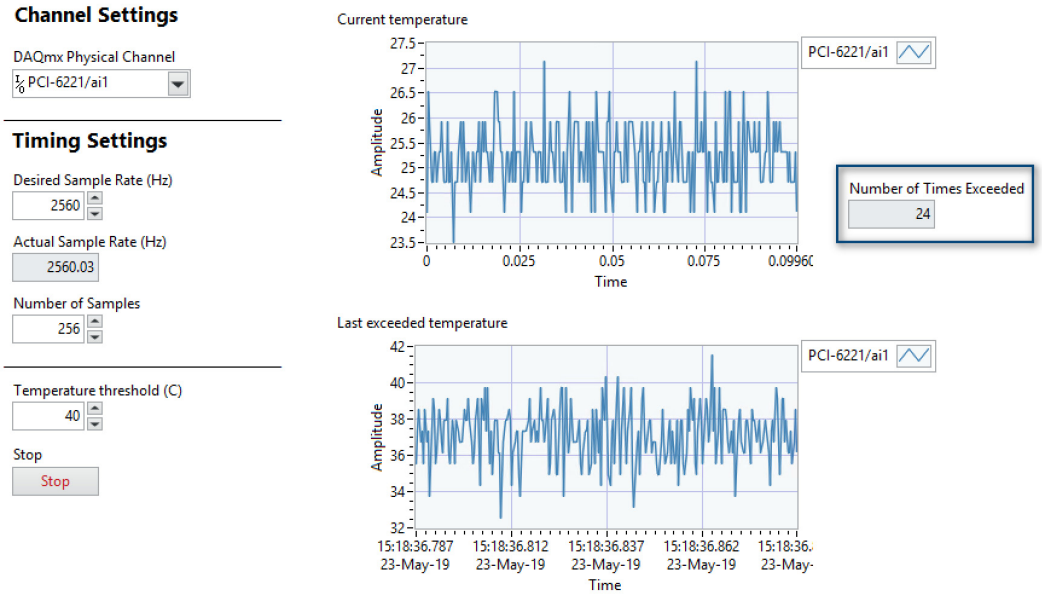
☐ **(Simulated hardware)** Because you are using a simulated DAQ device, the PCI-6221/PFI1 line randomly changes between True and False. When this line goes from False to True, the VI detects this as a rising edge, which triggers the VI to acquire data.

☐ On the block diagram, use execution highlighting 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 the Condition Based on Measurement Result VI 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>\Exercise 9-1\[Challenge] Condition Based on Measurement Result with Running Total.vi`.

On the Job

1. In your own applications, do you need to execute code conditionally based on a measurement result? If so, describe it 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 it below.

End of Exercise 9-2

10 Writing and Reading Data to File

In this lesson, you explore how to write data to file and read data from file.

Exercises

Exercise 10-1: Using High-Level I/O VIs/functions

Goal

Use high-level file I/O VIs and functions to write an array of acquisition data to a file.

Hardware Setup

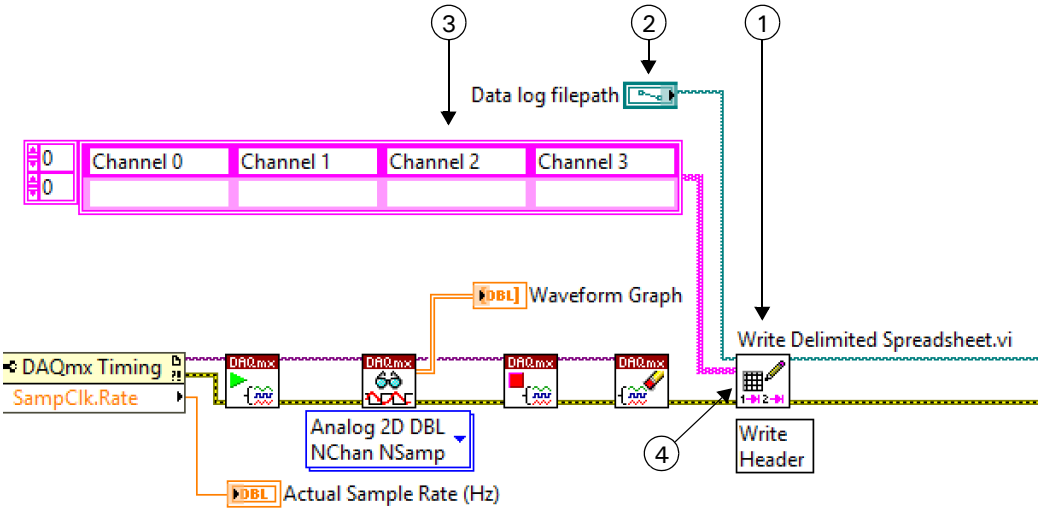
- BNC-2120—Set up the Temperature Reference, Thermocouple, and Sine Wave Function & TTL Square Wave Function inputs (ai0:3).
- Simulated—Set up ai0:3 of an analog input module.

Instructions

Finite Acquisition of *N* Channels and *N* Samples

1. Open the following project: `<Exercises>\LabVIEW Core 1\High-Level Write to File\High-Level Write to Text File.lvproj`.
2. From the **Project Explorer** window, open the High-Level Write to Text File – N-Channel VI.
3. Examine the VI.
   - ☐ Notice that the DAQmx Read VI is configured to read multiple samples from multiple channels as a 2D DBL array.
   - ☐ Notice that the DAQmx Timing VI 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).

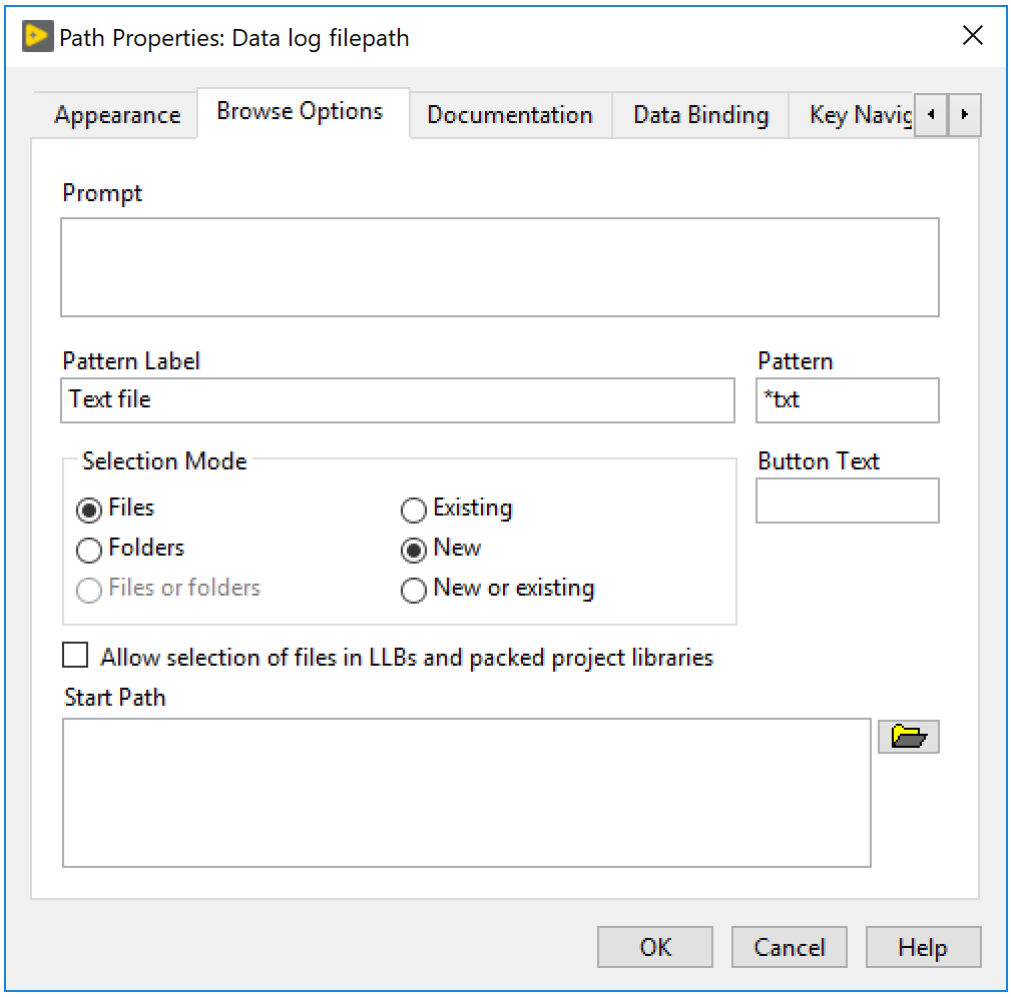
4. Modify the block diagram, as shown in the following figures, to write the acquisition data to a file.



1 **Write Delimited Spreadsheet VI**—Use the first Write Delimited Spreadsheet VI to write the header to the file. Place the VI on the block diagram.
2 Right-click the **file path** input and select **Create Control**. Rename the control `Data log filepath`.
3 **String Array Constant**—Place an Array Constant on the block diagram. Create a String Constant and drag it into the Array Constant, which changes the Array Constant to a string array constant. Right-click the array index, select **Properties» Size**, and 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 VI, 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 text file already exists at the data log file path, this VI will overwrite any data in that file.

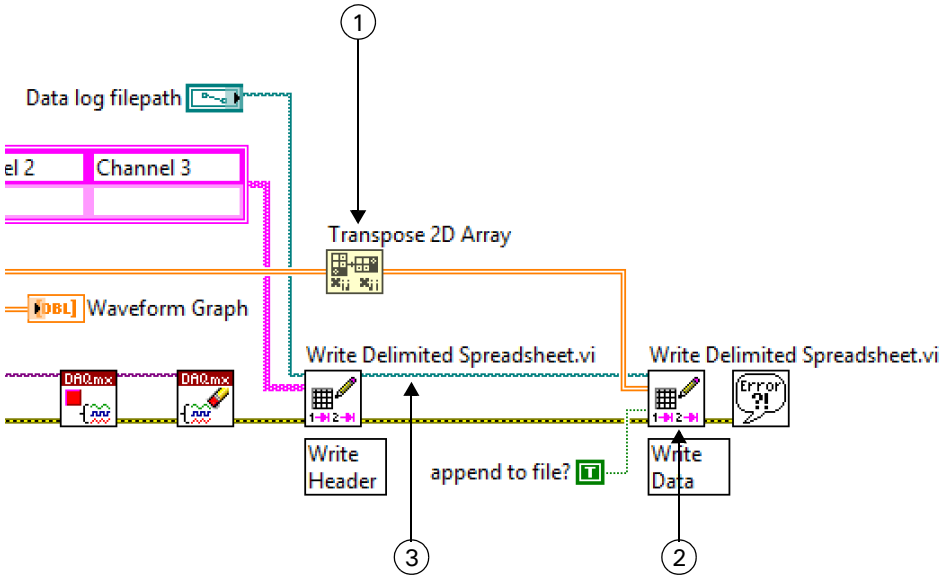
☐ Configure the **Data log filepath** control.

- Switch to the front panel.
- Right-click the **Data log filepath** control and select **Properties**.
- In the **Browse Options** tab set **Selection Mode** as shown in the figure below. This allows the user to browse to a new or existing file path where the user wants to save the log file.



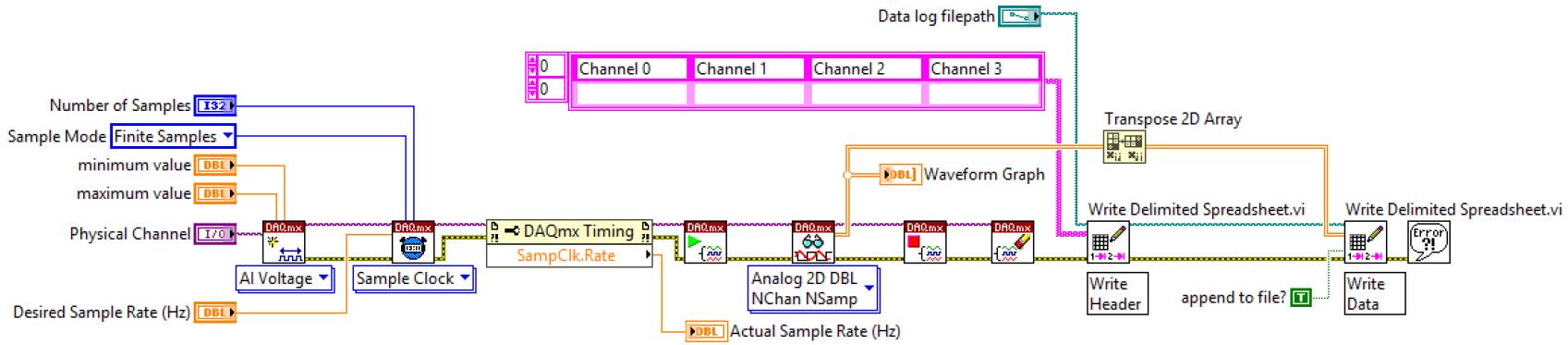
- Set **Pattern Label** to `Text file`.
- Set **Pattern** to `*txt`.

☐ 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 function**—Use this function 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 VI**—use the second Write Delimited Spreadsheet VI to write the numeric data to the file. The previous Write Delimited Spreadsheet VI adds headers to the text file, and this Write Delimited Spreadsheet VI appends data to the file after the headers.
3 Wire the file path terminals between the two Write Delimited Spreadsheet VIs, as shown in the figure. This tells the second VI to write to the same file as the first VI.

5. Complete the wiring as shown in the following figure.



6. Create the text file.

   ☐ On the front panel, set the **Physical Channel** control to the following value:

   (BNC-2120/Simulated Hardware) PCI-6221/ai0:3

   ☐ On the front panel, click the **...** button of the **Data log filepath** control. In the **Save As** dialog box, set the file path to `<Exercises>\LabVIEW Core 1\High-Level Write to File\Finite Acquisition Data.txt`.

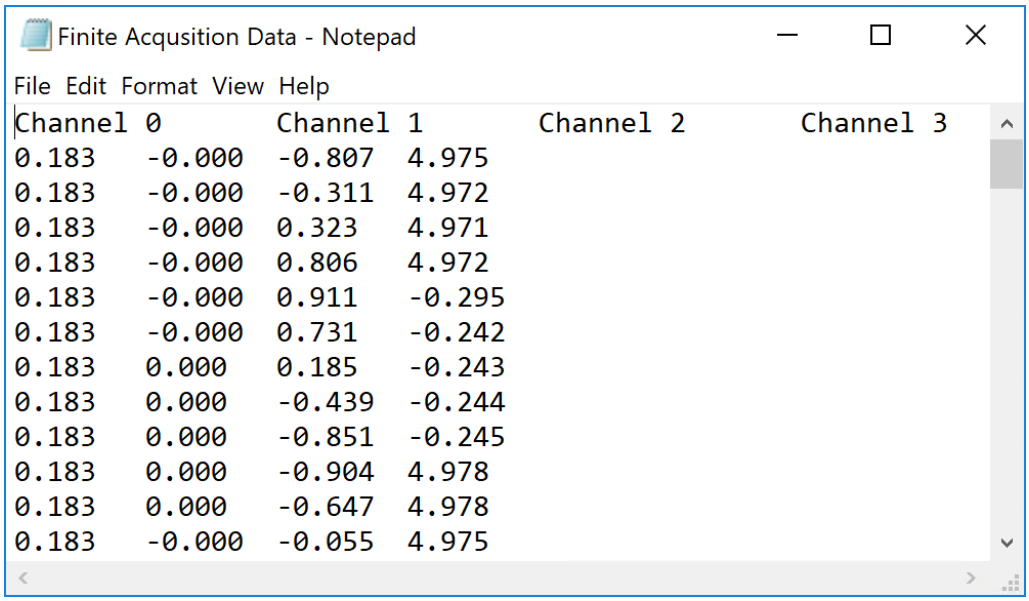
   ☐ Run the VI.

7. Explore the text file created by this VI.

   ☐ In **Windows Explorer**, navigate to the data log file and double-click it to view the contents.

   ☐ Notice the file includes a header describing each channel.

   ☐ Verify that the log file looks similar to the following figure.



   ☐ 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	0.183	0	-0.807	4.975
3	0.183	0	-0.311	4.972
4	0.183	0	0.323	4.971
5	0.183	0	0.806	4.972
6	0.183	0	0.911	-0.295
7	0.183	0	0.731	-0.242
8	0.183	0	0.185	-0.243
9	0.183	0	-0.439	-0.244
10	0.183	0	-0.851	-0.245
11	0.183	0	-0.904	4.978
12	0.183	0	-0.647	4.978
13	0.183	0	-0.055	4.975
14	0.183	0	0.555	4.972
15	0.183	0	0.886	4.972

Your Turn

1. Use the LabVIEW 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 function?

   ☐ What is the effect on the VI if you remove the filepath input of the first Write Delimited Spreadsheet VI?

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 VIs/functions to Stream Data to a Text File

Goal

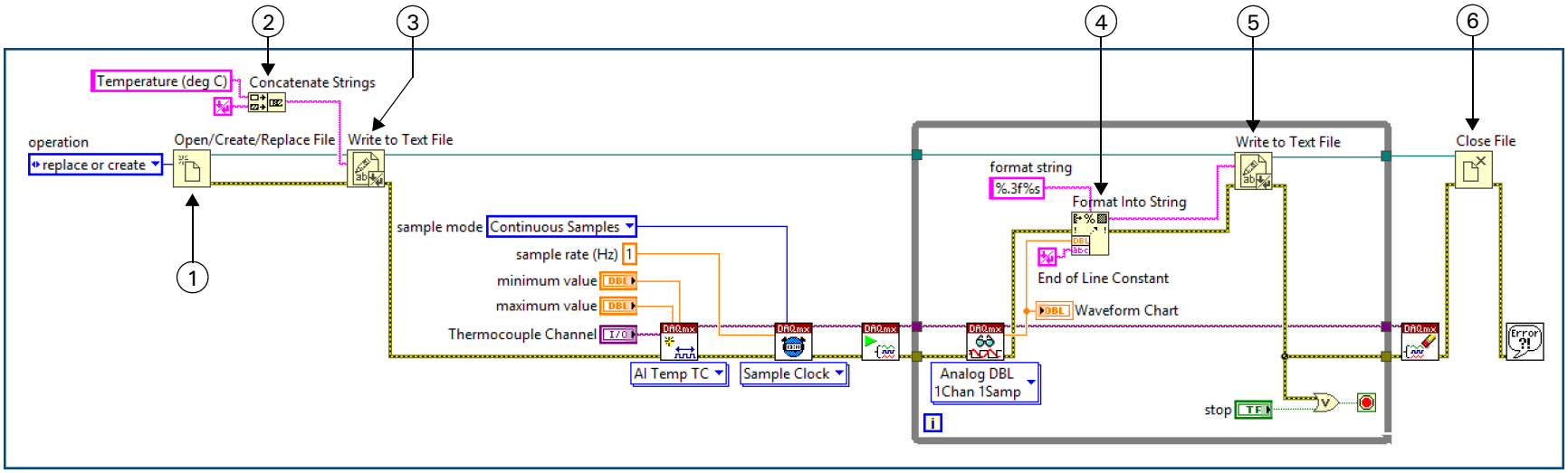
Stream single channel, single sample temperature data to text file.

Instructions

1. Open the following project: `<Exercises>\LabVIEW Core 1\Low-Level Stream to Text File (1Chan 1Samp)\Low-Level Stream to Text File (1Chan 1Samp).lvproj`.
2. From the **Project Explorer** window, open the Stream to Text File – 1 channel 1 sample VI.
3. Examine the VI.
   - ☐ Notice that the DAQmx Create Channel VI configures the task to acquire measurements from a thermocouple.
   - ☐ Notice that the DAQmx Timing VI configures the task to acquire measurements continuously at a sample rate of 1 Hz.
   - ☐ Notice that the DAQmx Read VI reads 1 sample from 1 channel as a single DBL data type.

   Question 1 - How frequently does the While Loop execute? _____ times per second.

4. Modify the block diagram, as shown in the following figure, to stream data to a text file.



1 **Open/Create/Replace File function**—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 function will launch a file dialog box for the user to choose the file path for the data log file.
2 **Concatenate Strings function**—Use this function to create the header string with an End of Line constant.
3 **Write to Text File function**—Use the first Write to Text File function to write the header to the first line of the file before the VI enters the While Loop.
4 **Format Into String function**—Use this function to convert the numeric DBL data into a string data type because the Write to Text File function requires a string input.
This function, as configured in this exercise (formating string syntax is **"%.3f%s"**), 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 function and the format string syntax.
5 **Write to Text File function**—Use the second Write to Text File function to continuously write data to file inside the While Loop.
6 **Close File fucntion**—Closes the file. You should always close a file when you are done accessing the file in the VI.

5. Save the VI.

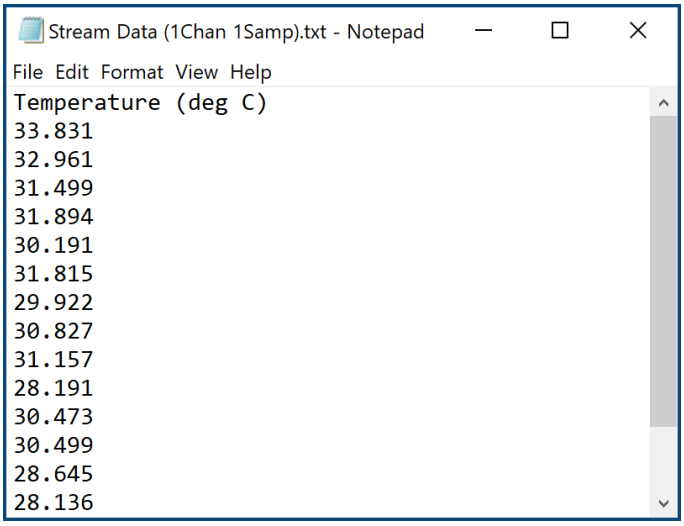
6. Run the VI.

   ☐ On the front panel, set the **Thermocouple Channel** control to a single thermocouple channel on your DAQ device (e.g. PCI-6221/ai1).

   ☐ Click the **Run** button.

- ☐ Save the file as `<Exercises>\LabVIEW Core 1\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 a log file with two column headers (“Current Temperature (deg C)”, “Current Temperature (deg F)”) and two columns of data.

Answers

Question 1 - **Answer:** The While Loop executes 1 time per second. The DAQmx Read VI 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 VI must wait until a sample is available to read.

End of Exercise 10-2

Exercise 10-3: Streaming *N*-Channel Acquisition Data to a Text File

Goal

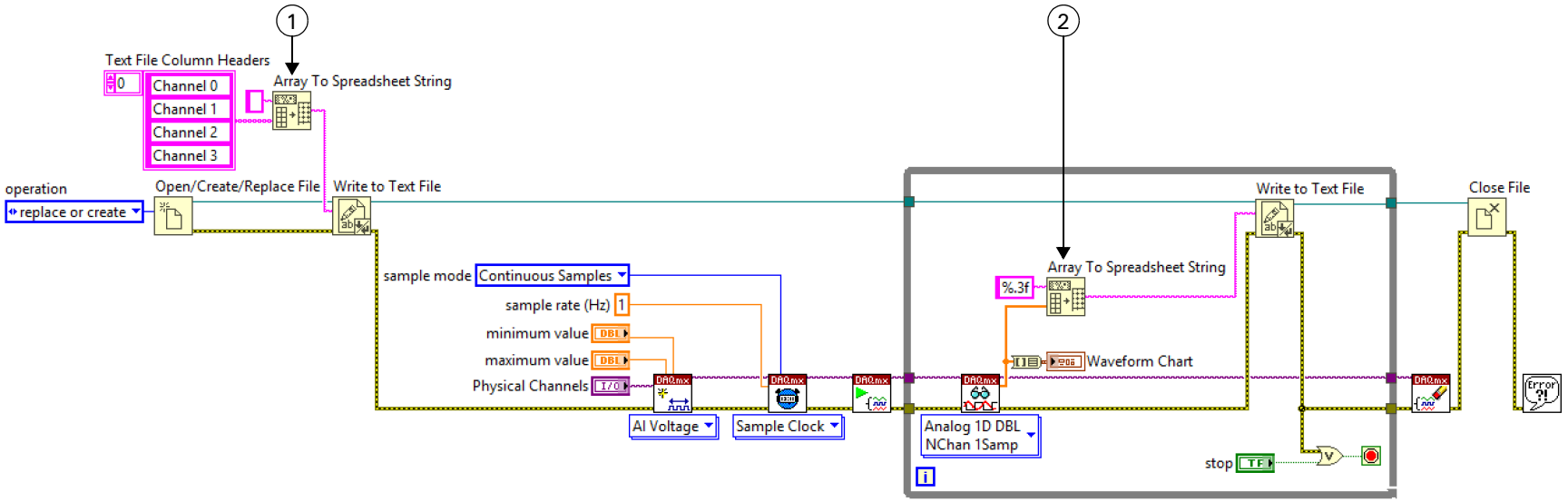
Explore examples that stream *N* channel 1 sample and *N* channel *N* sample acquisition data to a file.

Instructions

N Channels, 1 Sample (1D DBL) Streaming Example

1. Open the following project: `<Exercises>\LabVIEW Core 1\Low-Level Stream to Text File (NChan)\Low-Level Stream to Text File (NChan).lvproj`.
2. From the **Project Explorer** window, open the Low-Level Stream to Text File (NChan 1Samp) VI.

3. Explore the block diagram.

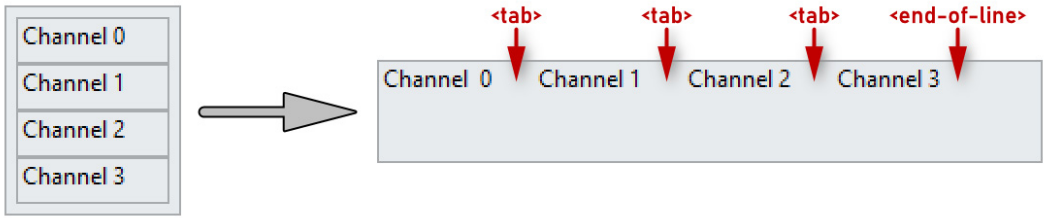


1 **Array to Spreadsheet String function**—This function is configured to create a spreadsheet string, which adds a delimiter between each array element and adds an end-of-line constant at the end of the string. In this VI, it turns the incoming string array into the following string:

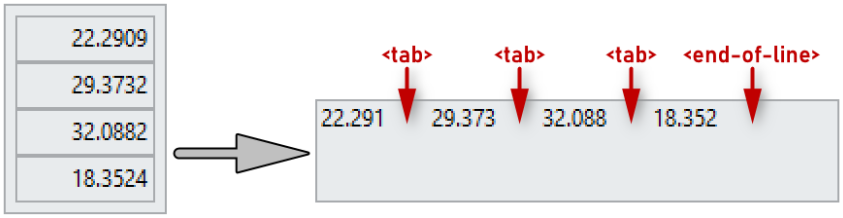
`Channel 0<tab> Channel 1<tab> Channel 2<tab> Channel 3<end-of-line>.`

2 **Array to Spreadsheet String function**—Use this function to convert the 1D DBL array data into a string data type because the Write to Text File function requires a string input.

The first Array to Spreadsheet String function will create the following result.



The second Array to Spreadsheet String function, as configured in this exercise, would format a 1D DBL input to a spreadsheet string, as shown in the following example.



4. Test the VI.

   ☐ Run the VI.

   ☐ In the file dialog box, save the log file as `<Exercises>\LabVIEW Core 1\Low-Level Stream to Text File (NChan)\Stream Data (NChan 1Samp).txt`.

   ☐ Click the **Stop** button after approximately 10 seconds.

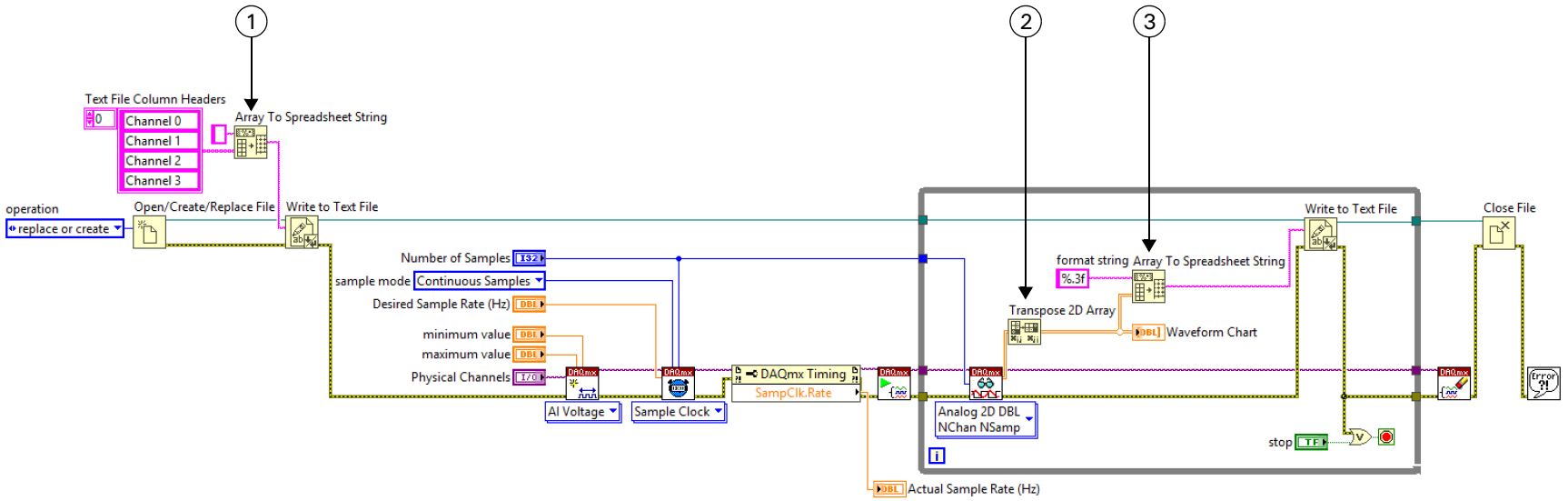
   ☐ In Windows Explorer, open and explore the text file.

5. Use a probe to view wire values to better understand the functionality of the functions and VIs. You can also create indicators of wires and view the indicator values on the front panel, if you prefer.

N Channels, *N* Samples (2D DBL) Streaming Example

1. From the **Project Explorer** window, open the Low-Level Stream to Text File (NChan NSamp) VI.

2. Explore the block diagram.

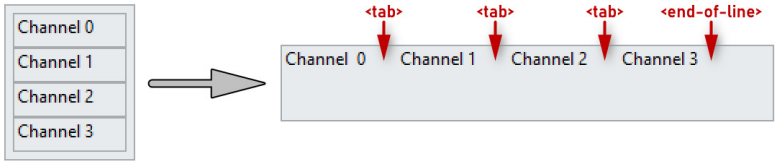


1 **Array to Spreadsheet String function**—In this VI, the Array to Spreadsheet String function is configured to create a spreadsheet string, which adds a delimiter between each array element and adds an end-of-line constant at the end of the string. In this example, it turns the incoming string array into the following string:

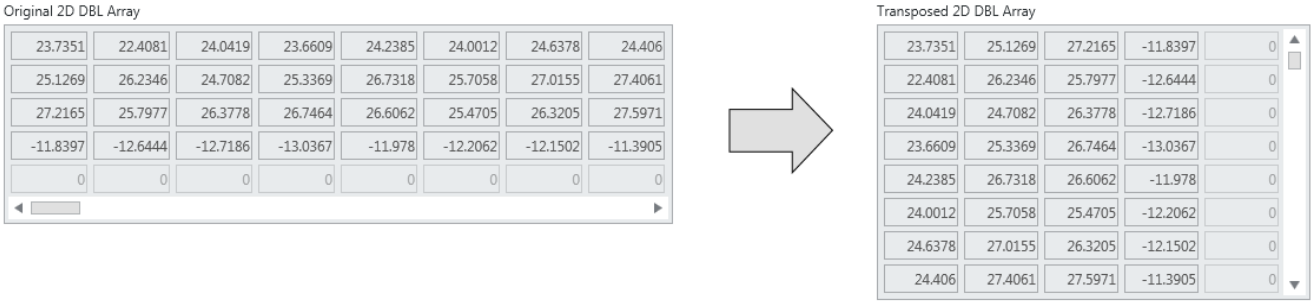
```
Channel 0<tab> Channel 1<tab> Channel 2<tab> Channel 3<end-of-line>.
```

2 **Transpose 2D Array function**—Rearranges the elements of 2D array such that 2D array becomes transposed array.
3 **Array to Spreadsheet String function**—Use this function to convert the 2D DBL array data into a string data type because the Write to Text File function requires a string input.

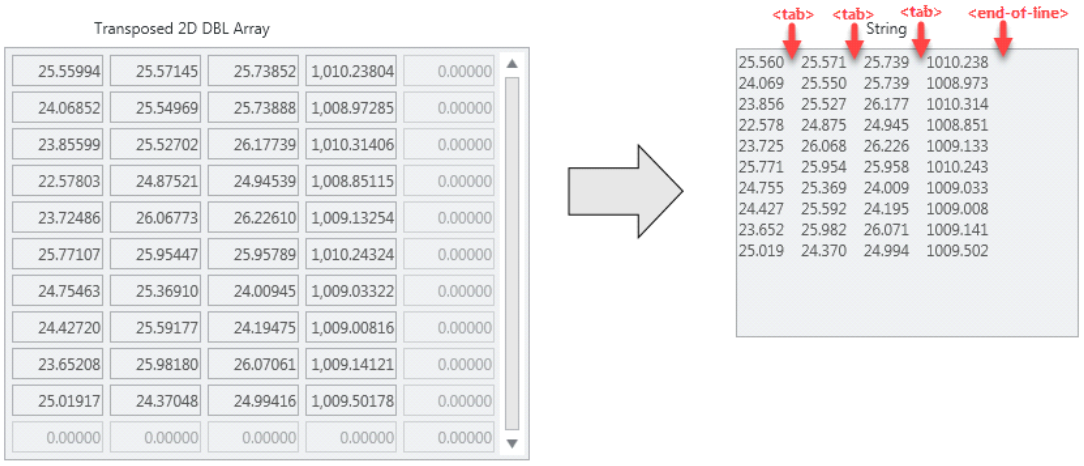
☐ The first Array to Spreadsheet String function, as configured in this exercise results in the following output.



☐ The Transpose 2D Array function rearranges the elements of the array, as shown in the following figure.



☐ The Second Array to Spreadsheet String function, as configured in this exercise, would format the output of the Transpose 2D Array function to a spreadsheet string, as shown in the following figure.



3. Test the VI.

   ☐ Run the VI.

   ☐ In the file dialog box, save the log file as `<Exercises>\LabVIEW 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 functions and VIs. You can also create indicators of wires and view the indicator values on the front 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

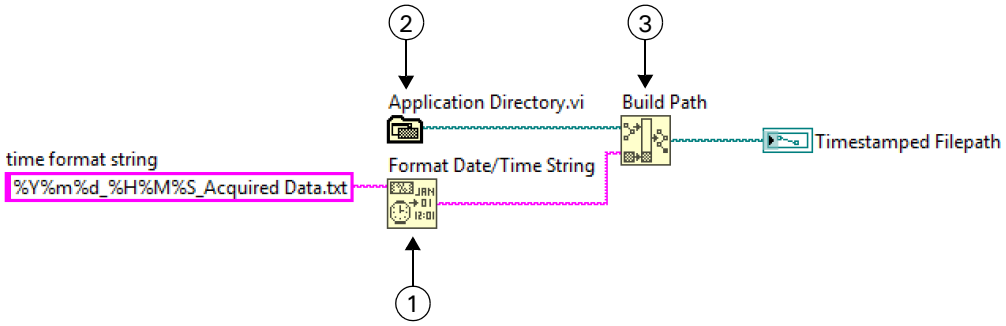
Exercise 10-4: Programmatically Creating Filenames Based on Current Timestamp

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 the following project: `<Exercises>\LabVIEW Core 1\Timestamped Filepath\Timestamped Filenames.lvproj`.
2. From the **Project Explorer** window, open the Generate Timestamped Filepath VI.
3. Examine the block diagram.



1 **Format Date/Time String function**—Converts a timestamp of the current time or numeric value into a string that displays the corresponding time. Using the time format string in this block diagram, this function will return a string similar to the following:

`<year><month><day>_<hour><minute><second>_Acquired Data.txt`

Example: `201901014_101530_Acquired Data.txt`

2 **Application Directory VI**—Returns the path to the directory containing the current project (`.lvproj`).

3 **Build Path function**—Creates a new path by appending a filename or relative path to an existing path. In this VI, the output of this function will be a timestamped filename in the directory containing the current project. (For example, `C:\Exercises\LabVIEW Core 1\Timestamped Filepath\20170601_101530_Acquired Data.txt`)

Note Refer to the *LabVIEW Help* for how to use the **time format string** input.

4. Test the code.

   ☐ Run the VI multiple times.

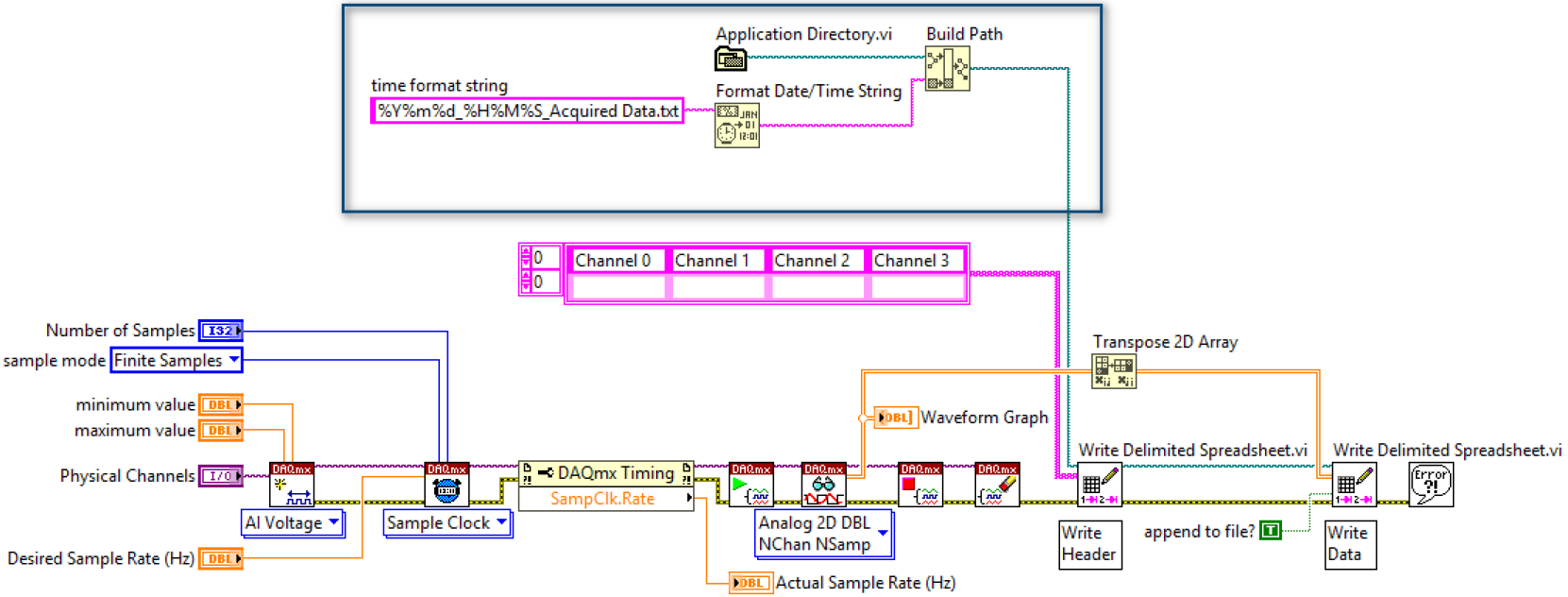
   ☐ On the front panel, notice that the generated filename changes to match the current date and time.

   ☐ Notice that the filepath directory is the same as the directory containing the current project file (`.lvproj`).

Examine Timestamped Filename Code in a Data Logging VI

1. From the **Project Explorer** window, open the [Timestamped File] High-Level Write to Text File VI.

2. Examine the block diagram. Notice how the timestamped filename code passes the timestamped filepath to the **file path** input of the file Write Delimited Spreadsheet VI.



3. Test the VI.

   ☐ Run the VI.

- ☐ Navigate to the log file in the `<Exercises>\LabVIEW Core 1\Timestamped Filepath` 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 Core 1\Timestamped Filepath` 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 Core 1\Timestamped File` directory.

`<year><month><day>_<hour><minute><second>_<your own custom filename>`

Example: 20190511_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: Reading and Analyzing Data from a File in LabVIEW

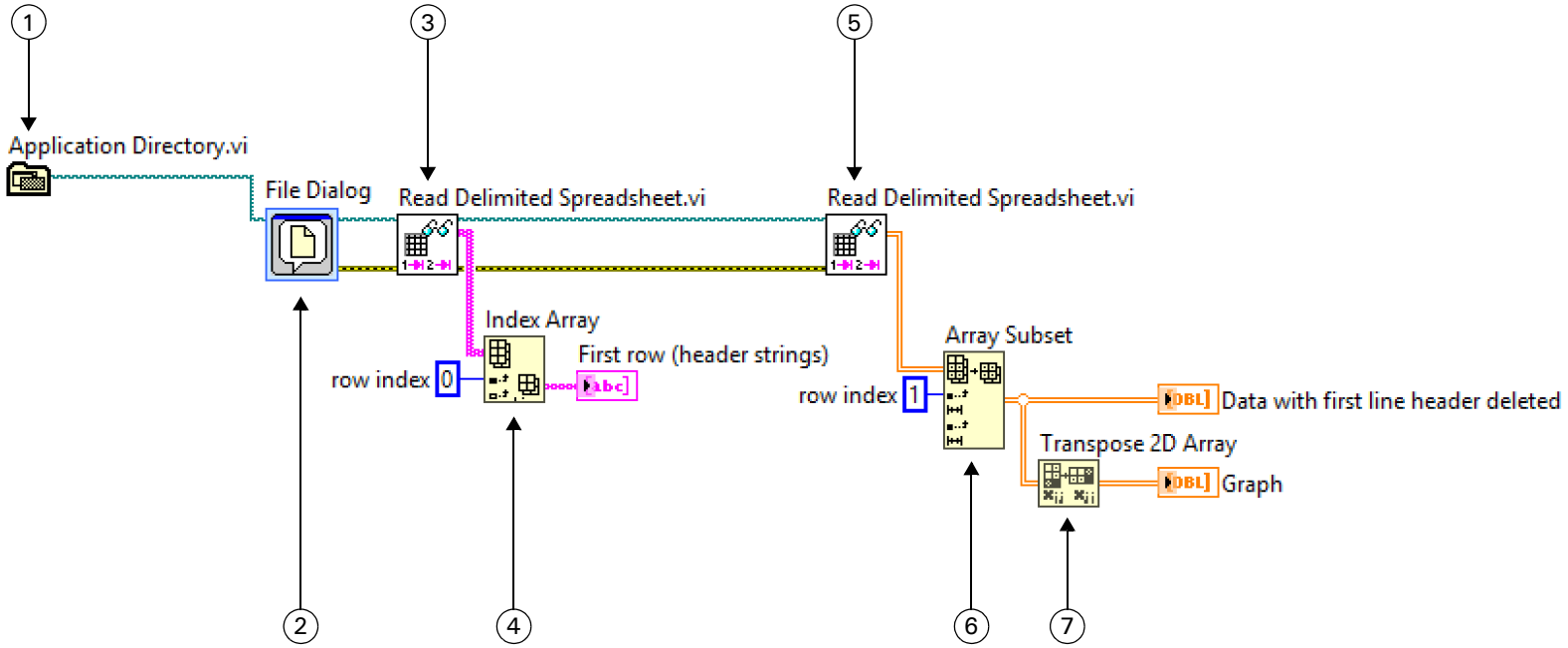
Goal

Read and analyze data from a text file in LabVIEW.

Instructions

1. Examine the text file containing the data that we want to read into LabVIEW.
   - ☐ Open and examine the following text file: `<Exercises>\LabVIEW 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 the following project: `<Exercises>\LabVIEW Core 1\Read Delimited File\Read Delimited File into VI.lvproj`.
3. From the **Project Explorer** window, open the Read Delimited File into VI.

4. Examine the block diagram.



1 **Application Directory VI**—Returns the path to the directory containing the current project (`.lvproj`)
2 **File Dialog Express VI**—Displays a dialog box with which a user can specify the path to a new or existing file or directory.
3 **Read Delimited Spreadsheet VI (string)**—Returns the contents of the delimited text file as a 2D string array.
4 **Index Array function**—Extracts the first row, which contains a string for each column header.
5 **Read Delimited Spreadsheet VI (DBL)**—Returns the contents of the delimited text file as a 2D DBL array.
6 **Array Subset function**—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 contains only the numeric data.
7 **Transpose 2D Array function**—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 box, navigate to the following text file: `<Exercises>\LabVIEW Core 1\Read Delimited File\Acquired Data (NChan NSamp).txt`.

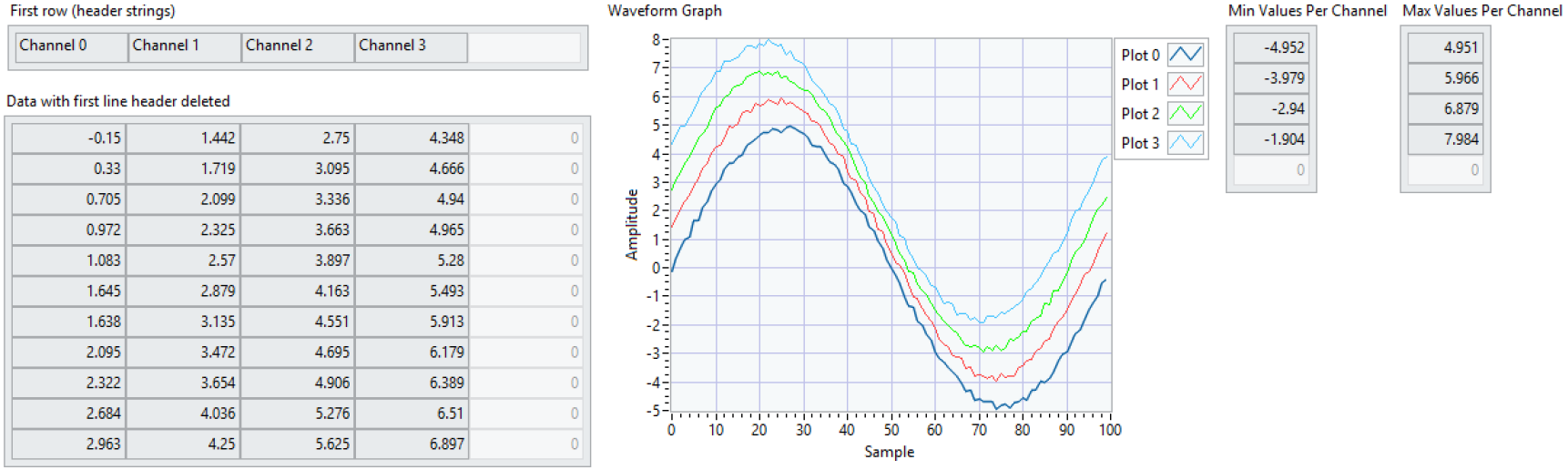
   ☐ Use Highlight Execution and probes to examine the functionality of this VI.

Your Turn

1. Open and examine the following text file: `<Exercises>\LabVIEW Core 1\Read Delimited File\Acquired Data (NChan NSamp).txt`.
2. Modify the block 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 front panel should look similar to the following figure.



End of Exercise 10-5

11 Reusing Code

In this lesson, you learn to recognize the benefits of reusing code and create a subVI with a properly configured connector pane, meaningful icon, documentation, and error handling.

Exercises

Exercise 11-1 Creating and Using a SubVI

Exercise 11-1: Creating and Using 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 file path 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

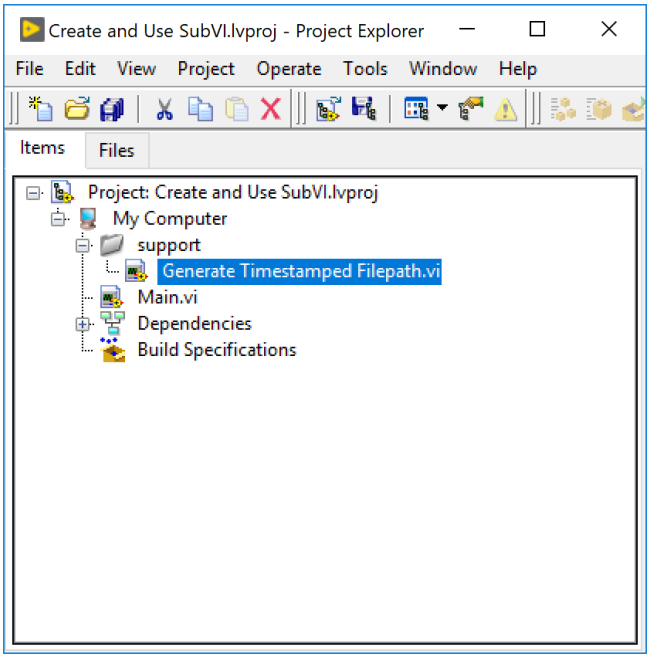
The Generate Timestamped Filepath VI contains the following inputs and outputs.

Inputs	Outputs
Appended filename	timestamped relative filepath
Error In	Error Out

Creating a SubVI

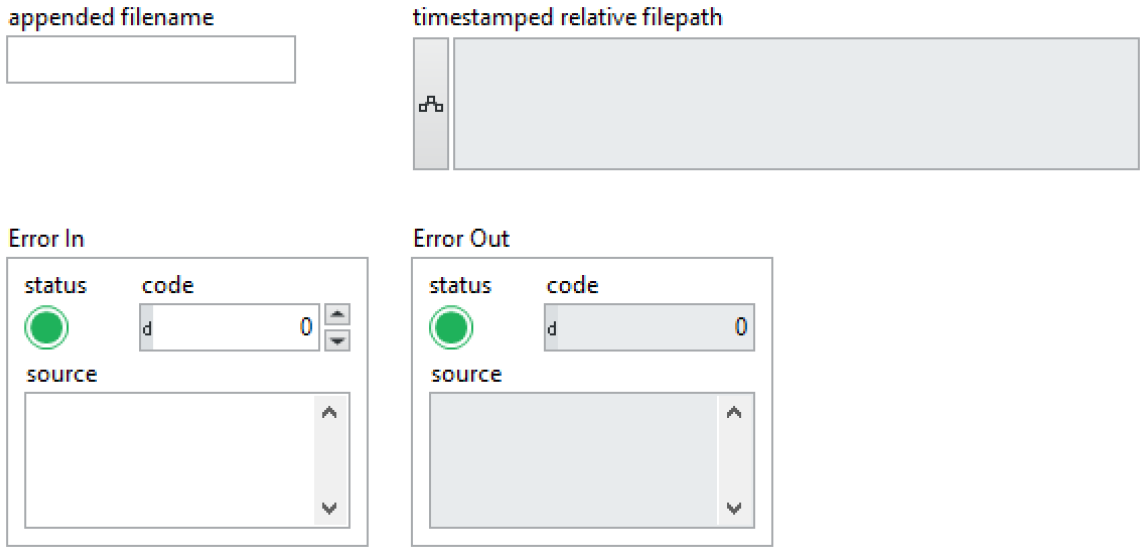
1. Open `<Exercises>\LabVIEW Core 1\Create SubVI\Create and Use SubVI.lvproj`.

2. From the support folder of the **Project Explorer** window, open the Generate Timestamped Filepath VI.



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.

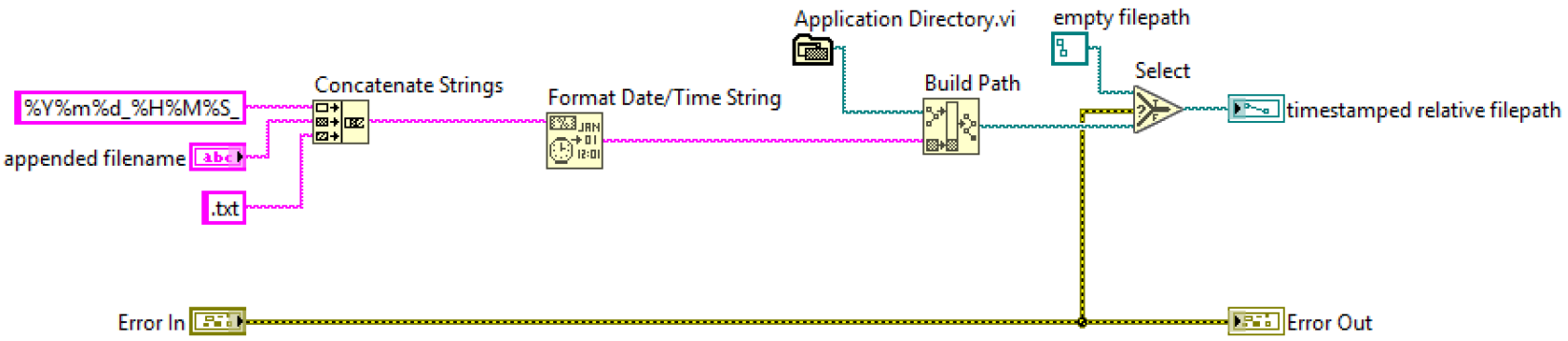
3. Place an **Error In** control and **Error Out** indicator on the front panel, as shown in the figure below.



4. Modify the block diagram, as shown in the following figure.

   ☐ Drag a **Select** function and **Path** constant to the block diagram from the Quick Drop menu.

   ☐ Wire the block diagram as shown in the figure below.

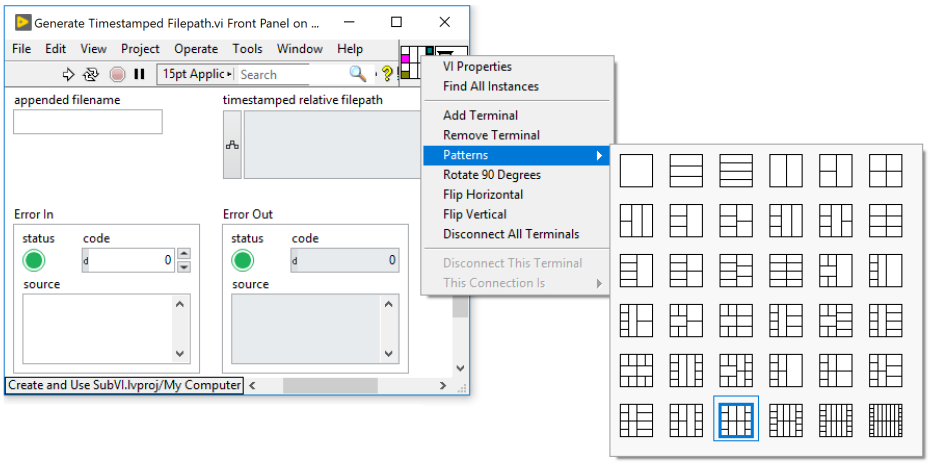


5. Connect the inputs and outputs to the connector pane, as shown in the following figure below.

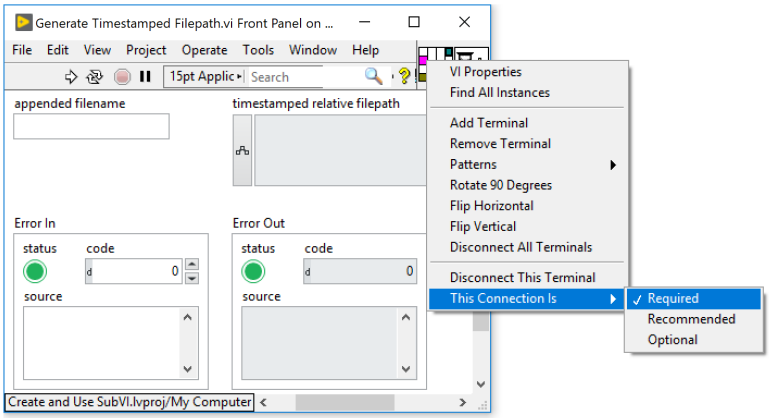
☐ The connector pane is located in the upper-right corner of the front panel.

☐ Click any input or output you want to assign and then click the corresponding control or indicator on the front panel.

☐ You can change the number of input and output terminals by right-clicking the connector pane and selecting a suitable connector pane pattern in the **Patterns** shortcut menu.



☐ Right-click the **appended filename** input terminal and select **This Connection Is»Required** from the shortcut menu.

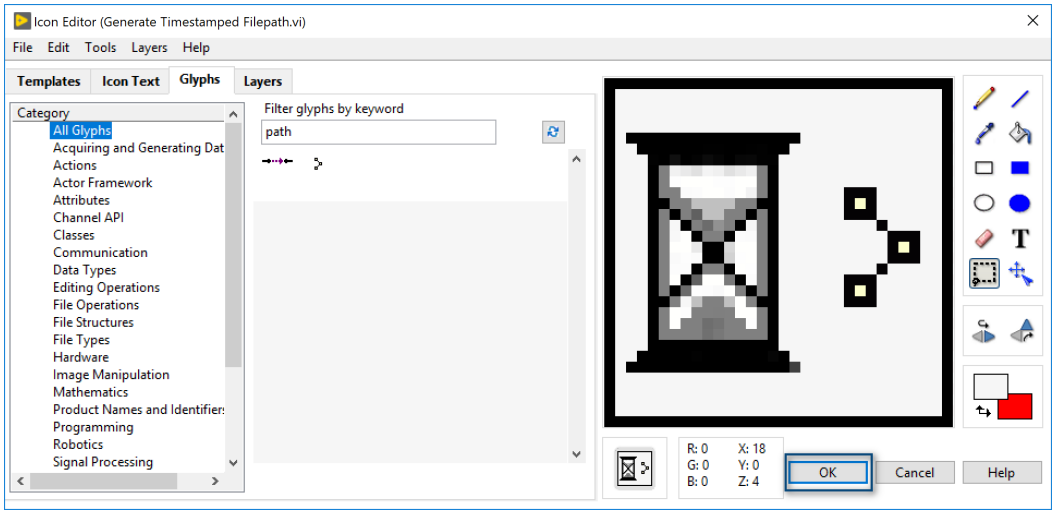


Now when you use this subVI in a main VI, the **Run** button of the main VI will be broken until the **appended filename** input terminal is wired.

6. Create an icon, as shown in the figure below.

   ☐ Double-click the VI icon in the upper-right corner of the front panel to open the Icon Editor. Search for `delay` and `path` and place the corresponding graphics onto the VI icon.

   ☐ Click the **OK** button when finished.



7. Create a description for the subVI.

   ☐ Click **File** in the upper-left corner of the VI and select **VI Properties**.

   ☐ Select **Documentation** from the **Category** pull-down menu and enter the following description: `Generates a timestamped filepath with an appended user-specified name and the same directory containing the project.`

   ☐ Click **OK**.

8. Save the VI.

9. Test the VI when **Error In** contains no error.

   ☐ On the front 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 file path similar to `<Exercises>\LabVIEW Core 1\Create SubVI\20190509_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

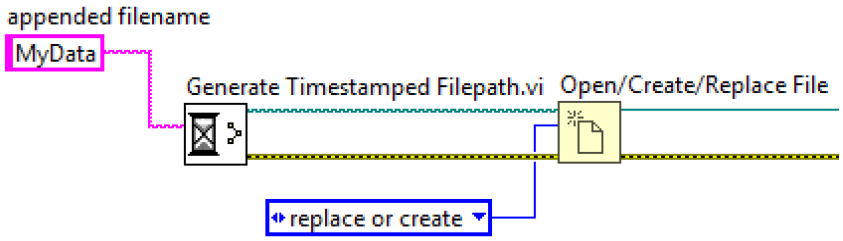
1. From the **Project Explorer** window, open the Main VI.
2. Examine the Open/Create/Replace File function on the block diagram.

   Notice that its file path input is unwired, which means when you run the VI, this function will launch a file dialog for you to specify the file path.
3. Use the Generate Timestamped Filepath subVI to output a timestamped filename in the same directory as the current project file (.lvproj).

   ☐ Drag the **Generate Timestamped File** VI from the **Project Explorer** window to the block diagram of the 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 of the Generate Timestamped Filepath VI to the Open/Create/Replace File function.



4. View the subVI description.

   ☐ Open the Context Help window by pressing <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.

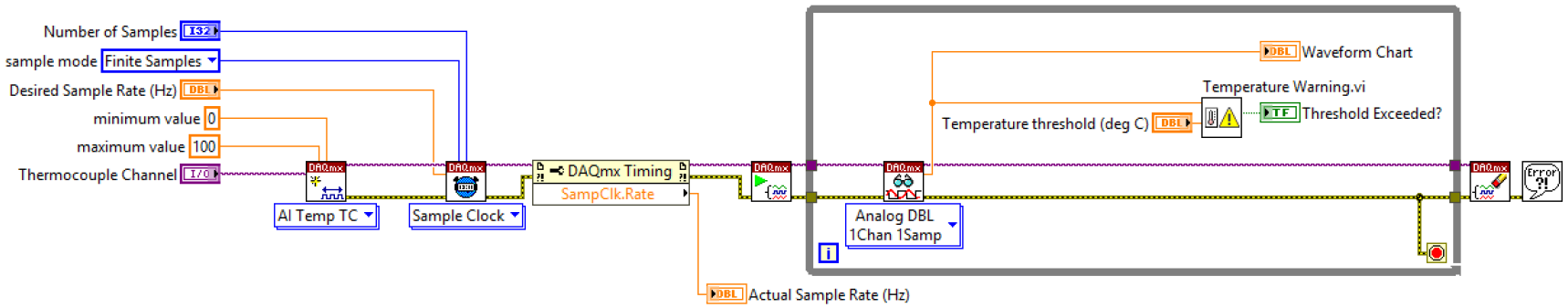
   ☐ In Windows Explorer, go to the `<Exercises>\...\Create and Use SubVI` directory and notice the timestamped file created by the VI.

   ☐ 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 a temperature measurement, compares it to the set temperature threshold, and returns if the threshold is exceeded or not.

The following figure shows how your completed Temperature Warning subVI could be used in an example VI.



Note For the answer, refer to the `<Solutions>\LabVIEW Core 1\11-1\[Your Turn] Create SubVI` 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

12 Grouping Data of Mixed Data Types

In this lesson, you learn how to create and use clusters to group data of mixed types.

Exercises

Exercise 12-1 Grouping Related Data Using a Cluster

Exercise 12-1: Grouping Related Data Using a Cluster

Goal

- Group related data into a new cluster.
- Create a 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.

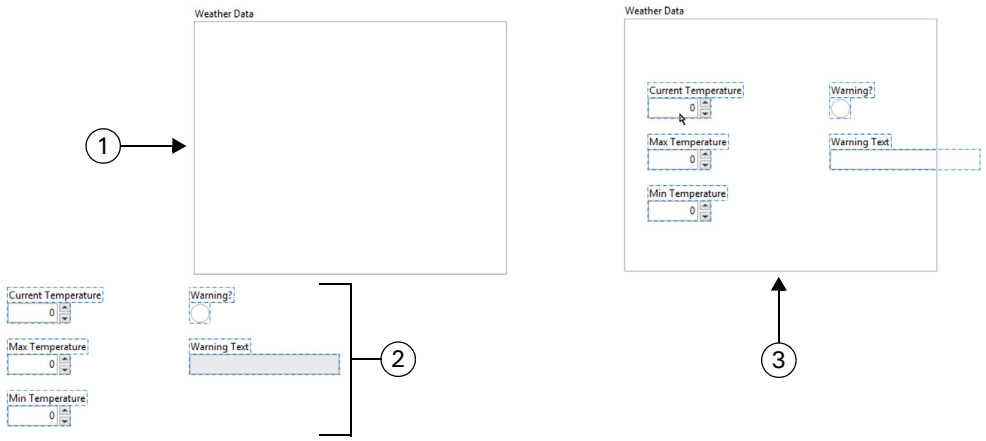
Another developer has created a VI that displays temperature warnings. This VI is part of a temperature weather station project. Your task is to update this VI to use clusters instead of individual controls/indicators for inputs and outputs.

Instructions

Determine Warnings SubVI

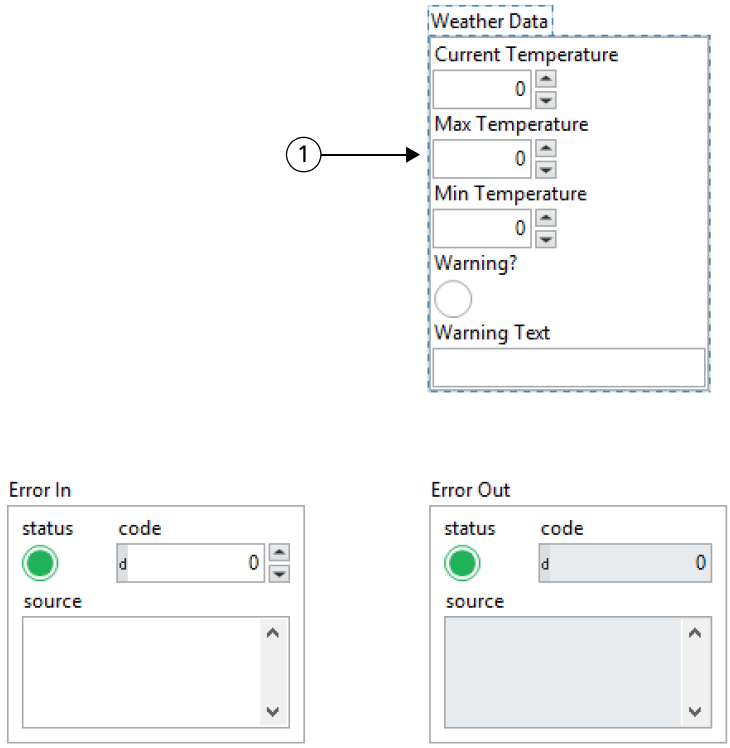
In this section, you will create a subVI that handles temperature warnings. All the weather-related data is grouped together in a Weather Data cluster.

1. Open `<Exercises>\LabVIEW Core 1\Clusters\Weather Warning Cluster\Weather Warnings.lvproj`.
2. From the **Project Explorer** window, open the Determine Warnings VI.
3. Place existing controls and indicators in a cluster named `Weather Data`, as shown in the following figure.



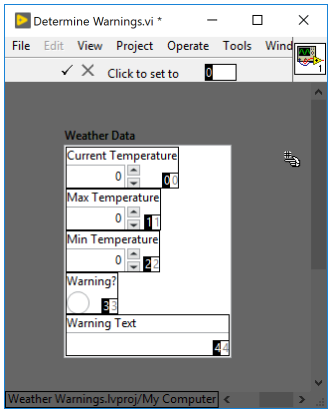
1	**Cluster**—Drop a cluster control from the Data Containers palette and change the label to `Weather Data`.
2	Select controls and indicators to include in the cluster. <Shift>-click to select multiple objects.
3	Drag the controls and indicators into the **Weather Data** cluster.

4. Resize the cluster so that all the elements are visible and arranged vertically, as shown in the following figure.



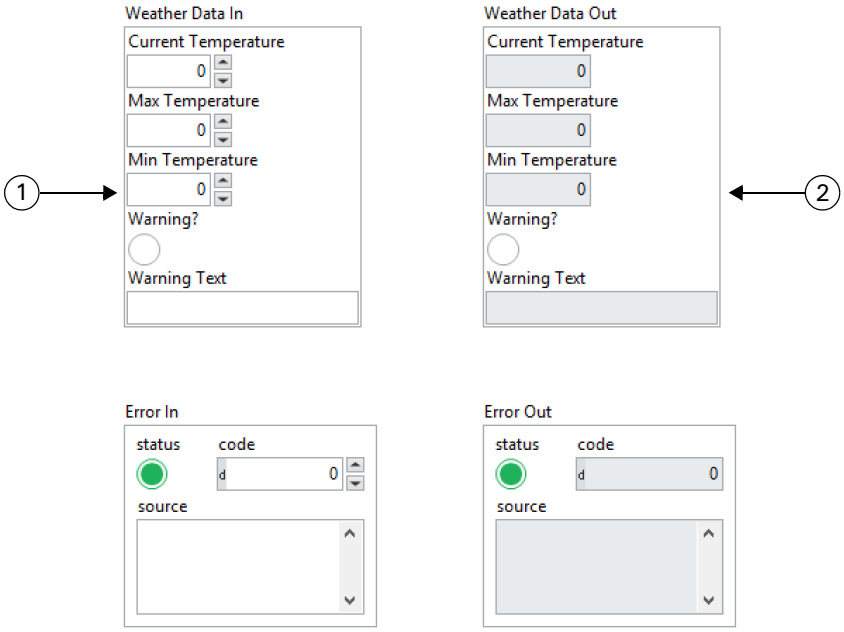
1 Autosize cluster—LabVIEW can rearrange and resize the cluster for you. Right-click the border of the **Weather Data** cluster and select **AutoSizing»Arrange Vertically**.

5. Reorder the items in the cluster, as shown in the following figure.



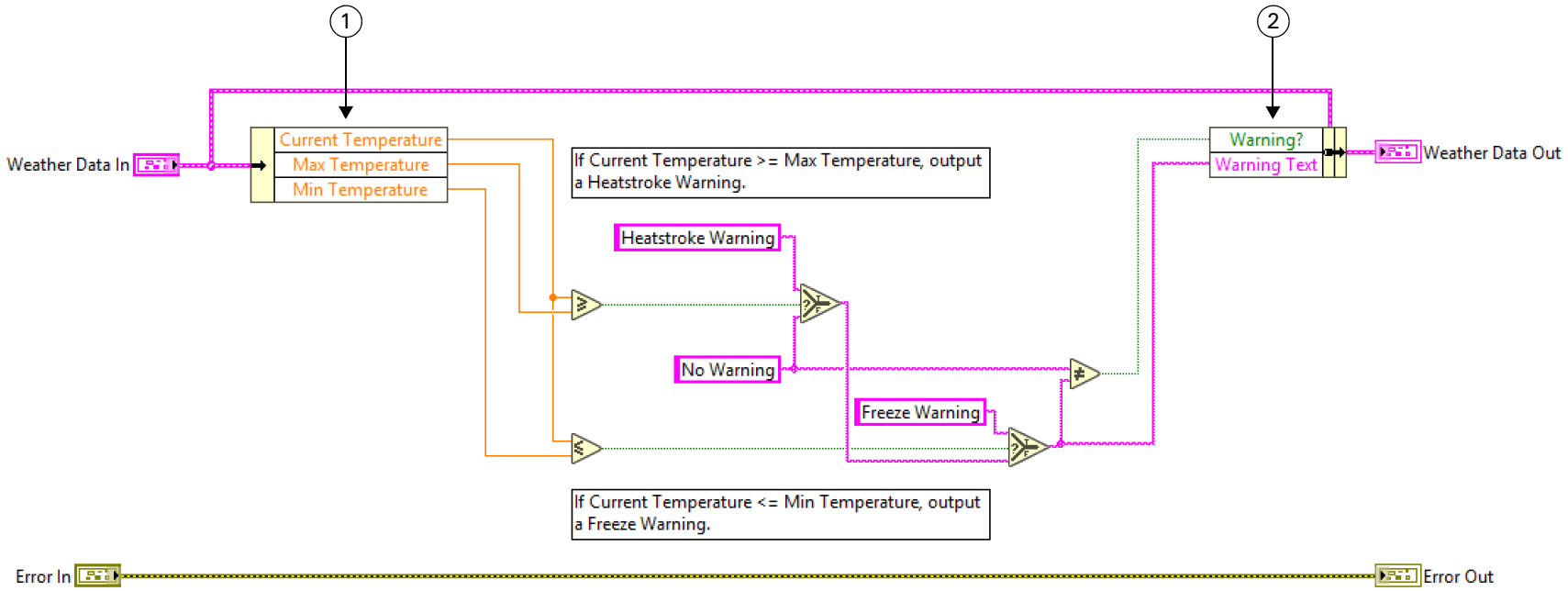
☐ Right-click the border of the **Weather Data** cluster and select **Reorder Controls In Cluster**. Verify that your **Weather Data** cluster order matches the order in this figure.

6. Modify the VI to receive and return cluster data.



1 **Weather Data In**—<Ctrl>-click the **Weather Data** cluster and drag it to create a copy. Rename the copy `Weather Data In`.
2 **Weather Data Out**—Right-click the original **Weather Data** cluster and select **Change to Indicator**. Rename the indicator `Weather Data Out`.

7. Modify the block diagram, as shown in the following figure, to extract and modify data from the input cluster.



1 **Unbundle By Name**—Wire the **Weather Data In** cluster and expand the Unbundle By Name function to display three items. Wire the outputs of the Unbundle By Name function to the broken wires in the order shown. Because you moved individual controls and indicators into a single cluster, you must use the Unbundle By Name function to access each cluster element.

2 **Bundle By Name**—Wire the **Weather Data In** cluster around the analysis code to the input cluster of the Bundle By Name function. Display two elements and click to select the **Warning?** and **Warning Text** elements. Connect the broken wires to the inputs of Bundle By Name as shown.

Note If the order of the elements in the Unbundle By Name or Bundle By Name functions 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 Out** indicator displays correct values.

3. Save and close the VI.

Explore Another VI Using the Weather Data Cluster

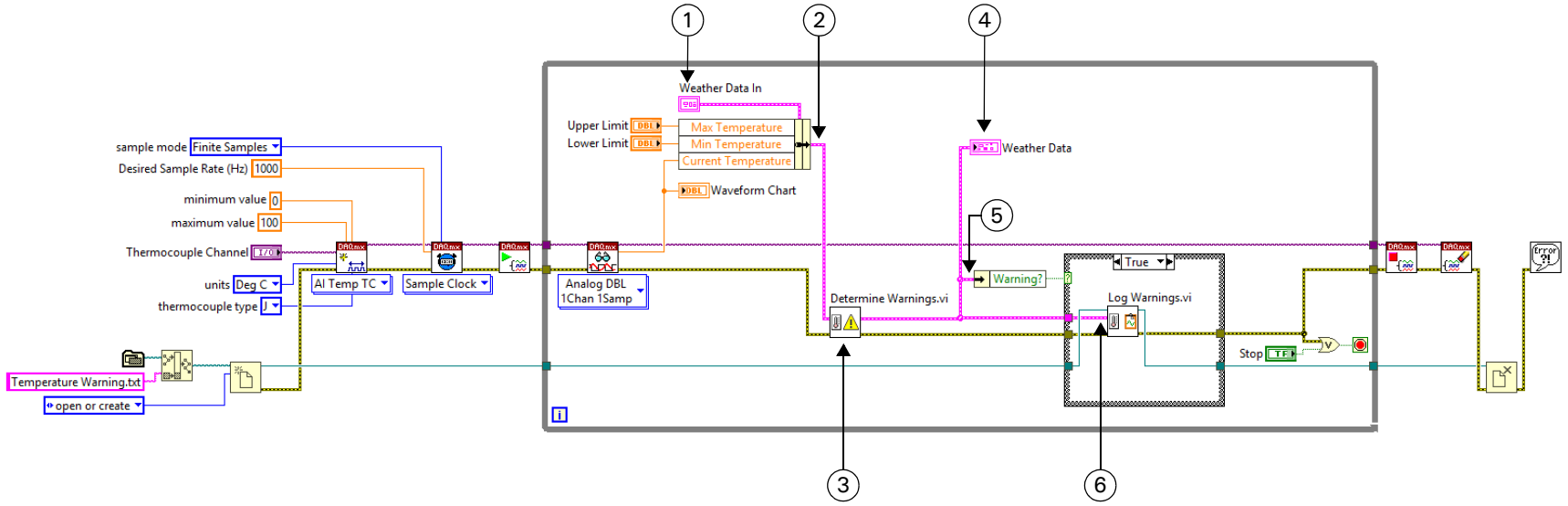
Main VI

In this section, you explore a full example that uses the Weather Data cluster in multiple places.

1. Open `<Exercises>\LabVIEW Core 1\Clusters\Full Weather Warnings Example\Full Weather Warnings Example.lvproj`.

2. From the **Project Explorer** window, open the Main VI.

3. Examine the block diagram. Notice the following places where the VI uses the Weather Data cluster.

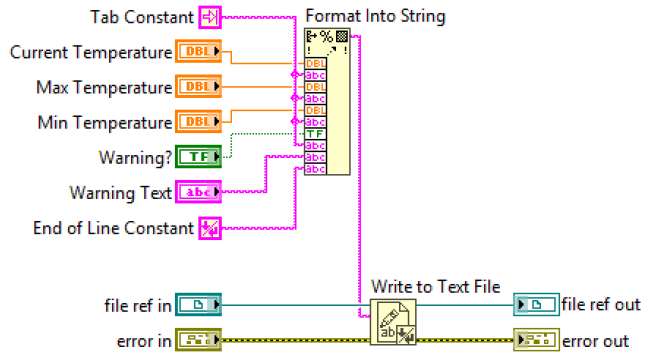


1	Weather Data cluster constant as input to the **Bundle By Name** function.
2	Output of the **Bundle By Name** function.
3	Input and output terminals of the Determine Warnings subVI.
4	Weather Data indicator.
5	Input of the **Unbundle By Name** function.
6	Input terminal of the Log Warnings subVI.

Your Turn

Log SubVI

How would you modify the following subVI block diagram to use a cluster instead of 5 individual controls?



Note To view the solution, open the Log Warnings subVI from `Full Weather Warnings Example.lvproj`.

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

13 Propagate Data Type Changes Using Type Definitions

In this lesson, you use type definitions to propagate data type changes throughout an application.

Exercises

Exercise 13-1 Using a Type Definition

Exercise 13-1: Using a Type Definition

Goal

- Create a weather data cluster type definition.
- Place the type definition in several places (e.g. subVI input/output, main VI controls/indicators).
- Update the type definition and save to propagate data type changes to all instances of the type definition

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 type definition, which is defined as follows:

Type Definition—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 type definition propagate through the application.

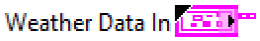
When completed, the Weather Station application monitors temperature and wind information.

1. Open `<Exercises>\LabVIEW Core 1\Type Def.s\Type Def.s.lvproj`.
2. From the **Project Explorer** window, open the Determine Warnings VI.
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 block 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 type definition.
   - ☐ On the front panel, right-click the border of the **Weather Data In** control and select **Make Type Def**.

     Notice in the **Project Explorer** window that this has created a new file on your hard drive marked **Control 1**.
   - ☐ On the block diagram, the **Weather Data In** terminal now has a black triangle on the upper-left edge indicating that it is connected to a type definition.



   - ☐ Right-click the **Weather Data In** control and select **Open Type Def.** to display the type definition in a new window.

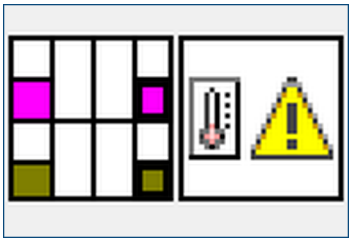
     The type definition window looks like the front panel of a VI but it does not have a block diagram.
   - ☐ Rename the cluster as `Weather Data`.
   - ☐ Press <Ctrl-S> to save the type definition and name the type definition file as `Weather Data`.
   - ☐ In the **Project Explorer** window, move the type definition file into the `support` directory by dragging the file.
   - ☐ Close the type definition window when finished.
   - ☐ On the block diagram of the Determine Warnings VI, notice the coercion dot on the **Weather Data Out** indicator terminal. This shows that the indicator is not tied to the type definition.

5. Replace the **Weather Data Out** indicator with the type definition you just created.
   - ☐ On the front panel of the Determine Warnings VI, delete the **Weather Data Out** indicator.
   - ☐ Drag the **Weather Data** type definition item from the **Project Explorer** window to the front panel.
   - ☐ Right-click the border of the Weather Data type definition on the front panel and select **Change to Indicator**. Rename the indicator as `Weather Data Out`.
   - ☐ Go to the block diagram and notice that the Weather Data Out terminal has a black triangle on the upper-left edge, which indicates that it is connected to a type definition.

☐ 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.

– The connector pane is located in the upper-right corner of front panel.

– Connect the second-from-the-top terminal on the right side to the Weather Data Out indicator, as shown in the following figure.



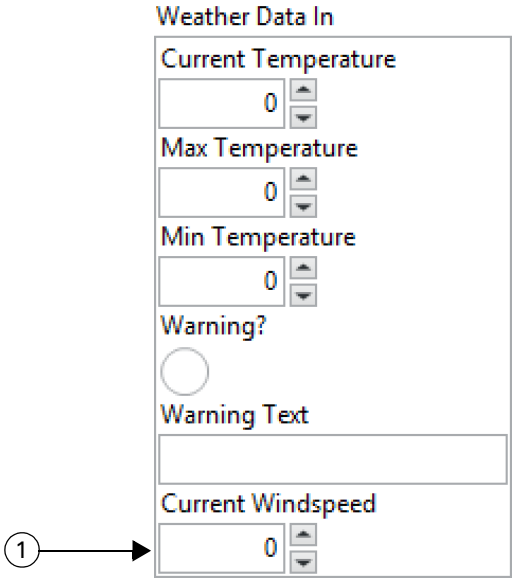
Note You can no longer add or remove elements to or from a type definition cluster control/indicator on the front panel. You must open and edit the type definition file to add or remove the element.

☐ Save the Determine Warnings subVI.

6. Edit the Weather Data type definition to include wind speed information.

☐ Right-click the border of the **Weather Data In** control and select **Open Type Def.** from the shortcut menu.

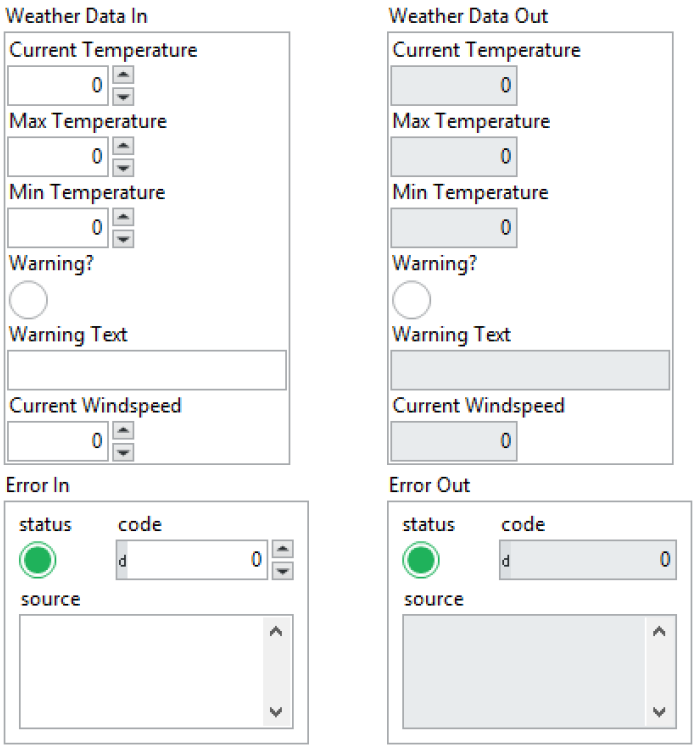
☐ Modify the front panel as shown in the following figure.



1 **Numeric Control**—Place a numeric control in the cluster and rename it as `Current Windspeed`.

☐ Apply the changes (File»Apply Changes) and Save the Weather Data type definition before closing its window.

☐ 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 definition. Arrange the front 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 the Main VI and Log Warnings VI. Replace all instances (listed below) of the cluster in these VIs to use the Weather Data type definition.

   ☐ Weather Data instances in the Log Warnings VI

   – Weather Data In cluster

   Note Make sure you reconnect the new type definition to the connector pane

   ☐ Weather Data instances in the Main VI

   – Weather Data cluster constant

   – Weather Data indicator

2. Modify the Weather Data type definition 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 type definition in all VIs are updated to include the new elements.

Your Turn

1. Create a new project and a new VI.
2. Create an enum that contains 3 items (e.g., Add, Subtract, and Multiply).
3. Convert the enum into a type definition.
4. Add the type definition to multiple places and VIs.
5. Modify the enum type definition to include a fourth item (for example, Divide). Save the type definition.
6. Verify that all instances of the enum type definition 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 a type definition?

   If the cluster or enum exist in more than one place, you should definitely make it a type definition.

End of Exercise 13-1

14 Implementing a Sequencer

In this lesson, you learn how to sequence the tasks in your application by using the state machine design pattern.

Exercises

Exercise 14-1: Creating a State Machine

Goal

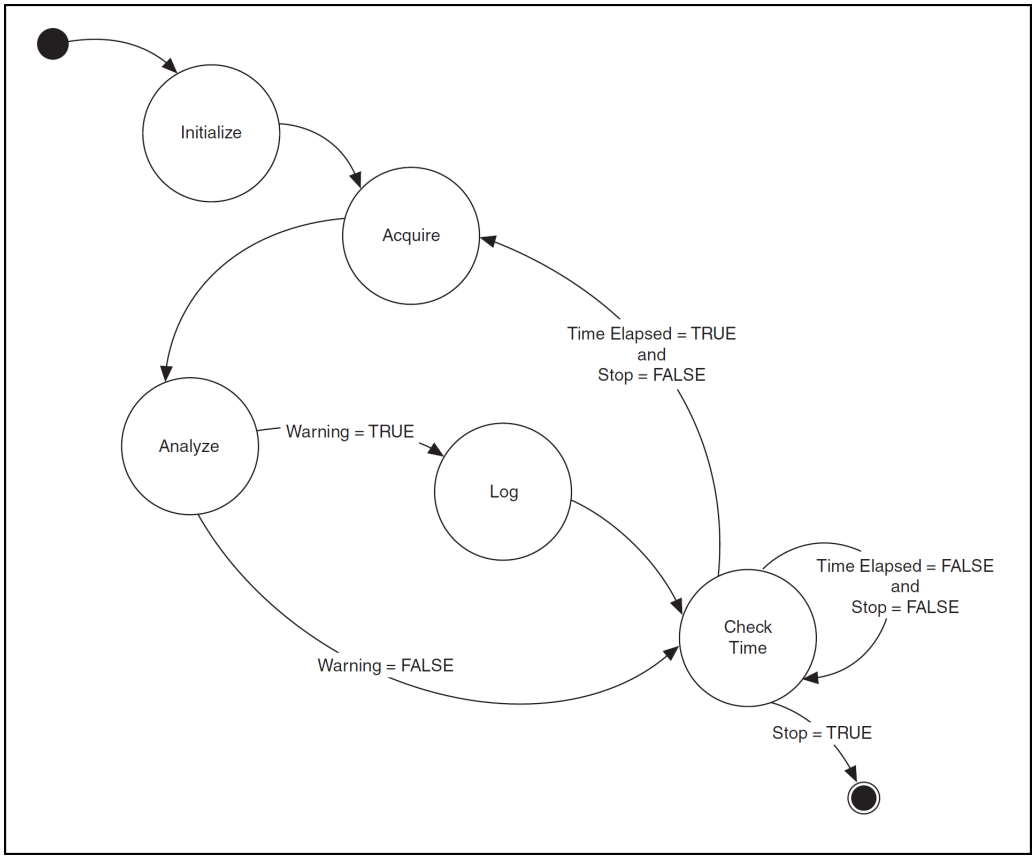
Create a VI that implements a state machine using a type definition enum.

Scenario

You must design a VI for a user interface state machine. The VI acquires a temperature every half a 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.



The following table describes the states in this state machine.

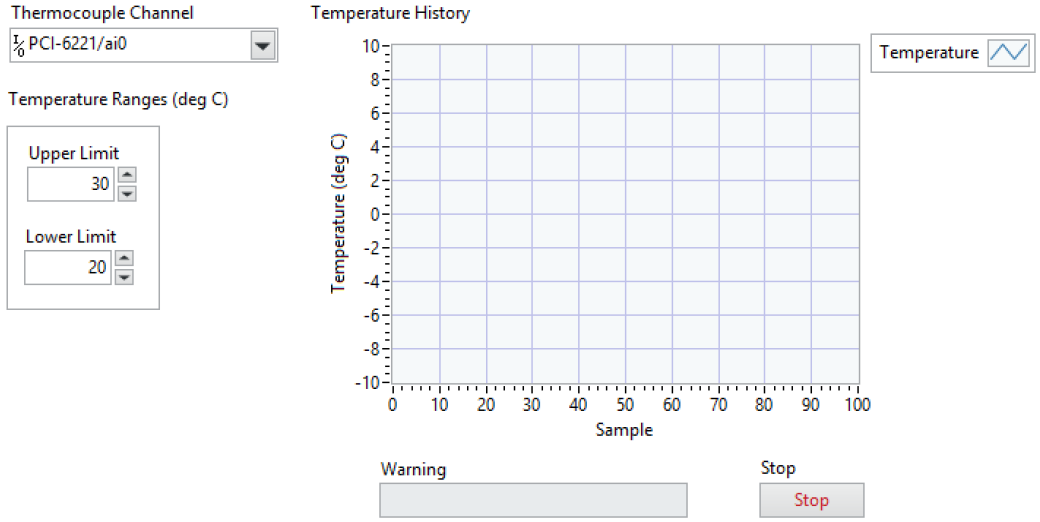
State	Description	Next State
Acquire	Set time to zero and acquire data from the temperature sensor	"Analyze"
Analyze	Read front panel controls and determine the warning level	"Log" if a warning occurs "Check Time" if no warning occurs
Log	Log the data in a tab-delimited ASCII file	"Check Time"
Check 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.lvproj` in the `<Exercises>\LabVIEW Core 1\State Machine` directory.

2. Open the Weather Station UI VI from the **Project Explorer** window.

   The figure below shows the front panel of the Weather Station UI VI that has been provided for you. You modify the block diagram to create a state machine for the weather station.



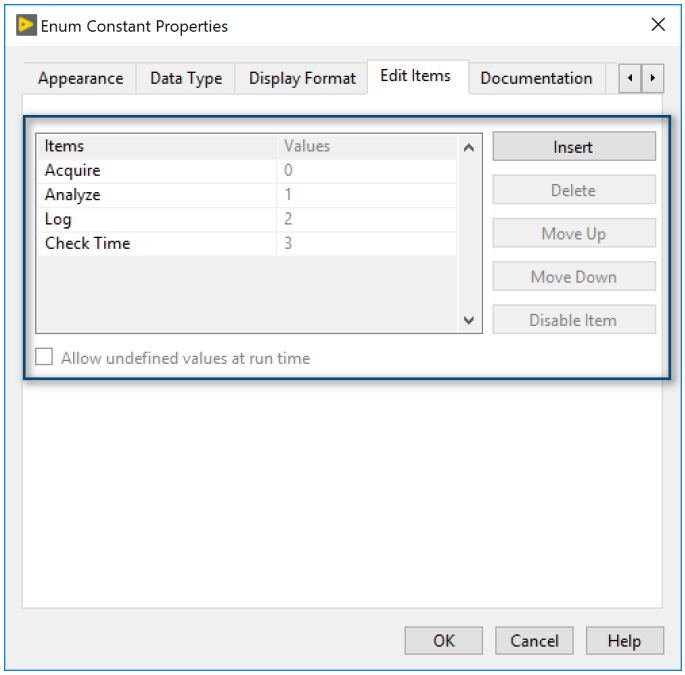
The following figure shows the starting point of the block diagram for the Weather Station UI VI. You edit this block diagram to implement a state machine for the weather station application.

1 These are the unused controls and indicators from the front panel. You will use these controls and indicators to program different cases.

3. Create a new type definition to control the weather station application.

   ☐ Open the block diagram and create an enum constant to the left of the While Loop.

   ☐ Right-click the enum constant and select **Edit Items** to open the **Enum Constant Properties** window. On the **Edit Items** tab, add the enum items, as shown in the following figure.



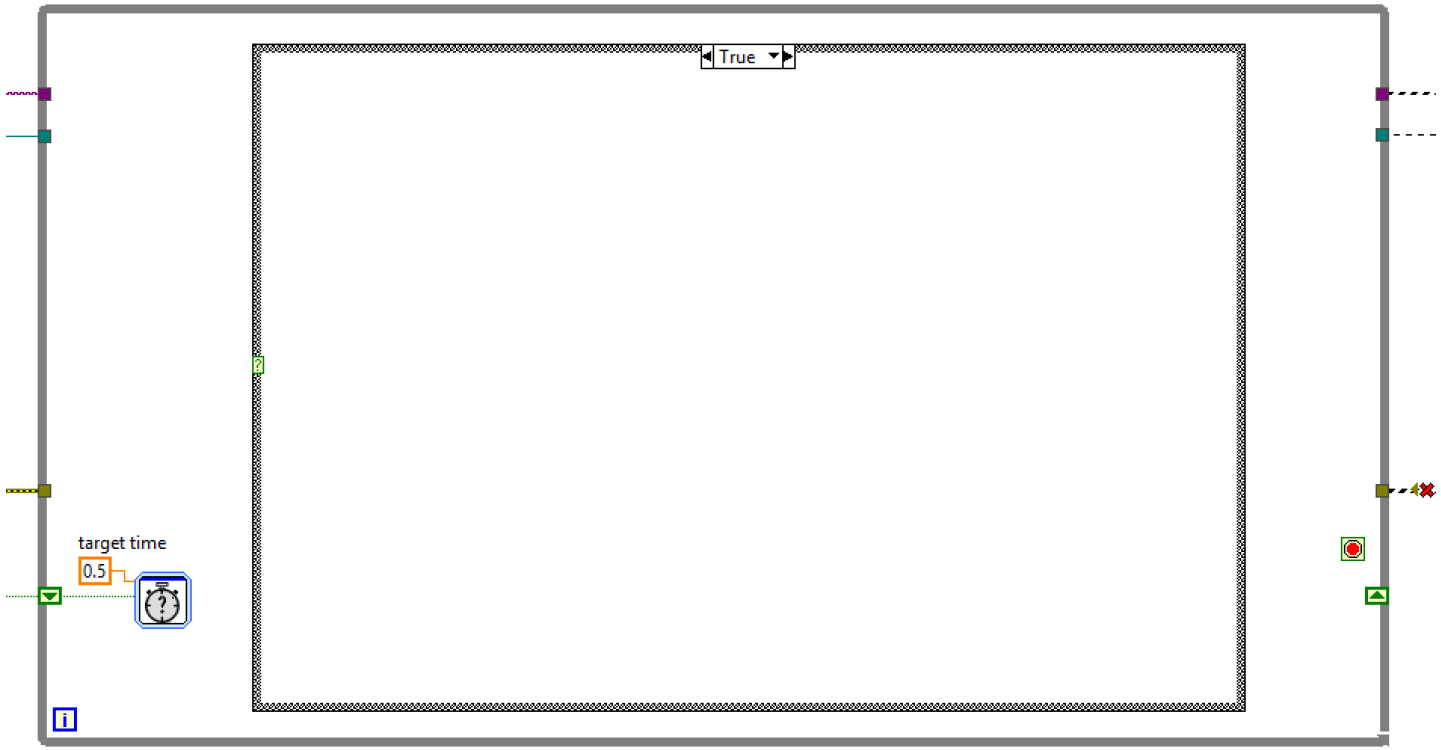
   ☐ Right-click the enum constant on the block diagram and select **Make Type Def.**

4. Modify the new type definition and add it to the Weather Station project.

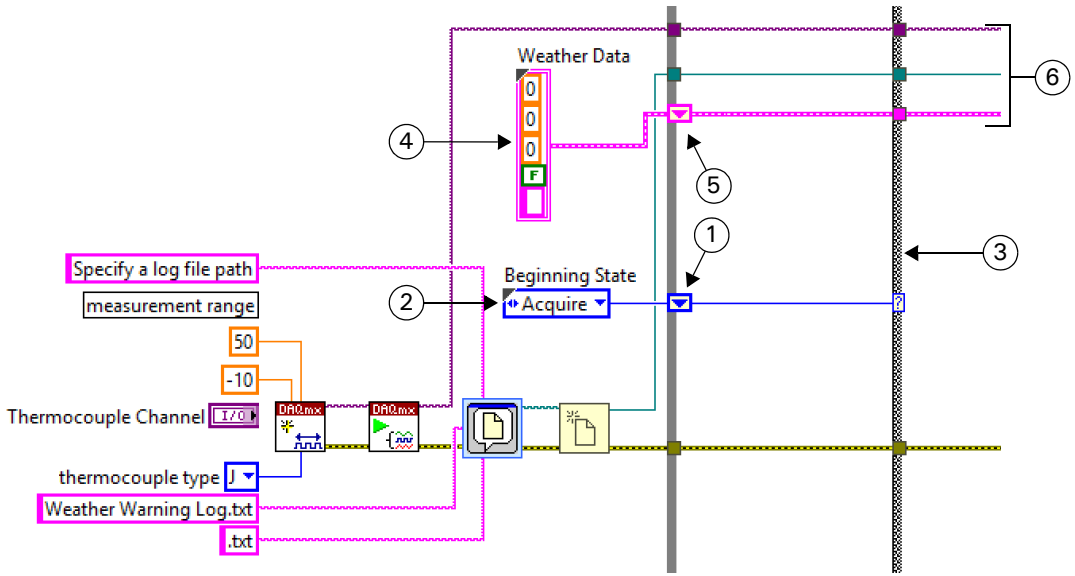
   ☐ Right-click the enum constant and select **Open Type Def**.

- ☐ Rename the label on the enum control to `States`.
- ☐ Apply the changes made to type definition (File»Apply Changes).
- ☐ Save the type definition as `Weather Station States` in the `<Exercises>\LabVIEW Core 1\State Machine\Supporting Files` directory.
- ☐ Close the **Type Def** window.
- ☐ In the **Project Explorer** window, drag the Weather Station States type definition 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.



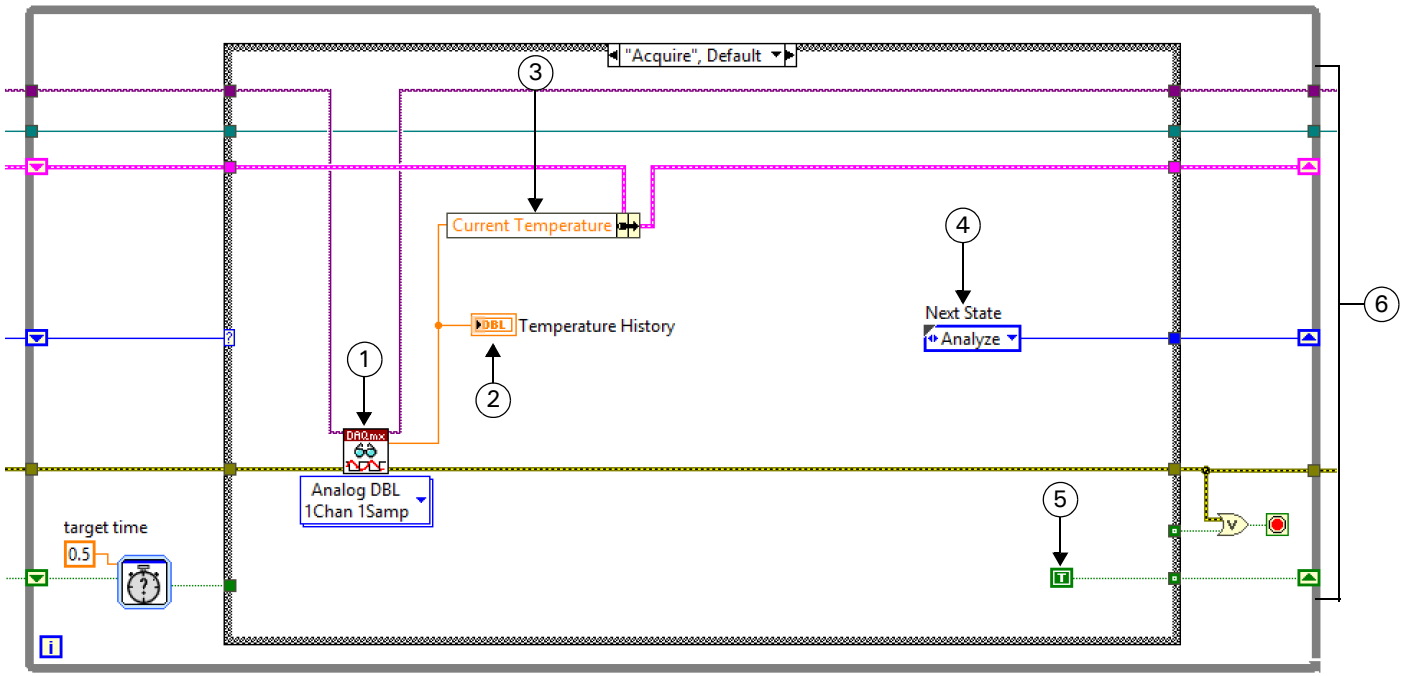
6. Control the state machine with the type definition enum and update the framework as shown in the following figure.



1 **Shift Register**—Right-click the While Loop and select **Add Shift Register**.
2 **Type definition enum** constant—In the **Properties** window, enable the **Show Label** checkbox. Rename the label as `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 **Add Case For Every Value** to create different cases for each value in the enum.
4 **Weather Data**—Drag the **Weather Data** type definition from the **Project Explorer** window to the block diagram to create a type definition 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.



1 **DAQmx Read**—In the Polymorphic VI Selector, set the following values:
Channel Type: Analog
Channel Count: Single Channel
Sample Count: Single Sample
Data Format: DBL

2 **Temperature History**—Move this indicator into the Acquire state of the Case Structure.

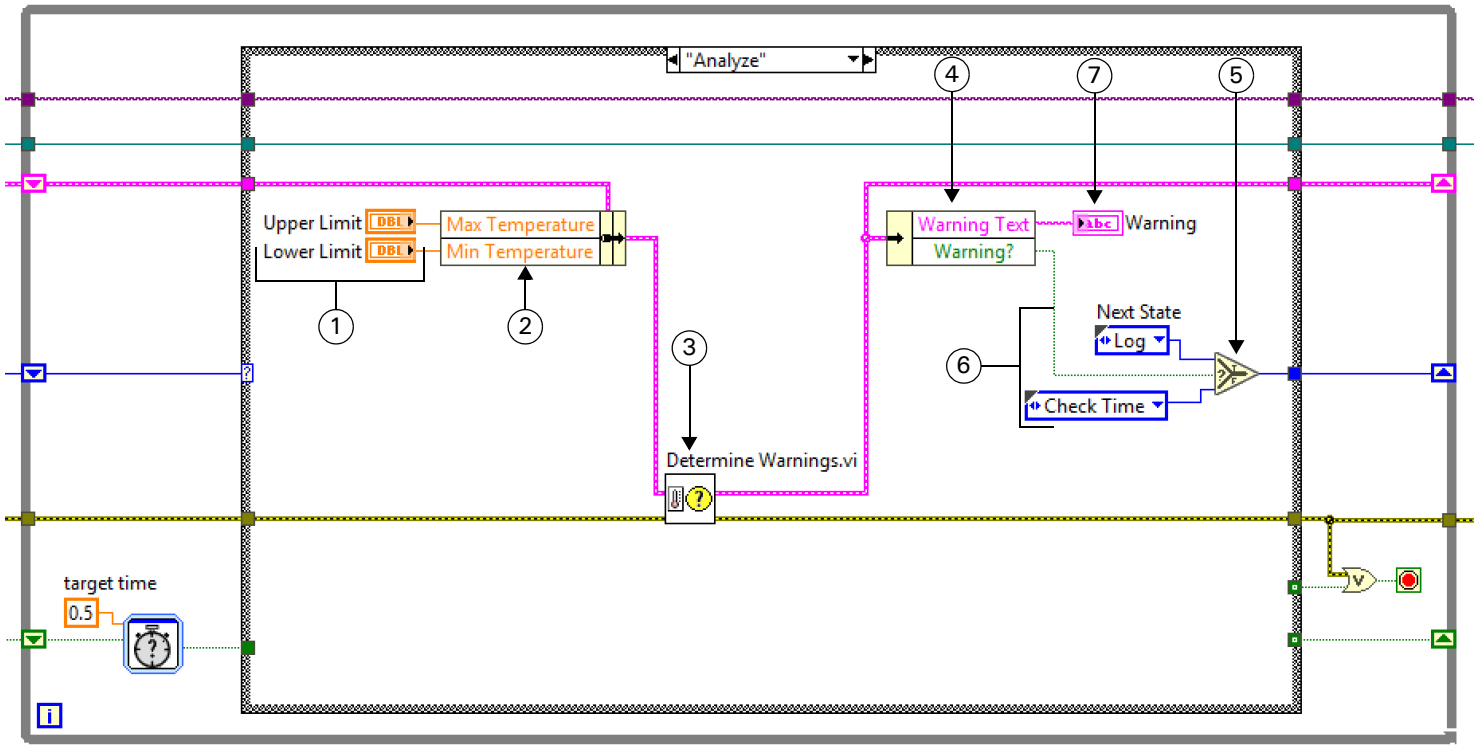
3 **Bundle By Name** —Set the element to **Current Temperature.** Wire the data output of the DAQmx Read VI to the **Current Temperature** input.

4 **Next State enum**—Press <Ctrl> and click the **Beginning State** enum and drag a copy into the Acquire case. Rename this copy of the Weather Station States type definition as `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.

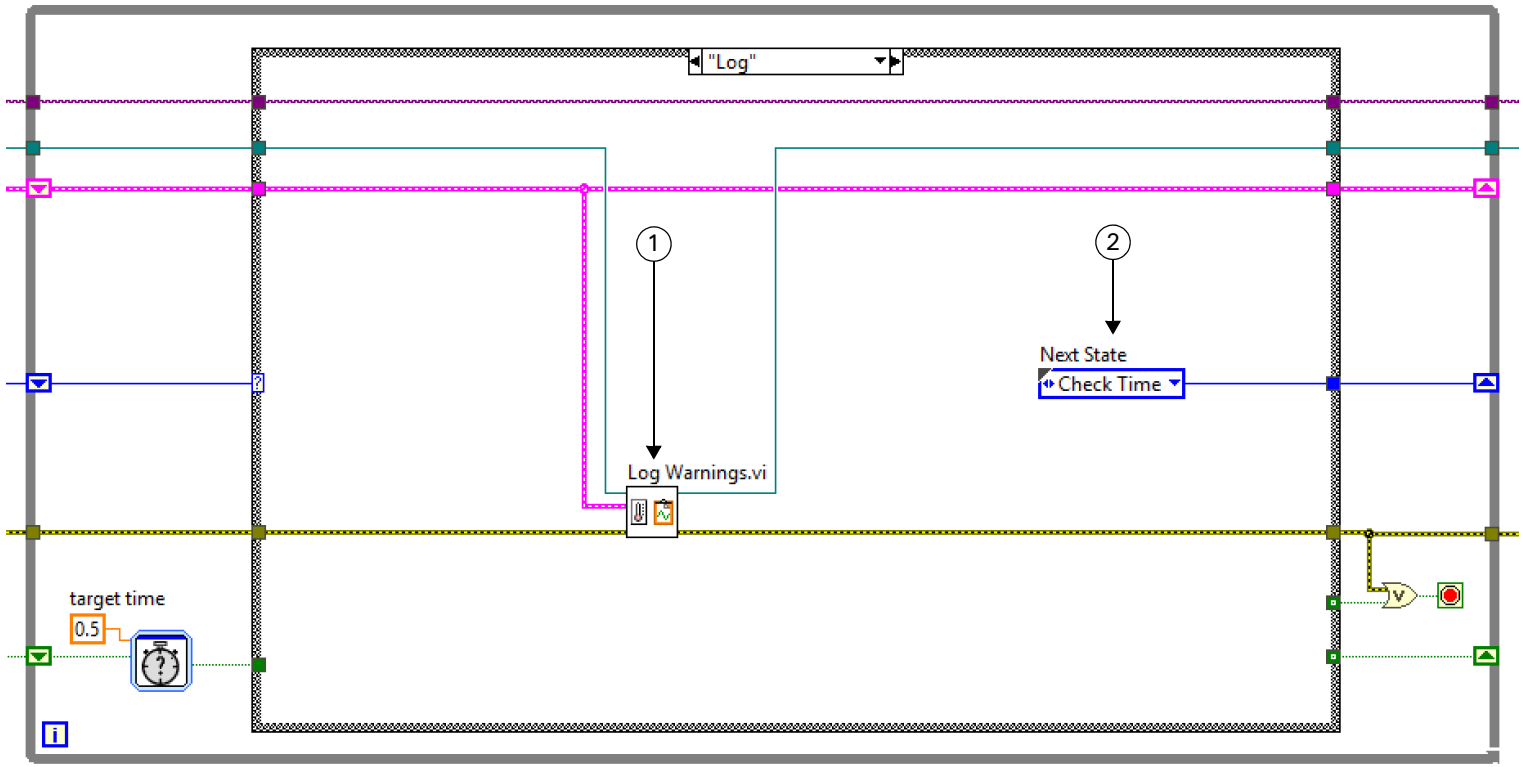
6 Wire the DAQmx task, file reference, **Weather Data** cluster, enum, error, and Boolean through the block diagram to their corresponding shift registers as shown.

8. Complete the Analyze case as shown in the following figure.



1 **Upper Limit and Lower Limit**—Move these controls into the Analyze state of the Case Structure.
2 **Bundle By Name**—Replaces the **Max Temperature** and **Min Temperature** items with the values from the **Upper Limit** and **Lower Limit** controls. The Bundle By Name function 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**—In the **Project Explorer** window, find Determine Warnings VI inside the Supporting Files folder and drag it to the block diagram.
4 **Unbundle By Name**—Returns the values 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** type definition enum renamed as `Next State` to the Select function. You can create these copies from the **Beginning State** enum.
7 **Warning**—Move this indicator into the Analyze state of the Case Structure.
8 Wire the DAQmx task and file reference through the case as shown.

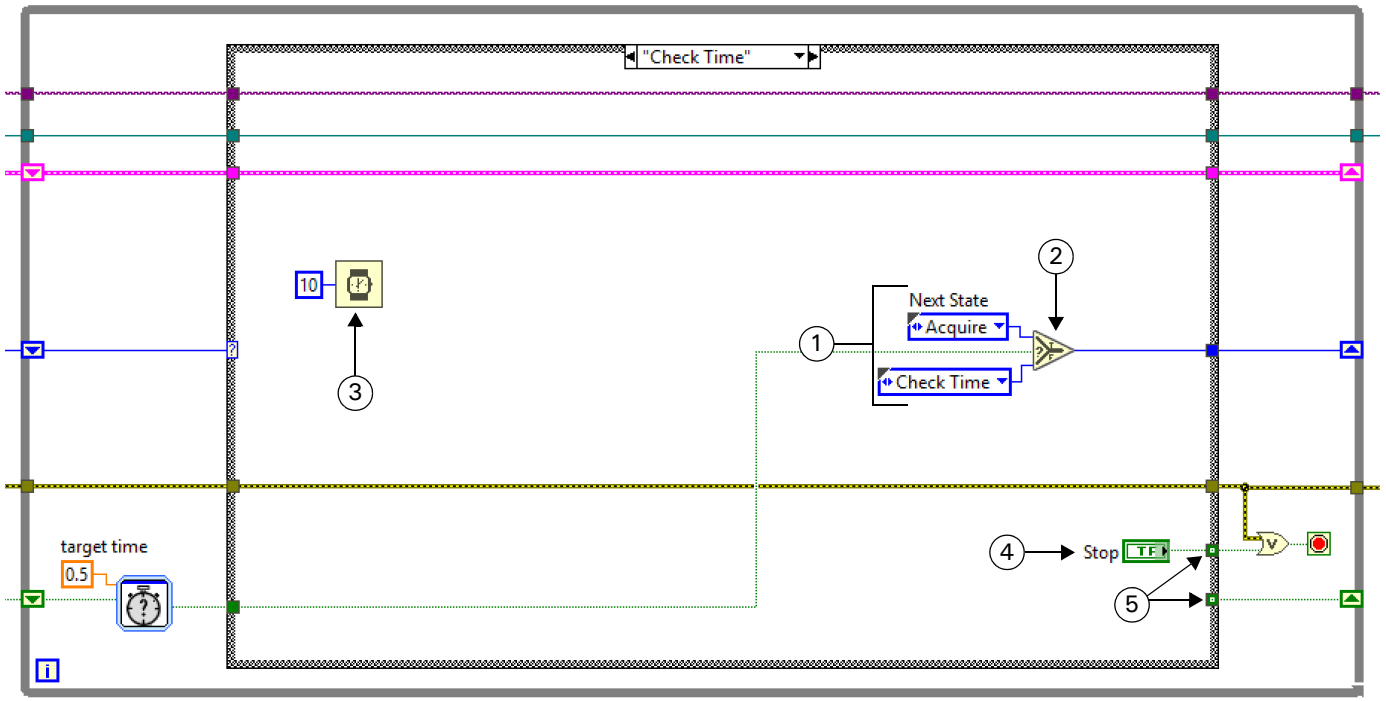
9. Complete the Log case as shown in the following figure.



1 **Log Warnings**—From the **Project Explorer** window, find Log Warnings VI inside the Supporting Files folder and drag it to the block diagram. This subVI logs weather data to file.

2 **Next State**—Create a copy of the **Weather Station States** type definition enum , label it `Next State`, and set the next state to **Check Time**.
Wire the DAQmx task and **Weather Data** cluster through the case as shown.

10. Complete the Check Time case as shown in the following figure.



1 **Next State**—Wire two copies of the **Weather Station States** type definition enum to the Select function.
2 **Select**—Determines which state to execute next depending on whether or not time has elapsed.
3 **Wait (ms)**—This function allows the CPU to do other tasks while this VI repeatedly executes the Check Time case until the Elapsed Timer VI returns a TRUE value when 0.5 seconds have elapsed.
4 **Stop**—Move the **Stop** button terminal from outside the While Loop. Wire the **Stop** button terminal to the Or function outside of the Case Structure.
5 **Use Default if unwired**—Right-click these tunnels and select **Use 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 through the case as shown.

11. Save and test the VI.

Test

1. Run the VI.
   - ☐ Name the log file `Weather Warning Log.txt` 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 code?

End of Exercise 14-1

A Advanced File I/O

In this lesson, we will take a closer look at TDMS files and binary files.

Topics

+ File Formats
+ Access TDMS Files in LabVIEW and Excel
+ Write and Read Binary Files

A. File Formats

You can view this entire lesson from the *Advanced File I/O* multimedia module, available in the `<Exercises>\...\Multimedia` folder.

Common Log File Formats

There are three common file types that you might work with in LabVIEW:

- Text (ASCII) files
- TDMS files
- Binary files

Text files are the easiest format to use and share, as many applications, such as notepad and Microsoft Excel, can read from or write to a text file. Data is represented as strings in text files.

TDMS is a binary file format created by National Instruments that is optimized for saving measurement data to file. You can use TDMS files to organize data and store information about your data. You can also use TDMS files to write and read data at high speeds. You can also use an add-in to view TDMS files in Microsoft Excel.

Binary is a file format that you can use when you want to write to or read from a binary file with a specific binary format.

Explore the table below to compare text, binary, and TDMS files.

	Text (ASCII)	Binary	TDMS
Easily Readable	X		
Small disk footprint		X	X
High-speed streaming		X	X
Searchable			X
Inherent attributes			X

B. Access TDMS Files in LabVIEW and Excel

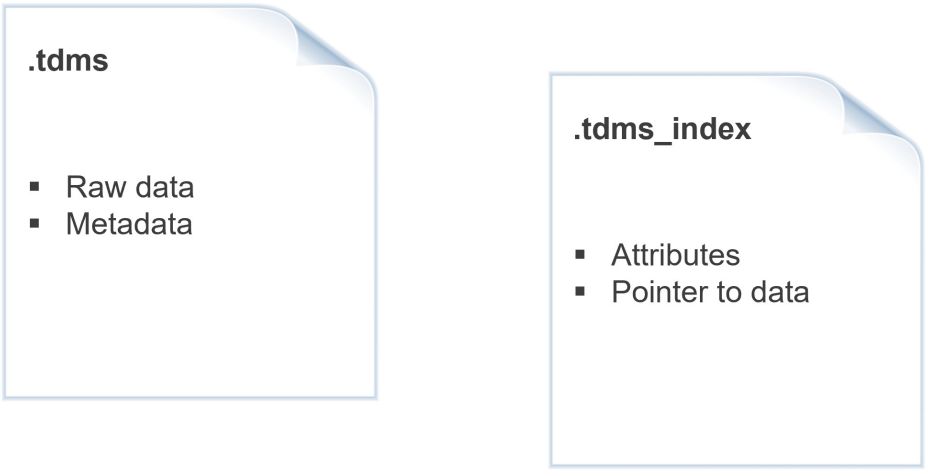
In this section, we will learn about TDMS files.

TDMS File Format

The TDMS file format is a binary format that was designed specifically for saving measurement data to file.

It contains two types of data—raw data and metadata.

You can use the TDMS file format to store measurement data along with channel-specific properties such as channel name, measurement units, test limits, and sensor information.



The TDMS index file is a binary file that provides consolidated information on all the attributes and pointers in the TDMS file. This speeds up access to data. The TDMS index file automatically regenerates if it is lost.

TDMS Files—Data Hierarchy and Properties

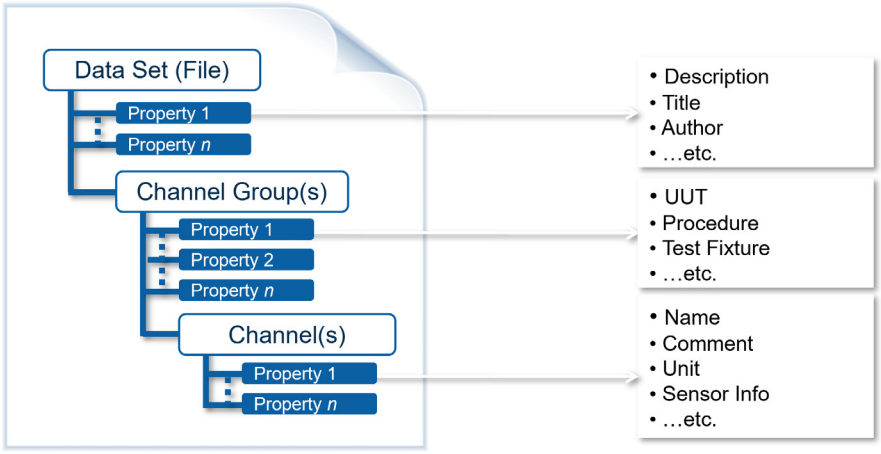
The TDMS file format is structured using three levels

- File
- Channel group
- Channel

A single TDMS file can contain multiple channel groups, and each channel group can contain multiple channels. You can choose how to organize your data to make it easier to understand.

For example, you may have one group for your raw data and another group for your analyzed data in a single file. Or, you might have multiple groups that correspond to sensor types or locations.

Additionally, at each level, you can store multiple properties to provide metadata. This helps you achieve well-documented and search-ready data files.

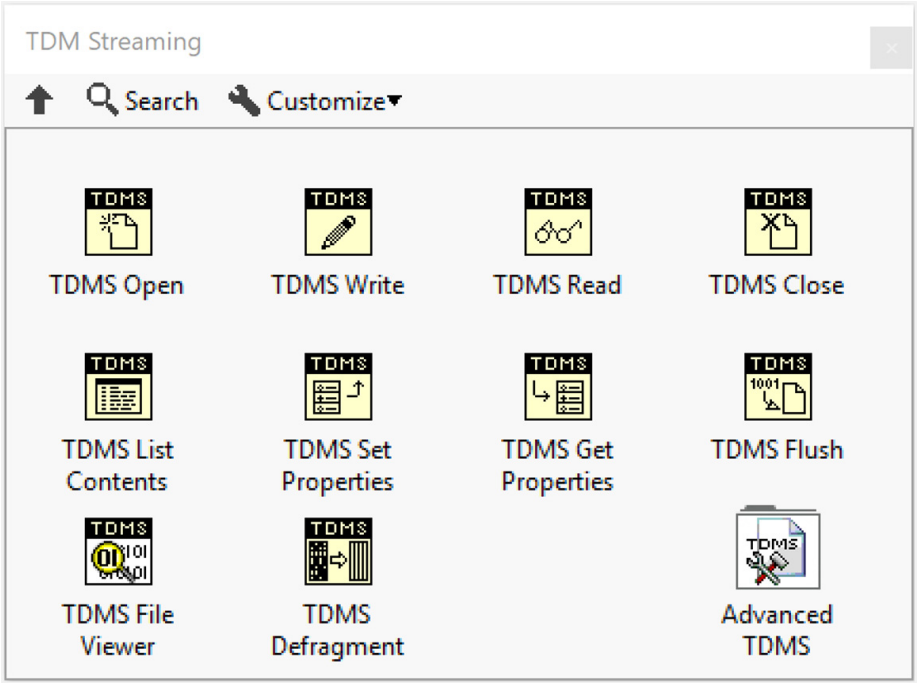


Channel A segment of a TDMS file that stores measurement signals or raw data as binary data. Each channel can have properties that describe the data.

Channel group A segment of a TDMS file that contains properties to store information as well as one or more channels. Use channel groups to organize your data and to store information that applies to multiple channels.

TDMS Functions

Use the TDMS functions to write to, read from, and search TDMS files.



Demonstration: Writing and Reading TDMS Files

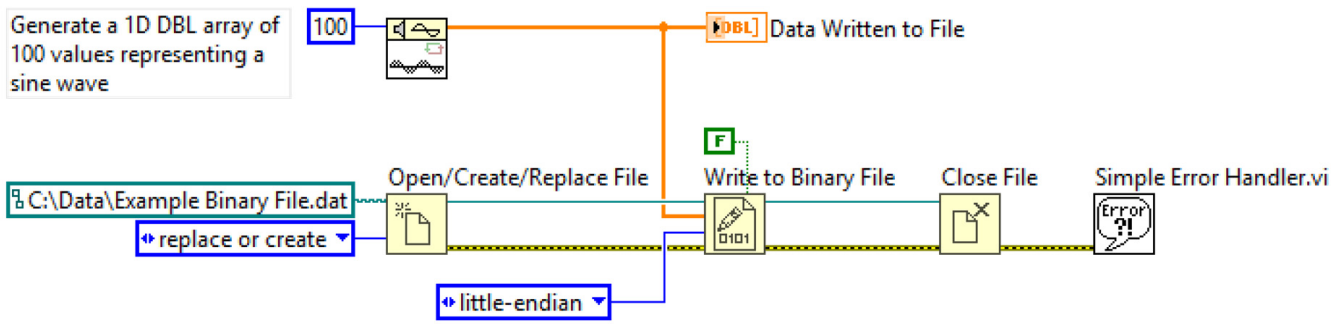
C. Write and Read Binary Files

In this section, we will briefly discuss writing and reading binary files.

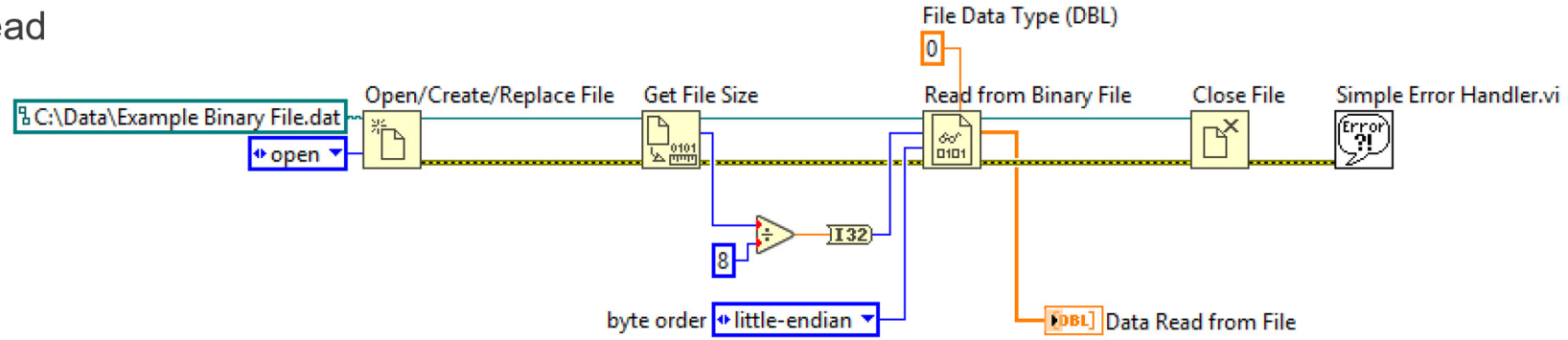
In the example shown in the figure below, the Writer VI passes a 1D double-precision array containing 100 values to the Write to Binary File function. Each DBL has a size of 64 bits which is 8 bytes. The binary file created by the Writer VI will contain 800 bytes because there are 100 DBLs and each DBL is 8 bytes.

To read a binary file correctly, the Reader must know exactly how the binary file organizes its data. Because we know that this binary file contains multiple DBLs next to each other, this VI uses the Get File Size to get the file size in bytes. In this case the file is 800 bytes. The Reader VI then divides the total number of bytes by the size of the DBL data type, which is 8 bytes, to get the total number of DBL numbers in this file.

Write



Read



Then, the Read from Binary File function is configured to interpret the read data as a DBL and is configured to read 100 DBLs from the file.

Binary File Considerations and Summary

- You should evaluate using text and TDMS files before considering using binary files.
- Compared to other formats, binary files are complex to work with and require an intimate knowledge of the file format.
- Debugging code can add considerable time to the development cycle if things go wrong.

B Additional Information and Resources

National Instruments provides global services and support as part of our commitment to your success. Take advantage of product services in addition to training and certification programs that meet your needs during each phase of the application life cycle; from planning and development through deployment and ongoing maintenance.

A. NI Services

To get started, register your product at ni.com/myproducts.

As a registered NI product user, you are entitled to the following benefits:

- Access to applicable product services.
- Easier product management with an online account.
- Receive critical part notifications, software updates, and service expirations.

Log in to your National Instruments ni.com User Profile to get personalized access to your services.

B. Services and Resources

- **Maintenance and Hardware Services**—NI helps you identify your systems' accuracy and reliability requirements and provides warranty, sparing, and calibration services to help you maintain accuracy and minimize downtime over the life of your system. Visit ni.com/services for more information.
  - **Warranty and Repair**—All NI hardware features a one-year standard warranty that is extendable up to five years. NI offers repair services performed in a timely manner by highly trained factory technicians using only original parts at a National Instruments service center.
  - **Calibration**—Through regular calibration, you can quantify and improve the measurement performance of an instrument. NI provides state-of-the-art calibration services. If your product supports calibration, you can obtain the calibration certificate for your product at ni.com/calibration.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit ni.com/alliance.
- **Training and Certification**—The NI training and certification program is the most effective way to increase application development proficiency and productivity. Visit ni.com/training for more information.
  - The Skills Guide assists you in identifying the proficiency requirements of your current application and gives you options for obtaining those skills consistent with your time and budget constraints and personal learning preferences. Visit ni.com/skills-guide to see these custom paths.
  - NI offers courses in several languages and formats including instructor-led classes at facilities worldwide, courses on-site at your facility, and online courses to serve your individual needs.
- **Technical Support**—Support at ni.com/support includes the following resources:
  - **Self-Help Technical Resources**—Visit ni.com/support for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on. Registered users also receive access to the NI Discussion Forums at ni.com/forums. NI Applications Engineers make sure every question submitted online receives an answer.
  - **Software Support Service Membership**—The Standard Service Program (SSP) is a renewable one-year subscription included with almost every NI software product, including NI Developer Suite. This program entitles members to direct access to NI Applications Engineers through phone and email for one-to-one technical support, as well as exclusive access to online training modules at

ni.com/self-paced-training. NI also offers flexible extended contract options that guarantee your SSP benefits are available without interruption for as long as you need them. Visit ni.com/ssp for more information.

- **Declaration of Conformity (DoC)**—A DoC is our claim of compliance with the Council of the European Communities using the manufacturer's declaration of conformity. This system affords the user protection for electromagnetic compatibility (EMC) and product safety. You can obtain the DoC for your product by visiting ni.com/certification.

For information about other technical support options in your area, visit ni.com/services, or contact your local office at ni.com/contact.

You also can visit the Worldwide Offices section of ni.com/niglobal to access the branch office websites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

C. Other National Instruments Training Courses

National Instruments offers several training courses for LabVIEW users. These courses continue the training you received here and expand it to other areas. Visit ni.com/training to purchase course materials or sign up for instructor-led, hands-on courses at locations around the world.

D. National Instruments Certification

Earning an NI certification acknowledges your expertise in working with NI products and technologies. The measurement and automation industry, your employer, clients, and peers recognize your NI certification credential as a symbol of the skills and knowledge you have gained through experience. Visit ni.com/training for more information about the NI certification program.