LabVIEW™ Core 1 Participant Guide

Course Software Version 2014
November 2014 Edition
Part Number 326292A-01

Copyright

End-User License Agreements and Third-Party Legal Notices

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

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

Trademarks

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

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

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

TETRIX by Pitsco is a trademark of Pitsco, Inc.

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

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

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

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

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

Xilinx is the registered trademark of Xilinx, Inc.

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

FireWire® is the registered trademark of Apple Inc.

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

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

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

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

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

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

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

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

Patents

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

Worldwide Technical Support and Product Information

ni.com

Worldwide Offices

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

National Instruments Corporate Headquarters

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

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

Table of Contents

Student Guide

Lesson 1
Navigating LabVIEW

Lesson 2
Creating Your First Application

Lesson 3
Troubleshooting and Debugging VIs

Lesson 4
Using Loops

Lesson 5
Creating and Leveraging Data Structures

Lesson 6
Using Decision-Making Structures

Lesson 7
Modularity

Lesson 8
Acquiring Measurements from Hardware

Lesson 9
Accessing Files in LabVIEW

Lesson 10
Using Sequential and State Machine Programming

Appendix A
Additional Information and Resources

Student Guide

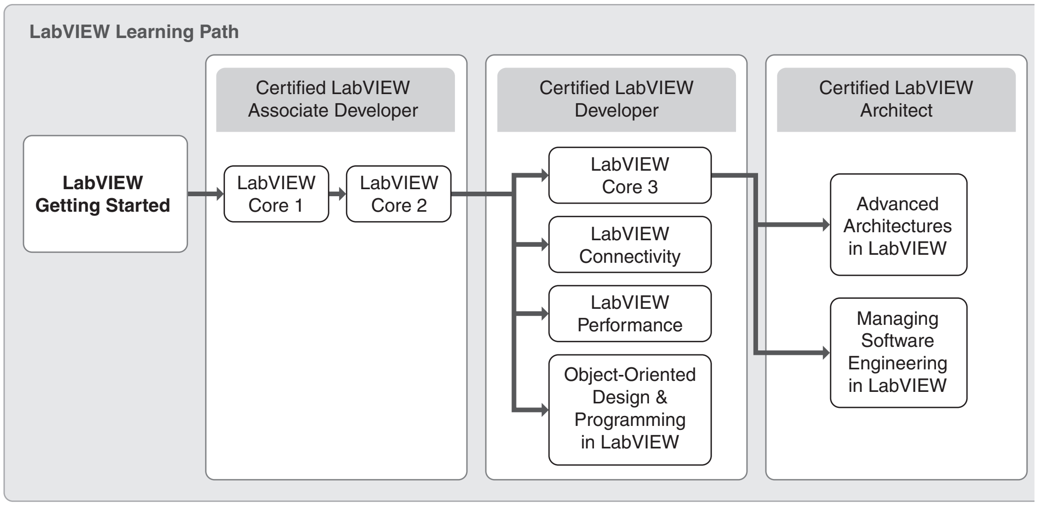
In this student guide, you will learn about the LabVIEW Learning Path, the course description, and the items you need to get started in the LabVIEW Core 1 course.

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.


LabVIEW Learning Path
Certified LabVIEW Associate Developer
Certified LabVIEW Developer
Certified LabVIEW Architect
LabVIEW Getting Started
LabVIEW Core 1
LabVIEW Core 2
LabVIEW Core 3
LabVIEW Connectivity
LabVIEW Performance
Object-Oriented Design & Programming in LabVIEW
Advanced Architectures in LabVIEW
Managing Software Engineering in LabVIEW

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.

Note For Participant Guide updates and corrections, refer to `ni.com/info` and enter the Info Code `core1`.

Several exercises use a plug-in multifunction data acquisition (DAQ) device connected to a DAQ Signal Accessory or BNC-2120 containing a temperature sensor, function generator, and LEDs.

If you do not have this 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/Vista/XP

□ Multifunction DAQ device configured as Dev1 using Measurement & Automation Explorer (MAX)

□ DAQ Signal Accessory or BNC-2120, wires, and cable

□ LabVIEW Professional Development System 2014 or later

□ DAQmx 14 or later

□ A GPIB cable

□ NI-488.2 14 or later

□ NI-VISA 14.0.1 or later

□ NI Instrument Simulator and power supply

□ NI Instrument Simulator Wizard installed from the NI Instrument Simulator software CD

□ *LabVIEW Core 1* course CD, from which you install the following folders:

Directory	Table Head
`Exercises`	Contains VIs used in the course
`Solutions`	Contains completed course exercises

D. Installing the Course Software

Complete the following steps to install the course software.

1. Insert the course CD in your computer. The **LabVIEW Core 1 Course Setup** dialog box appears.
2. Click **Install the course materials**.
3. Follow the onscreen instructions to complete installation and setup.

Exercise files are located in the `<Exercises>\LabVIEW Core 1\` folder.

Note Folder names in angle brackets, such as `<Exercises>`, refer to folders on the root directory of your computer.

E. Course Goals

This course prepares you to do the following:

- Understand front panels, block diagrams, icons, and connector panes
- Use the programming structures and data types that exist in LabVIEW
- Use various editing and debugging techniques
- Create and save VIs so you can use them as subVIs
- Display and log data
- Create applications that use plug-in DAQ devices
- Create applications that use serial port and GPIB instruments

This course does *not* describe the following:

- Every built-in VI, function, or object; refer to the *LabVIEW Help* for more information about LabVIEW features not described in this course
- Analog-to-digital (A/D) theory
- Operation of the GPIB bus
- Developing an instrument driver
- Developing a complete application for any student in the class; refer to the NI Example Finder, available by selecting **Help»Find Examples**, for example VIs you can use and incorporate into VIs you create

1 Navigating LabVIEW

In this lesson you will learn to recognize the main components of the LabVIEW environment and be able to create a new project and VI.

Topics

- What is LabVIEW?
- Project Explorer
- Parts of a VI
- Front Panel
- Block Diagram
- Searching for Controls, VIs, and Functions

Exercises

Exercise 1-1 Concept: Exploring a VI

Exercise 1-2 Concept: Locating Controls, Functions, and VIs

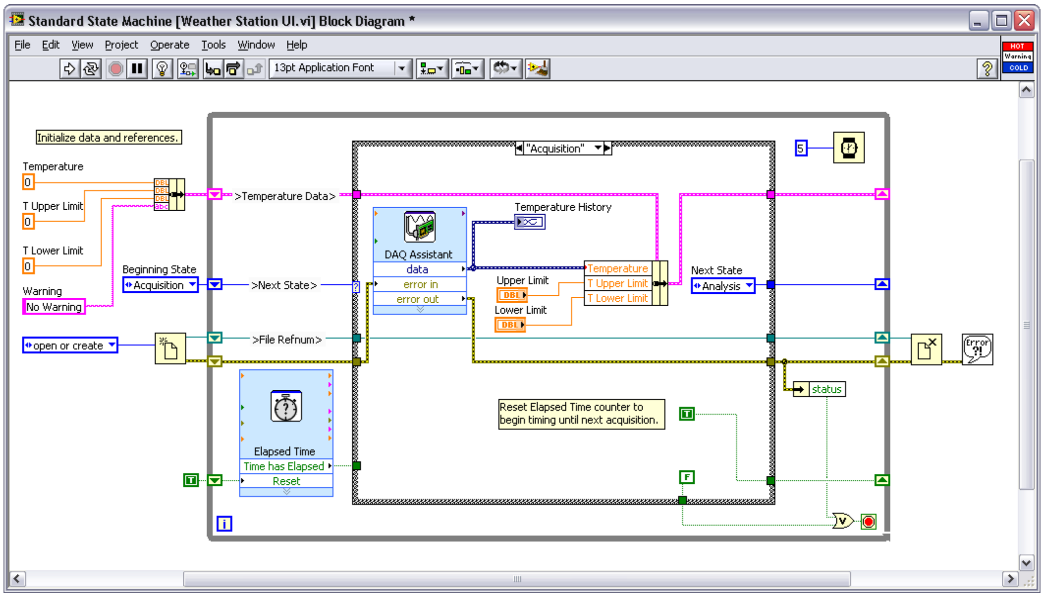
A. What is LabVIEW?

Objective: Recognize benefits of LabVIEW.

LabVIEW is a graphical programming environment you can used to develop sophisticated measurement, test, and control systems.

Benefits of LabVIEW

- A platform-based approach for measurement and control
- Interfaces with wide variety of hardware
- Scales across different targets and operating systems
- Provides built-in analysis libraries



Reset Elapsed Time counter to begin timing until next acquisition.

LabVIEW Language Characteristics

LabVIEW Core 1 Topics	LabVIEW Core 2 and Later Topics
Graphical Uses icons and wires instead of with text.	Multi-threaded Enables your code to have automatic parallelism.
Dataflow-oriented Data-driven, or data-dependent. The flow of data between nodes in the G code determines the execution order.	Object-oriented Allows a variety of similar, yet different items, to be represented as a class of objects in software.
Compiled Compiles G code directly to machine code.	Multi-target Targets multicore processors and other parallel hardware such as field-programmable gate arrays (FPGAs).

LabVIEW Core 1 Topics	LabVIEW Core 2 and Later Topics
Multi-platform Is portable between LabVIEW on different operating systems.	Memory-managed Provides you with options for improving performance.
Synchronous	
Event-driven Allows the user's direct interaction with the program without the need for polling.	

Activity 1-1: Confidence Activity—Building a VI

Goal

Build an application to calculate the product of two numbers.

B. Project Explorer

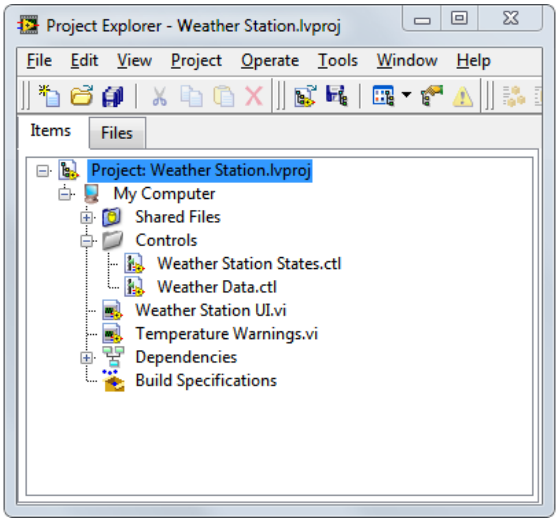
Objective: Recognize the main components of the Project Explorer.

Project Explorer Projects in LabVIEW consist of VIs, files necessary for those VIs to run properly, and supplemental files such as documentation or related links. Use the **Project Explorer** window to manage projects in LabVIEW. A LabVIEW project is denoted by a project file (`.lvproj`)

LabVIEW Files

Listed below are some of the characteristics of a LabVIEW project.

- Manages project files
- Deploys programs to remote embedded targets such as NI CompactRIO
- Creates stand-alone executables and installers for distribution
- Makes it possible to distribute code to developers and create shared libraries
- Can integrate LabVIEW programs with source code control software such as Perforce

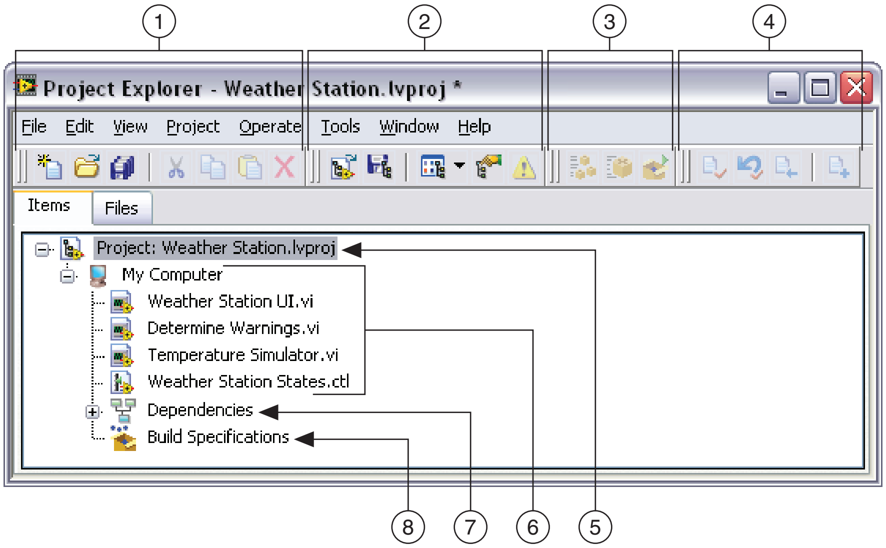
Common LabVIEW File Extensions	
 Project Explorer - Weather Station.lvproj File Edit View Project Operate Tools Window Help Items Files Project: Weather Station.lvproj My Computer Shared Files Controls Weather Station States.ctl Weather Data.ctl Weather Station UI.vi Temperature Warnings.vi Dependencies Build Specifications	LabVIEW Project — .lvproj
	Virtual Instrument (VI) — .vi
	Custom Control — .ctl

Demonstration: Using the Project Explorer and Starting a VI

The **Project Explorer** window includes the following items by default:

- **Project root**—Contains all other items in the **Project Explorer** window. The label on the project root includes the filename for the project.
- **My Computer**—Represents the local computer as a target in the project.
- **Dependencies**—Includes VIs and items that VIs under a target require.
- **Build Specifications**—Includes build configurations for source distributions and other types of builds available in LabVIEW toolkits and modules. If you have the LabVIEW Professional Development System or Application Builder installed, you can use **Build Specifications** to configure stand-alone applications, shared libraries, installers, and zip files.

When you add another target to the project, LabVIEW creates an additional item in the **Project Explorer** window to represent the target. Each target also includes **Dependencies** and **Build Specifications** sections. You can add files under each target.



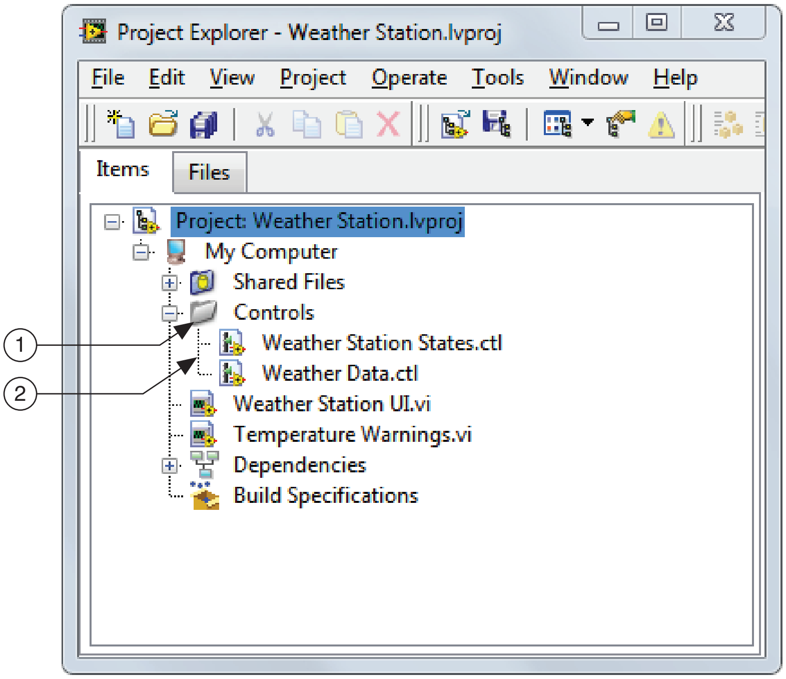
1	Standard Toolbar	5	Project Root
2	Project Toolbar	6	Target
3	Build Toolbar	7	Dependencies
4	Source Control Toolbar	8	Build Specifications

Tip The **Source Control** toolbar is only available if you have source control configured in LabVIEW.

Adding Folders to a Project

Add folders to a project to create an organizational structure for project items.

Auto-Populated Folders	Virtual Folders
📁	📁
Adds a directory on disk to the project	Organizes project items and does not represent files on disk
LabVIEW continuously monitors and updates the folder according to changes made in the project and on disk	Can convert a virtual folder to an auto-populated folder.


Project Explorer - Weather Station.lvproj
File Edit View Project Operate Tools Window Help
Items
Files
Project: Weather Station.lvproj
My Computer
Shared Files
Controls
Weather Station States.ctl
Weather Data.ctl
Weather Station UI.vi
Temperature Warnings.vi
Dependencies
Build Specifications
1
2

1 Auto-populating folder	2 Virtual folder

C. Parts of a VI

Objective: Recognize and understand the difference between the front panel and block diagram.

Demonstration: Components of a VI

LabVIEW VIs contain three main components—the front panel window, the block diagram, and the icon/connector pane.

Front Panel	Block Diagram	Icon/Connector Pane
Is the user interface.	Contains the graphical source code.	Represents the VI and makes it possible to use the VI as a subVI.
Has controls (inputs) and indicators (outputs).	Contains terminals for front panel controls and indicators.	Maps the inputs and outputs to the VI.

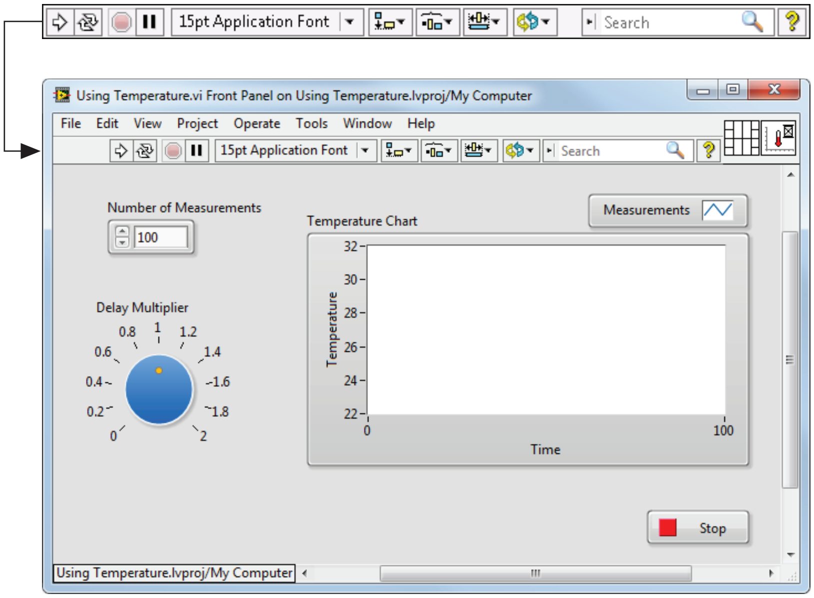
D. Front Panel

Objective: Recognize the components and functionality of the Front Panel window and select appropriate controls and indicators

Front Panel User interface for the VI. Contains controls and indicators, which are the interactive input and output terminals of the VI, respectively.

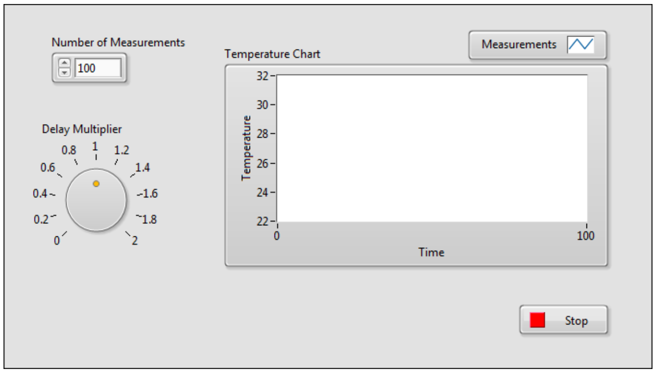
Front Panel Window Toolbar

Use the front panel window toolbar buttons to run and edit the VI.



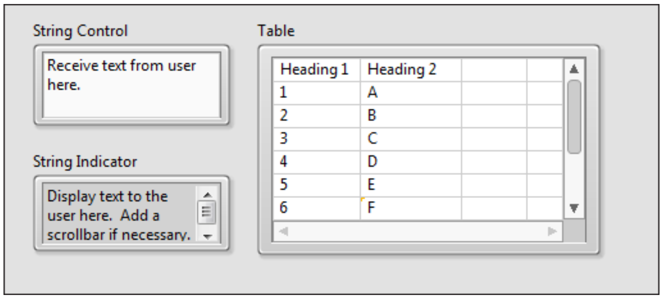
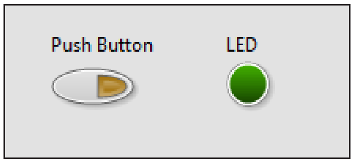
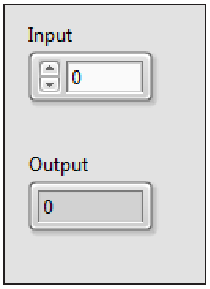
Controls and Indicators

Controls	Indicators
Interactive input	Interactive output
Knobs, push buttons, dials, and other inputs	Graphs, LEDs, and other displays
Simulate instrument input devices and supply data to the block diagram	Simulate instrument output devices and display data the block diagram acquires or generates



String, Boolean, and Numeric Data Types

The following table shows controls and indicators of the string, Boolean, and numeric data types.

 String Control Receive text from user here. String Indicator Display text to the user here. Add a scrollbar if necessary. Table Heading 1 Heading 2 1 A 2 B 3 C 4 D 5 E 6 F	The string data type is a sequence of ASCII characters. Use string controls to receive text from the user such as a password or user name. Use string indicators to display text to the user.
	The Boolean data type represents data that only has two possible states, such as TRUE and FALSE or ON and OFF. Use Boolean controls and indicators to enter and display Boolean values.
	The numeric data type can represent numbers of various types, such as integer or real. The two common numeric objects are the numeric control and the numeric indicator.

Activity 1-2: Select Front Panel Objects

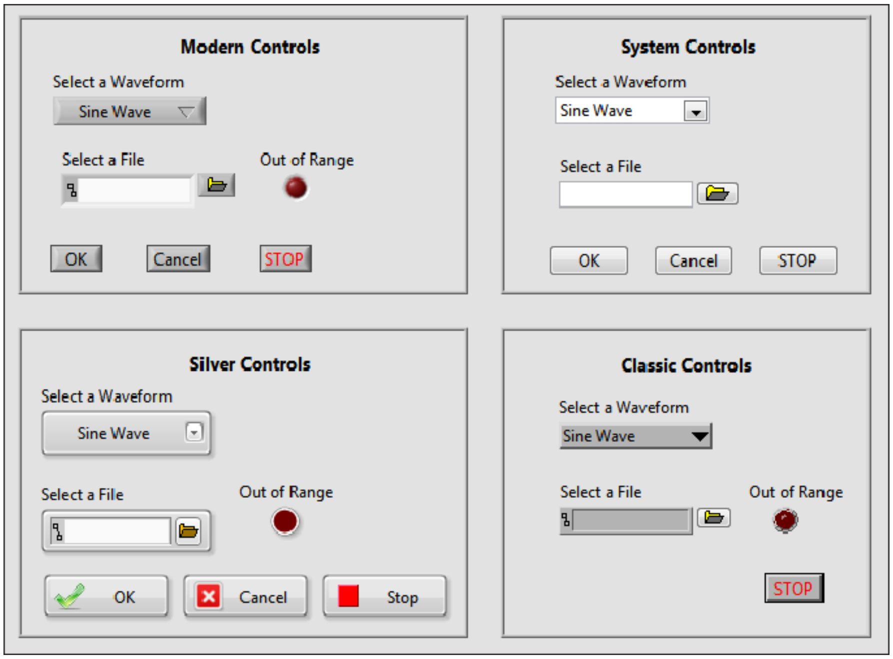
Goal

For each scenario, determine the appropriate data type and whether the front panel object should be a control or indicator.

Scenario	Data Type	Control/Indicator
Display the temperature of a room	**Numeric**	**Indicator**
An Emergency stop button to stop a process		
Username and password to login to your bank account		
An LED to display error status		

Front Panel Object Styles

LabVIEW has different control palettes with objects for building user interfaces. These palettes include the Modern, Silver, Classic, and Systems palettes.



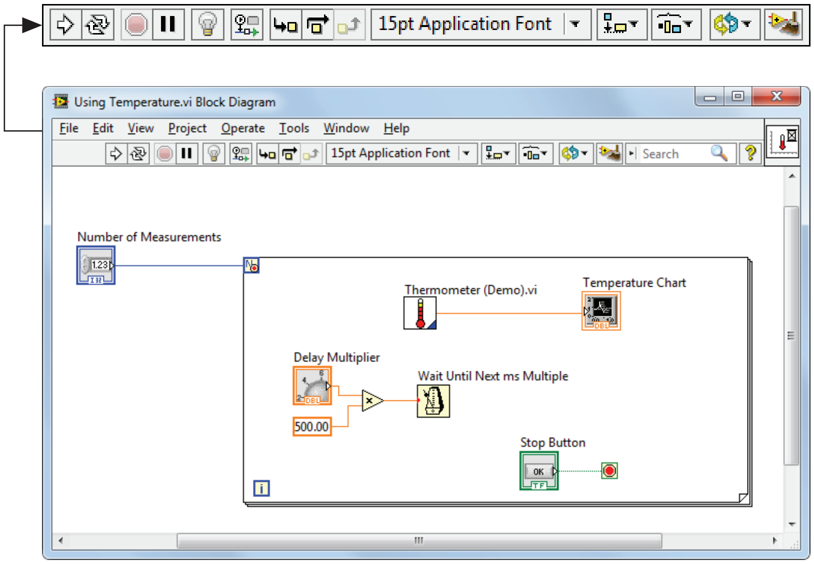
E. Block Diagram

Objective: Recognize features of the Block Diagram and be able to select functions.

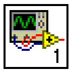
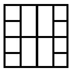
Block Diagram Block diagram objects include terminals, subVIs, functions, constants, structures, and wires, which transfer data among other block diagram objects.

Block Diagram Toolbar

When you run a VI, buttons appear on the block diagram toolbar that you can use to debug the VI.

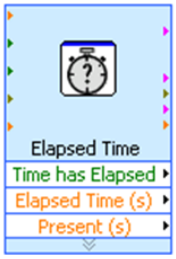


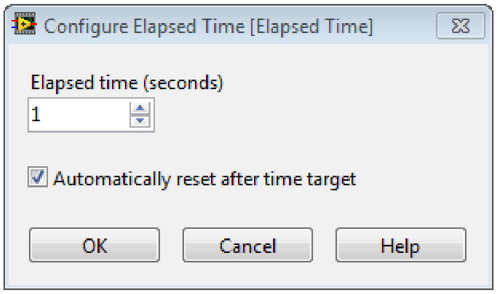
Demonstration: Components of a Block Diagram

Component	Description
Terminals	Terminals are the entry and exit ports that exchange information between the front panel and block diagram. You can view terminals as icon conserve space.
Nodes	Nodes are objects on the block diagram that have inputs and/or outputs and perform operations when a VI runs. Nodes are analogous to statements, operators, functions, and subroutines in text-based programming languages. Nodes can be functions, subVIs, or structures.
Function Nodes (Functions)	Functions are fundamental operating elements of LabVIEW and do not have front panels or block diagrams. Function icons have pale yellow backgrounds.
SubVIs	SubVIs are VIs that run on the block diagram of another VI. They have front panels and block diagrams. Any VI has the potential to be used as a subVI. When you double-click a subVI, the front panel and block diagram open.
Icon	Icons are the graphical representations of VIs.  Use the icon from the upper-right corner of the front panel as the icon that appears when you place the subVI on a block diagram.
Connector Pane	The connector pane is the map of inputs and outputs of a VI  The connector pane terminals correspond to the controls and indicators on the front panel of the VI. You cannot access the connector pane from the block diagram window.

Express VIs

Express VIs are a type of subVI that you configure with dialog boxes. Icons for Express VIs appear on the block diagram as icons surrounded by a blue field.





Configure Elapsed Time [Elapsed Time]

Elapsed time (seconds)

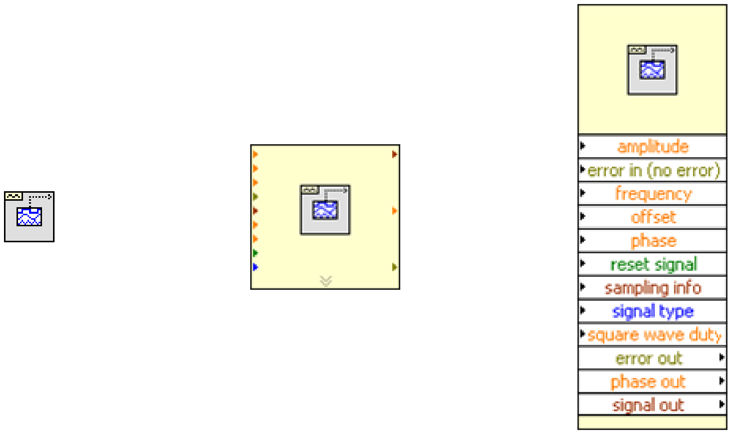
1

☑ Automatically reset after time target

Nodes Appearance

You can display VIs and Express VIs as either icons or expandable nodes. Use icons if you want to conserve space on the block diagram. By default, subVIs appear as icons on the block diagram, and Express VIs appear as expandable nodes.

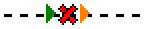
To display a subVI or Express VI as an expandable node, right-click the subVI or Express VI and remove the checkmark next to the **View As Icon** shortcut menu item.



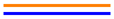
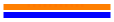
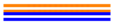
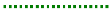
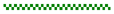
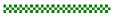
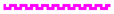
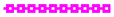
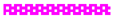
1 Icon 2 Expandable Node 3 Expanded Node

Wires

Wires transfer data between block diagram objects. A broken wire appears as a dashed black line with a red X in the middle. You cannot run a VI that contains a broken wire.



Broken wires occur for a variety of reasons, such as when you try to wire two objects with incompatible data types.

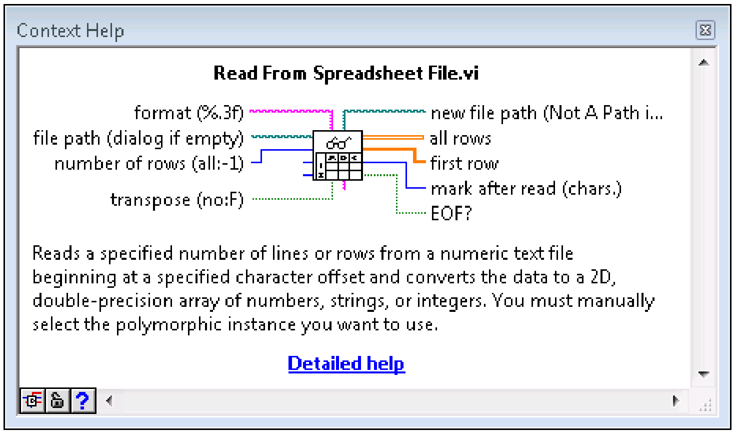
Common Wire Types				
Wire Type	**Scalar**	**1D Array**	**2D Array**	**Color**
Numeric				Orange (floating-point), Blue (integer)
Boolean				Green
String				Pink

Demonstration: Context Help and LabVIEW Help

Use the **Context Help** window and the *LabVIEW Help* to help you create and edit VIs.

Context Help Window

The **Context Help** window displays basic information about LabVIEW objects when you move the cursor over each object.



Context Help

Read From Spreadsheet File.vi

Reads a specified number of lines or rows from a numeric text file beginning at a specified character offset and converts the data to a 2D, double-precision array of numbers, strings, or integers. You must manually select the polymorphic instance you want to use.

Detailed help

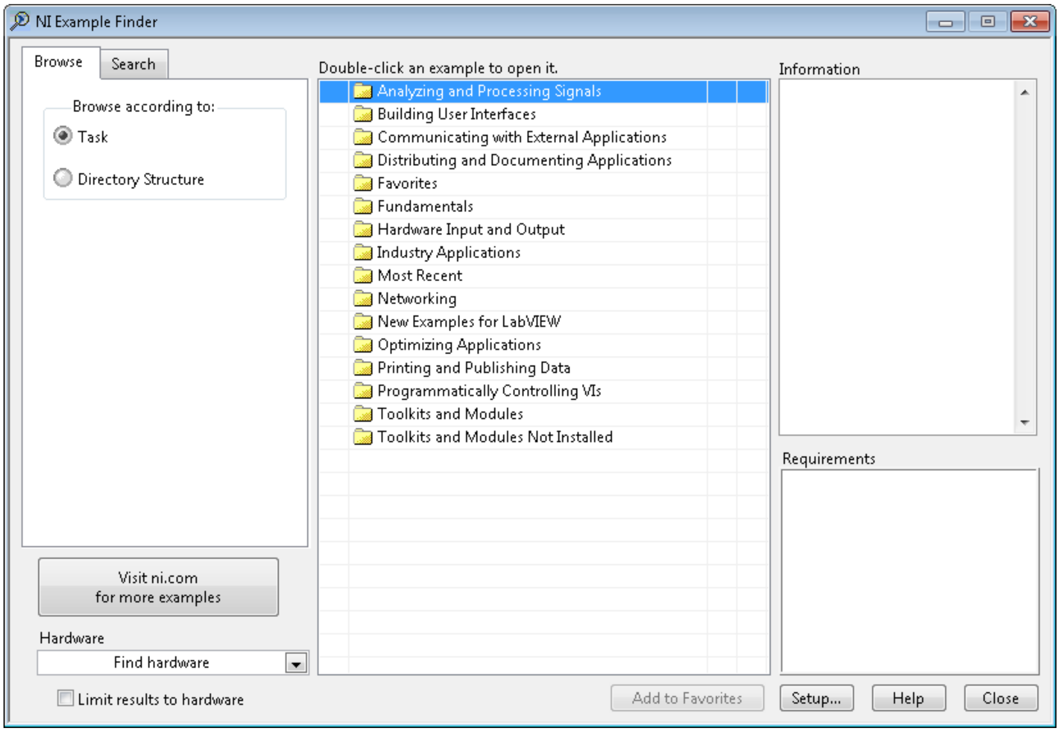
To toggle display of the **Context Help** window select **Help»Show Context Help**, press the <Ctrl-H> keys, or click the **Show Context Help Window** button on the toolbar.

LabVIEW Help

You can access the *LabVIEW Help* by clicking the **More Help** button in the **Context Help** window, selecting **Help»LabVIEW Help**, or clicking the blue **Detailed Help** link in the **Context Help** window. You also can right-click an object and select **Help** from the shortcut menu.

Examples

Use the NI Example Finder to browse or search examples installed on your computer or on the NI Developer Zone at `ni.com/zone`. These examples demonstrate how to use LabVIEW to perform a wide variety of test, measurement, control, and design tasks. Select **Help»Find Examples** to launch the NI Example Finder.


NI Example Finder
Browse
Search
Double-click an example to open it.
Information
Analyzing and Processing Signals
Building User Interfaces
Communicating with External Applications
Distributing and Documenting Applications
Favorites
Fundamentals
Hardware Input and Output
Industry Applications
Most Recent
Networking
New Examples for LabVIEW
Optimizing Applications
Printing and Publishing Data
Programmatically Controlling VIs
Toolkits and Modules
Toolkits and Modules Not Installed
Requirements
Visit ni.com for more examples
Hardware
Find hardware
Add to Favorites
Setup...
Help
Close

Browse according to:
● Task
○ Directory Structure

□ Limit results to hardware

Exercise 1-1 Concept: Exploring a VI

Goal

As a class, identify the parts of an existing VI.

Description

You received a VI from an employee that takes the seconds until a plane arrives at an airport and converts the time into a combination of hours/minutes/seconds. You must evaluate this VI to see if it works as expected and can display the remaining time until the plane arrives.

1. Open `Flight Delays.lvproj` in the `<Exercises>\LabVIEW Core 1\Exploring A VI` directory.

2. Open **Seconds Breakdown.vi** from the **Project Explorer** window.

3. On the front panel, identify the following items. How many can you find of each item?

   □ Controls

   □ Indicators

   □ Free labels

   □ Run button

   □ Icon

   □ Connector pane

4. Press <Ctrl-T> to view the front panel and block diagram at the same time or select **Window»Tile Up and Down** or **Window»Tile Left and Right**.

Tip To switch between the front panel window and the block diagram without tiling the windows, press <Ctrl-E>.

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

   □ Controls

   □ Indicators

   □ Constants

   □ Free labels

6. 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 block diagram.

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

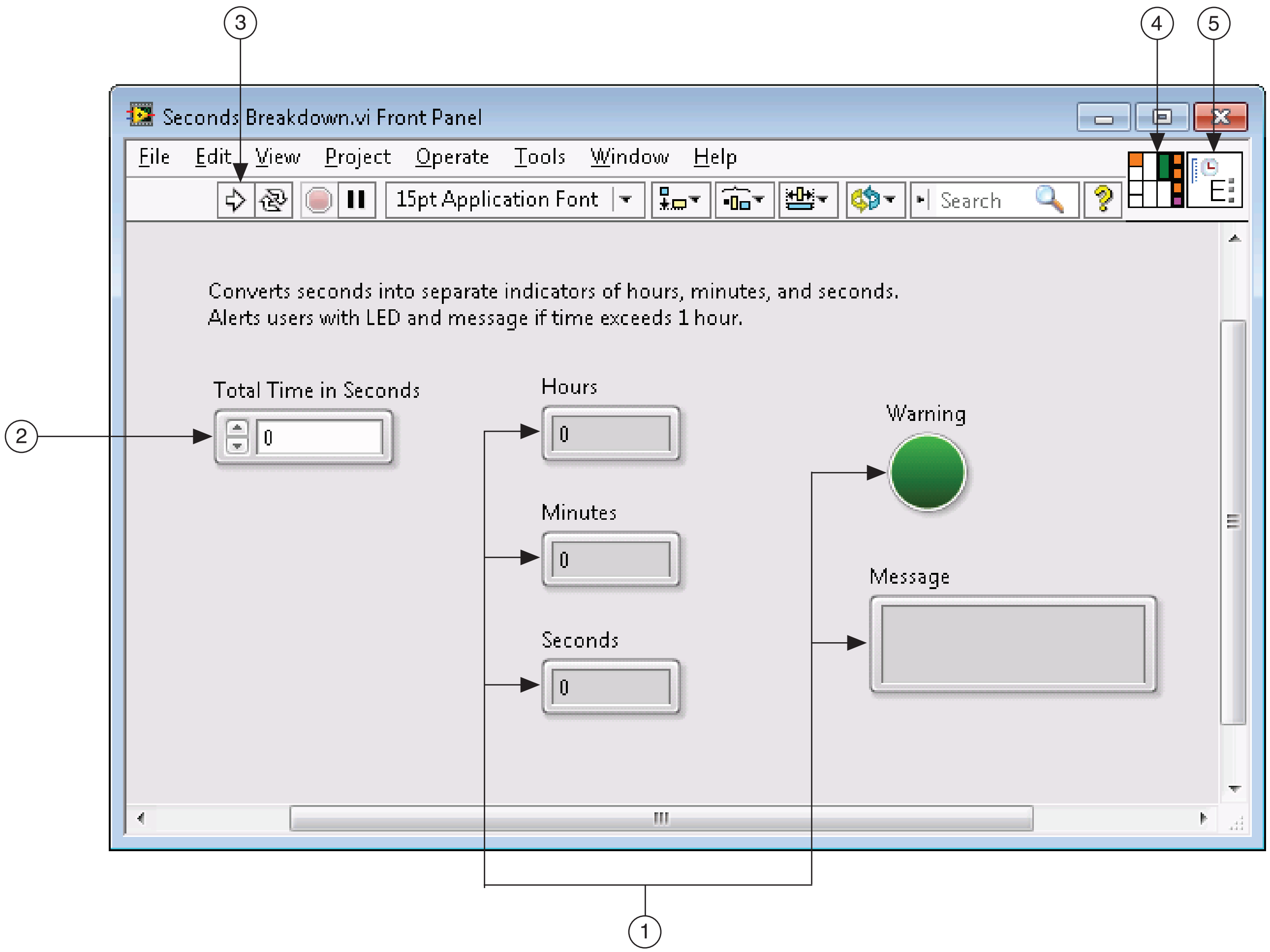
   □ The **Context Help** window content changes to show information about the object that your mouse is over.

7. Get detailed help for the Quotient & Remainder function.

   □ Place your cursor over the Quotient & Remainder function. Read the **Context Help** window and click the **Detailed Help** link to launch the *LabVIEW Help* and learn more about this function.

8. Refer to Figures 1-1 and 1-2 to verify that you identified all items correctly.

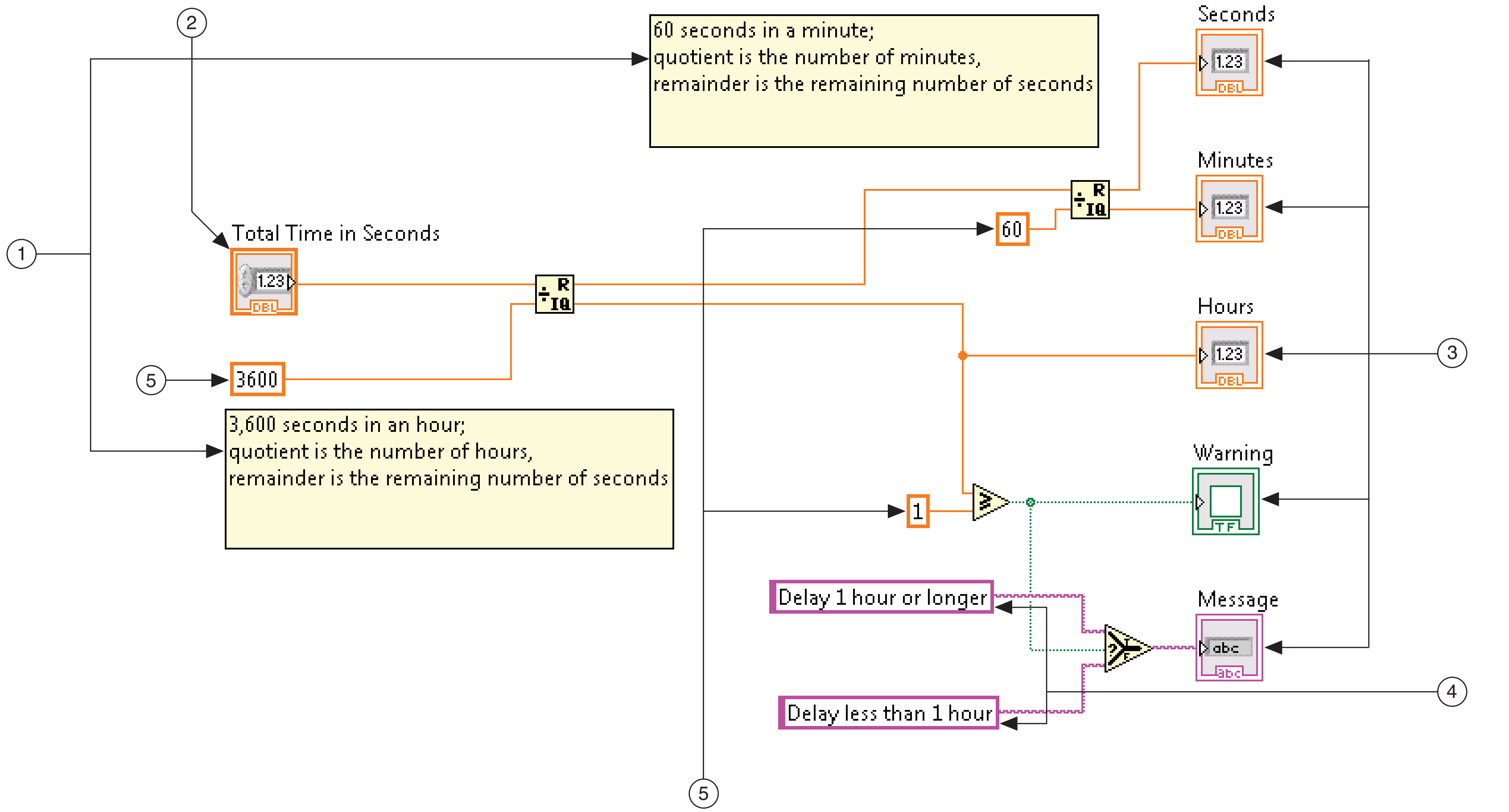
Figure 1-1. Front Panel Items



Converts seconds into separate indicators of hours, minutes, and seconds.
Alerts users with LED and message if time exceeds 1 hour.

1	Indicators	2	Control	3	Run Button	4	Connector Pane	5	Icon

Figure 1-2. Block Diagram Items


60 seconds in a minute;
quotient is the number of minutes,
remainder is the remaining number of seconds
Total Time in Seconds
3600
3,600 seconds in an hour;
quotient is the number of hours,
remainder is the remaining number of seconds
60
1
Seconds
Minutes
Hours
Warning
Message
Delay 1 hour or longer
Delay less than 1 hour
1.23
DBL
TF
abc
1
2
3
4
5
5

1	Free Labels	2	Control	3	Indicators	4	String Constants	5	Numeric Constants

9. Test the Seconds Breakdown VI using the values given in Table 1-1.

   □ Enter an input value in the **Total Time in Seconds** control.

   □ Click the **Run** button.

   □ For each input, compare the given outputs to the outputs listed in Table 1-1. If the VI works correctly, they should match.

Table 1-1. Testing Values for Seconds Breakdown.vi

Input	Numeric Indicators	LED Indicator	String Indicator
0 seconds	0 hours, 0 minutes, 0 seconds	Off	**`Delay less than 1 hour`**
60 seconds	0 hours, 1 minute, 0 seconds	Off	**`Delay less than 1 hour`**
3600 seconds	1 hour, 0 minutes, 0 seconds	On	**`Delay 1 hour or longer`**
3665 seconds	1 hour, 1 minute, 5 seconds	On	**`Delay 1 hour or longer`**

10. Save and close the VI and the LabVIEW Project.

End of Exercise 1-1

F. Searching for Controls, VIs, and Functions

Objective: Find the controls, indicators, and functions you need.

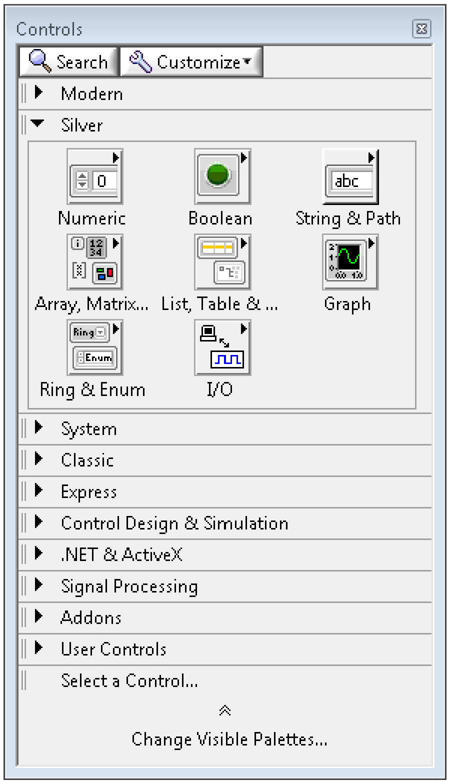
Demonstration: Searching For Controls, VIs, and Functions

Controls Palette

The **Controls** palette contains the controls and indicators you use to create the front panel. Select **View»Controls Palette** to display it.

Navigate the subpalettes or use the **Search** button to search the Controls palette.

Figure 1-3. Controls Palette

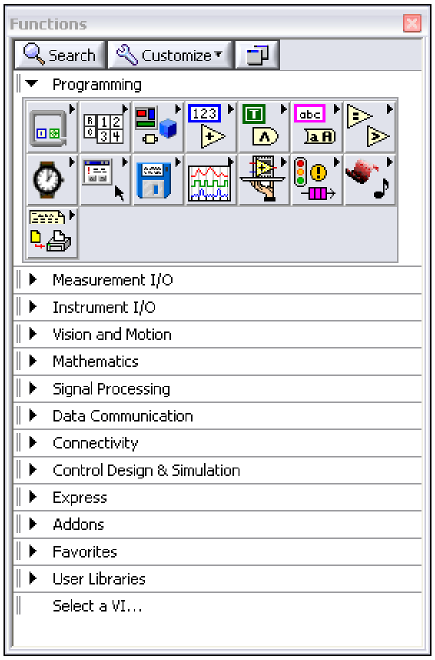


Functions Palette

The **Functions** palette contains the VIs, functions and constants you use to create the block diagram. Select **View»Functions Palette** to display it.

Navigate the subpalettes or use the **Search** button to search the Functions palette.

Figure 1-4. Functions Palette



Quick Drop

The Quick Drop dialog box lets you quickly find controls, functions, VIs, and other items by name. Press the <Ctrl-Space> keys to display the Quick Drop dialog box.

Global Search

Use the Search bar in the top right of the front panel and block diagram windows to search palettes, *LabVIEW Help*, and `ni.com`.

Exercise 1-2 Concept: Locating Controls, Functions, and VIs

Goal

Learn to use the palettes and search for controls, functions, and VIs.

Description

1. Open a blank LabVIEW project.

   □ Click the **Create Project** button in the LabVIEW **Getting Started** window and then click **Blank Project**.

   □ Click **Finish**.

2. Create a blank VI and add it to the project.

   □ Right-click **My Computer** in the **Project Explorer** window and select **New»VI** from the shortcut menu.

3. Select **View»Controls Palette** from the menu of the VI front panel window.

4. Customize the **Controls** palette.

   □ Click the **Customize** button and select **Change Visible Palettes**.

   □ Select the following palettes to add them to the **Controls** palette and click the **OK** button. Do not deselect any palettes.

   – **Silver**

   – **Control & Simulation**

   – **Signal Processing**

   □ Notice that the three palettes you just selected now appear in the **Controls** window.

5. Explore the **Controls** palette.

   Use palettes to locate controls and functions when you want to explore the options available to you or when you are not sure of the name of the control or function you need.

   □ Click the **Search** button.

   □ Type `string control` in the search text box.

   □ Click **String Control (Silver)** in the search results and drag it to the front panel window to place the object.

6. Open the block diagram and right-click anywhere on the block diagram to display the **Functions** palette.

   □ Click the pin in the upper left-hand corner to keep the palette open.

   Tip You can customize the **Functions** palette just like you customized the **Controls** palette.

7. Explore the **Functions** palette.

   □ Locate trigonometric functions.

   – Click the **Search** button.

   – Search for the term `cosine`.

   – In the search results, double-click **Cosine <<Trigonometric Functions>>** to display the function on the palette.

   □ Locate file I/O functions.

   – Search for the term `file i/o`.

   – Double-click **File I/O** in the search results to display the **File I/O** palette.

   – Drag the Write To Text File function from the palette to the block diagram.

8. Practice using the Quick Drop feature.

   Use the Quick Drop feature when you know the name of the function or VI you want to use.

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

   □ Type `Bundle By Name` and double-click **Bundle By Name** in the search results. The cursor changes to a hand with the Bundle By Name function.

   □ Click on the block diagram to place the Bundle By Name function.

   □ Open the **Quick Drop** dialog box again.

   □ Search for the `Wait Until Next ms Multiple`.

   □ Double-click the function in the search results and place the function on the block diagram.

9. Practice using the global search feature.

   □ Type `Random` in the Search bar in the upper right hand corner of the block diagram.

   Note As you type, the global search automatically looks for matches in the *LabVIEW Help* and LabVIEW palettes. It also searches for online material related to your query.

□ Hover the mouse over the first result in the **Palette** section, **Random Number (0-1)**. You now see the following three options:

- **Drop**—Allows you to place this function immediately on the block diagram
- **Find**—Locates the function on the **Functions** palette
- **Help**—Brings up the help topic for this function.

□ Click each of these options to observe the different behaviors.

10. Practice accessing similar functions.

□ Place an Add function on the block diagram.

□ Right-click the Add function and notice that **Numeric palette** is available from the shortcut menu.

□ Practice placing functions from the **Numeric** palette on the block diagram.

11. Close the VI and LabVIEW project. You do not need save the files.

End of Exercise 1-2

Additional Resources

Learn More About	LabVIEW Help Topic
Project Explorer	*Creating a Project*
	Managing a Project in LabVIEW
Front Panel	*Building the Front Panel*

Activity 1-3: Lesson Review

Answer the following questions and then discuss your answers with the class.

1. What are the three parts of a VI?
   a. Front panel
   b. Block diagram
   c. Project
   d. Icon/Connector pane

Activity 1-3: Lesson Review - Answers

1. What are the three parts of a VI?
    a. **Front panel**
    b. **Block diagram**
    c. Project
    d. **Icon/Connector pane**

2 Creating Your First Application

In this lesson you will learn to use Express VIs to produce a project and create a simple VI that acquires and analyzes data and then displays the results.

Topics

+ Dataflow
+ LabVIEW Data Types
+ Tools for Programming, Cleaning and Organizing Your VI
+ Building a Basic VI

Exercises

Exercise 2-1 Selecting a Tool

Exercise 2-2 Simple AAV VI

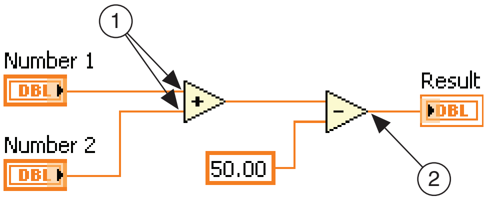
A. Dataflow

Objective: Recognize characteristics of dataflow on the block diagram.

Dataflow Dataflow is the movement of data through the nodes of a block diagram.

Key Points of Dataflow

LabVIEW follows a dataflow model for running VIs.



1 Node executed only when all the input data are available.
2 Node supplies data to the output terminals only when it is finished executing.

Activity 2-1: Exploring Dataflow

Goal: Understand how dataflow determines the execution order in a VI.

Description: Work through this exercise on your own first, and then as a class, we will discuss how data flow determines execution order. You can find answers to these questions following this section.

Note Nodes are objects on the block diagram that have inputs and/or outputs and perform operations when a VI runs.

Using Figure 2-1, answer questions 1 through 5.

1. Which node executes first? Is there any dependency between the File Dialog function and the Simulate Signal Express VI?

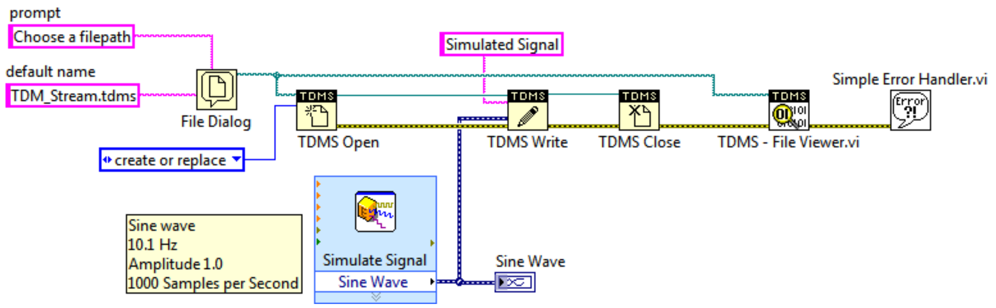
2. Which node executes last?

3. Because a wire connects the File Dialog function to the TDMS File Viewer VI, can the TDMS File Viewer VI execute before the TDMS Close function?

4. How many nodes must execute before the TDMS Write function can execute?

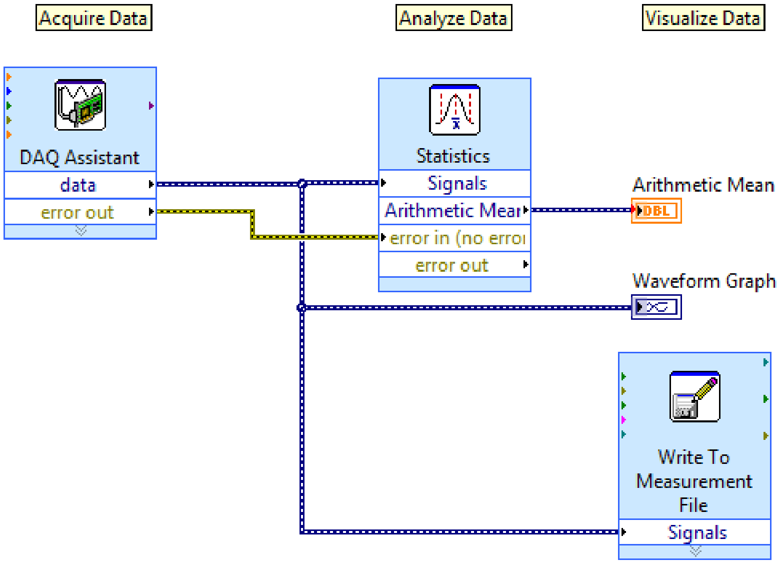
5. Should a well-designed block diagram flow in a particular direction?

Figure 2-1. Dataflow: Example A



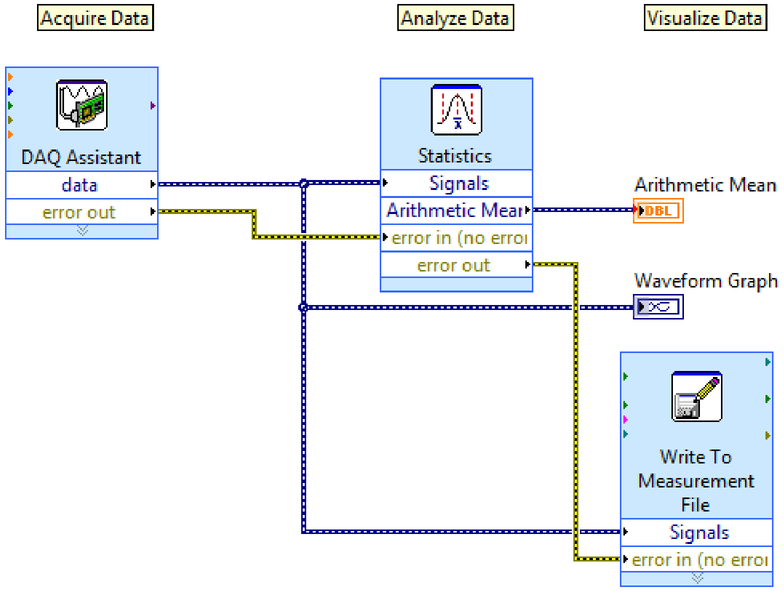
6. In Figure 2-2, which Express VI executes last?

Figure 2-2. Dataflow: Example B



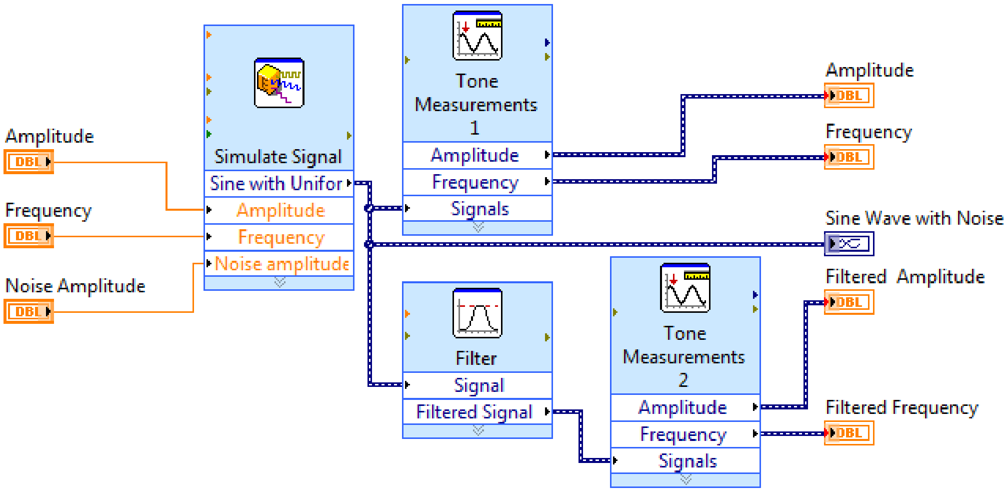
7. In Figure 2-3 an error wire connects the Express VIs. Which Express VI executes last?

Figure 2-3. Dataflow: Example C



8. In Figure 2-4, which Tone Measurements Express VI executes last?

Figure 2-4. Dataflow: Example D



Activity 2-1: Exploring Dataflow - Answers

1. Which node executes first? Is there any dependency between the File Dialog function and the Simulate Signal Express VI?

   Either the File Dialog function or the Simulate Signal Express VI can execute first. There is no data dependency between the two nodes so either of them can execute first or they can execute simultaneously.

2. Which node executes last?

   The last node to execute is the Simple Error Handler VI.

   Note Terminals are not considered nodes.

3. Because a wire connects the File Dialog function to the TDMS File Viewer VI, can the TDMS File Viewer VI execute before the TDMS Close function?

   No. The TDMS File Viewer VI cannot execute before the TDMS Close function because the yellow error wire connecting the TDMS Close function and the TDMS File Viewer VI forces data dependency. Remember, the data to all inputs of a node must be available before a node can execute. Therefore, the TDMS File Viewer VI must receive data from both the green Boolean wire and the yellow error wire before the VI can execute.

4. How many nodes must execute before the TDMS Write function can execute?

   Three nodes must execute before the TDMS Write function can execute: File Dialog, TDMS Open, and Simulate Signal. The TDMS Write function also depends on the Simulated Signal string constant, but that input is instantaneous.

5. Should a well-designed block diagram flow in a particular direction?

   Yes. A well-designed block diagram typically flows from left to right. This makes it easier to see the flow of data on the block diagram. However, do not assume left-to-right or top-to-bottom execution when no data dependency exists.

6. In Figure 2-2, which Express VI executes last?

   Either the Statistics Express VI or the Write to Measurement File Express VI executes last or they execute in parallel. The DAQ Assistant Express VI cannot execute last because both the Statistics Express VI and the Write to Measurement File Express VI are dependent on the data signal from the output of the DAQ Assistant Express VI.

   Note In LabVIEW, the flow of data, rather than the sequential order of commands, determines the execution order of block diagram elements. Therefore, it is possible to have simultaneous operations.

7. In Figure 2-3 an error wire connects the Express VIs. Which Express VI executes last?

   The Write to Measurement File Express VI executes last. It has a data dependency on both the DAQ Assistant Express VI and the Statistics Express VI.

8. In Figure 2-4, which Tone Measurements Express VI executes last?

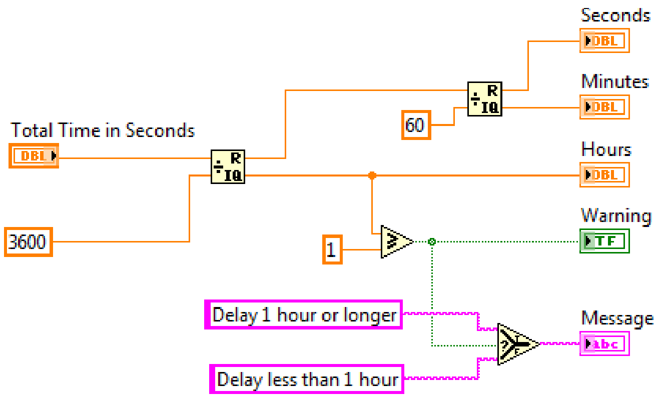
   Either one of the Tone Measurement Express VIs can execute last. Even though the Tone Measurements 2 Express VI has an extra dependency on the Filter Express VI, the Filter Express VI might execute before the Tone Measurements 1 Express VI allowing the Tone Measurements 2 Express VI to execute before the Tone Measurements 1 Express VI. Although it seems as if the Tone Measurements 1 Express VI would execute first, without an explicit data dependency there is no way to know definitely it would execute first.

B. LabVIEW Data Types

Objective: Recognize the different data types and how they relate to front panel objects.

Terminals and LabVIEW Data Types

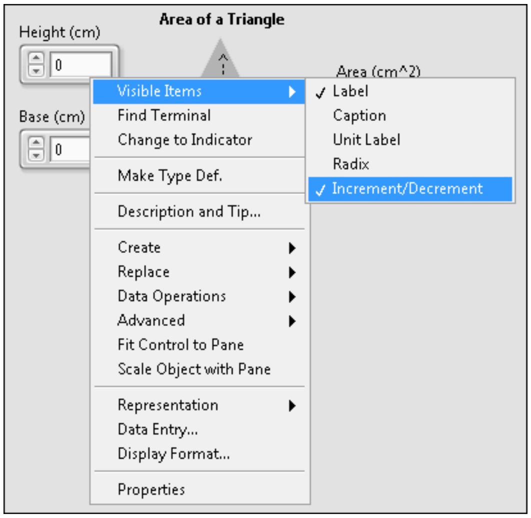
Block diagram terminals visually communicate information about the data type represented. Terminal colors, text, arrow direction, and border thickness all provide visual information about the terminal.

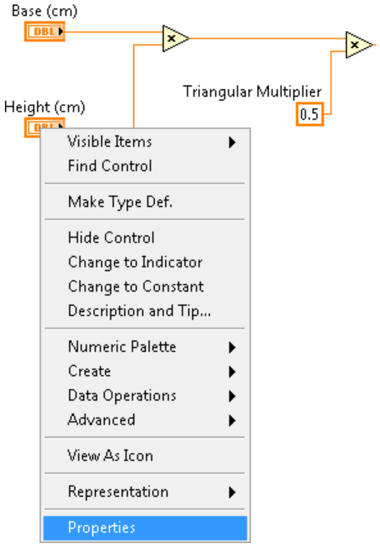


Demonstration: Accessing Object Properties

Shortcut Menus

- Use shortcut menu items to change the appearance or behavior of objects.
- To access the shortcut menu, right-click the object.



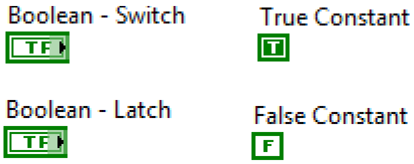


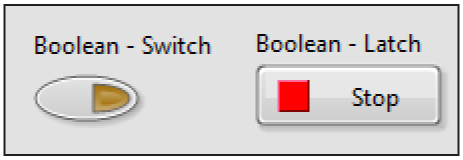
Property Dialog Boxes

- To access properties, right-click the object and select Properties.
- Select multiple objects to simultaneously configure shared properties.

Boolean Data

Use Boolean controls and indicators to enter and display Boolean (TRUE/FALSE) values. LabVIEW stores Boolean data as 8-bit values. If the 8-bit value is zero, the Boolean value is FALSE. Any nonzero value represents TRUE.





Multimedia: Mechanical Actions of Boolean Controls

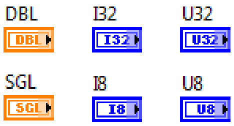
Boolean controls have six types of mechanical action that allow you to customize Boolean objects to create front panel windows that more closely resemble the behavior of physical instruments.

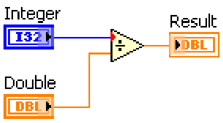
Complete the multimedia module, *Mechanical Actions of Boolean Controls*, available in the `<Exercises>\LabVIEW Core 1\Multimedia\` folder.

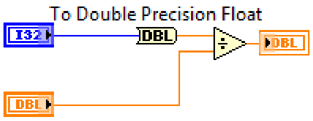
Multimedia: Numeric Data Representations

LabVIEW represents numeric data types as floating-point numbers, fixed-point numbers, integers, unsigned integers, and complex numbers. The difference among the numeric data types is the number of bits they use to store data and the data values they represent.

Complete the multimedia module, *Numeric Data Representations*, available in the `<Exercises>\LabVIEW Core 1\Multimedia\` folder.



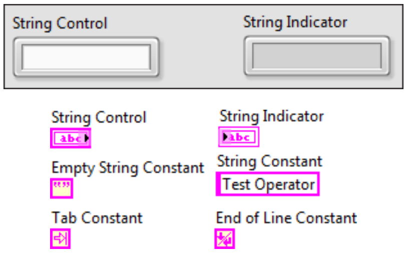


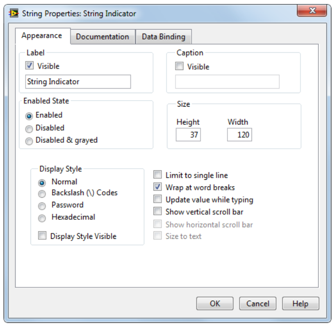


Strings

A string is a sequence of displayable or non-displayable ASCII characters. Strings provide a platform-independent format for information and data. Some of the more common applications of strings include the following:

- Creating simple text messages.
- Controlling instruments with text commands
- Storing numeric data to disk. To store numeric data in an ASCII file, first convert data to strings.
- Instructing or prompting with dialog boxes.




String Properties: String Indicator

Label
☑ Visible
String Indicator

Caption
□ Visible

Enabled State
● Enabled
○ Disabled
○ Disabled & grayed

Size
Height 37
Width 120

Display Style
● Normal
○ Backslash (\) Codes
○ Password
○ Hexadecimal
□ Display Style Visible

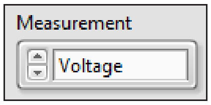
□ Limit to single line
☑ Wrap at word breaks
□ Update value while typing
□ Show vertical scroll bar
□ Show horizontal scroll bar
□ Size to text

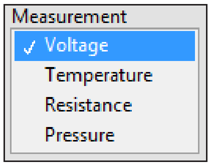
- Right-click a string control or indicator on the front panel to select from the display type.

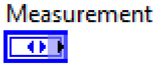
Display Type	Description	Message
Normal Display	Displays printable characters using the font of the control. Non-displayable characters generally appear as boxes.	`There are four display types. \ is a backslash.`
'\' Codes Display	Displays backslash codes for all non-displayable characters.	`There\sare\sfour\sdisplay\stypes.\n\\\sis\sa\sbackslash.`
Password Display	Displays an asterisk (*) for each character including spaces.	`*********************************************`
Hex Display	Displays the ASCII value of each character in hex instead of the character itself.	`5468 6572 6520 6172 6520 666F 7572 2064 6973 706C 6179 2074 7970 6573 2E0A 5C20 6973 2061 2062 6163 6B73 6C61 7368 2E`

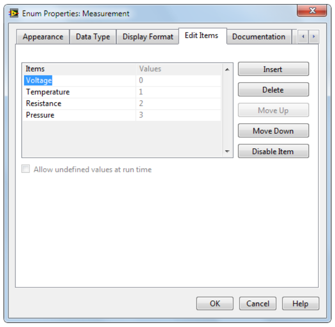
Demonstration: Enums

Enums give users a list of items from which to select. Enums are useful because they make strings equivalent to numbers, which are easy to manipulate on the block diagram. Each item represents a pair of values: a string value and a 16-bit integer.





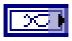
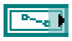
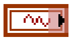




Items	Values
Voltage	0
Temperature	1
Resistance	2
Pressure	3

□ Allow undefined values at run time

Other Data Types

Type	Terminal	Description
Dynamic		Stores the information generated or acquired by an Express VI
Path		Stores the location of a file or directory using the standard syntax for the platform you are using
Waveform		Carries the data, start time, and dt of a waveform

How might a Path data type differ from a String data type?

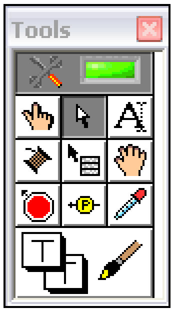
Although it is easy to convert a Path to a String, it is best to use a Path data type when working with a location of a file or directory because LabVIEW can handle the folder specifiers for the specific operating system, such as using a backslash on Windows OS.

C. Tools for Programming, Cleaning and Organizing Your VI

Objective: Recognize tools for making clean, readable front panels and block diagrams, and for documenting VIs.

Demonstration: Programming Tools

You can create, modify and debug VIs using the tools provided by LabVIEW. LabVIEW chooses which tool to select based on the current location of the mouse. If you need more control, use the Tools palette to select a specific tool.



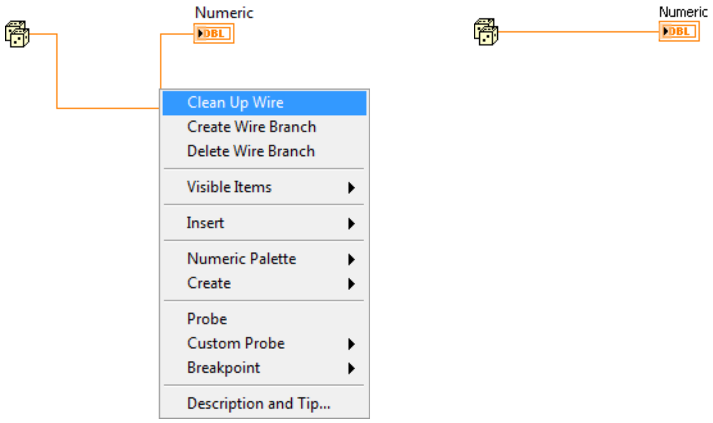
Tip You can manually choose the tool you need by selecting it on the **Tools** palette. Select **View»Tools Palette** or press <Shift> and right-click to display the palette.

Tool	Icon	Description
Operating	👆	Use the Operating tool to change the values of a control.
Positioning	↖	Use the Positioning tool to select or resize objects.
Labeling	I	Use the Labeling tool to enter text in a control, to edit text, and to create free labels.
Wiring	🧵	Use the Wiring tool to wire objects together on the block diagram.
Automatic Tool Selection	🛠	When this is selected, LabVIEW automatically chooses a tool based on the location of your cursor. You can turn off automatic tool selection by deselecting the item, or by selecting another item in the palette.
Object Shortcut Menu	↖▤	Use the Object Shortcut Menu tool to access an object shortcut menu with the left mouse button.
Scrolling	✋	Use the Scrolling tool to scroll through windows without using scrollbars.
Breakpoint	🛑	Use the Breakpoint tool to set breakpoints on VIs, functions, nodes, wires, and structures to pause execution at that location.

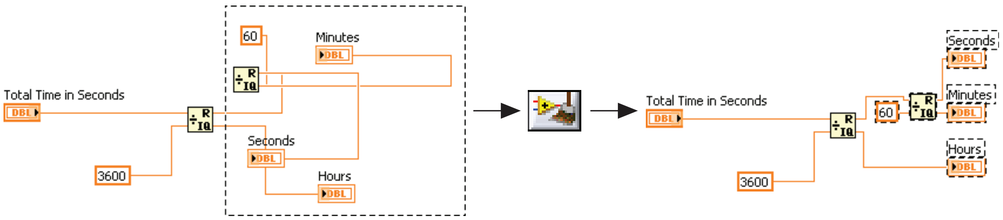
Tool	Icon	Description
Probe	P	Use the Probe tool to create probes on wires on the block diagram. Use the Probe tool to check intermediate values in a VI that produces questionable or unexpected results.
Color Copy		Use the Color Copy tool to copy colors for pasting with the Coloring tool.
Coloring		Use the Coloring tool and the color picker to color an object. The Coloring tool displays the current foreground and background color settings. Select the Coloring tool and right-click an object or workspace to display the color picker.

Demonstration: Wiring Tips

- Press <Ctrl-B> to delete all broken wires.
- To clean up one wire, right-click the wire and select **Clean Up Wire**.



- To reroute multiple wires and objects, select a section of your block diagram then click the **Clean Up Diagram** button.



- To clone an object, select the object with the Positioning tool and hold the <CTRL> key down while dragging the object to a new location.

Exercise 2-1 Selecting a Tool

Goal

Become familiar with automatic tool selection and the **Tools** palette in LabVIEW.

Description

During this exercise, you complete tasks in a partially built front panel and block diagram. These tasks give you experience using the automatic tool selection.

1. Open `Using Temperature.lvproj` in the `<Exercises>\LabVIEW Core 1\Using Temperature` directory.

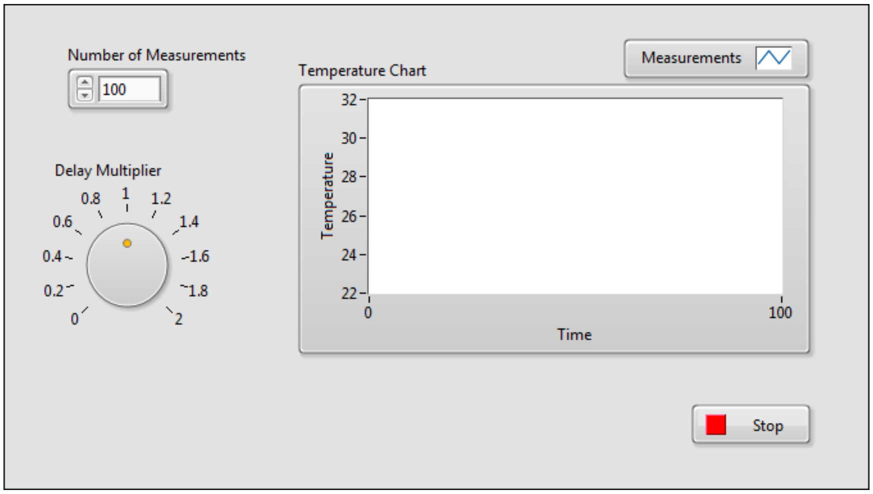
2. Open **Using Temperature.vi** from the **Project Explorer** window.

3. Select **View»Tools Palette** from the menu to display the **Tools** window.

Tip Press <Shift> and right-click the front panel to open the **Tools** palette temporarily.

Figure 2-5 shows an example of the front panel as it appears after your modifications. In steps 4 through 9 you increase the size of the waveform chart, rename the numeric control, change the value of the numeric control, and move the knob.

Figure 2-5. Using Temperature VI Front Panel

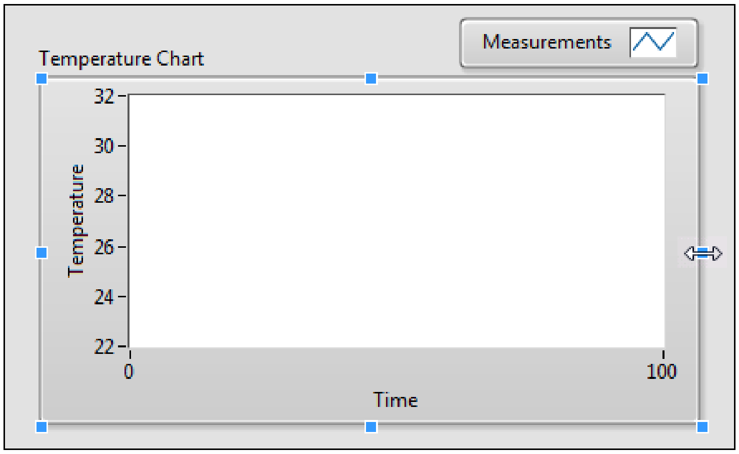


4. Expand the waveform chart horizontally using the Positioning tool.

   □ Move the cursor to the right edge of the **Chart** waveform chart until you see the resizing nodes appear around the chart.

   □ Move the cursor to the middle right resizing node until the cursor changes to a double arrow, as shown in Figure 2-6.

Figure 2-6. Resize Waveform Chart



   □ Drag the repositioning node until the waveform chart is the size you want.

5. Rename the waveform chart using the Labeling tool.

   □ Double-click the word `Chart`. LabVIEW highlights the word and automatically selects the Labeling tool in the **Tools** window.

   □ Enter the text `Temperature Chart`.

   □ Complete the entry by clicking outside the control label or clicking the **Enter Text** button on the toolbar.

   □ Notice that LabVIEW automatically returns to the Positioning tool in the **Tools** window. The Positioning tool is the default tool. If LabVIEW does not switch back to the Positioning tool, click the **Automatic Tool Selection** button in the **Tools** window to enable automatic tool selection.

6. Rename the **Numeric** control to `Number of Measurements` using the Labeling tool.

   □ Double click the word `Numeric`.

   □ Enter the text `Number of Measurements`.

   □ Complete the entry by clicking outside the control or clicking the **Enter Text** button on the toolbar.

7. Change the value of the **Number of Measurements** control to `100` using the Labeling tool.

   □ Move the cursor to the interior of the **Number of Measurements** control.

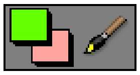
   □ When the cursor changes to the Labeling tool icon, click the mouse button.

   □ Enter the text `100`.

- □ Complete the entry by clicking outside the control, clicking the **Enter Text** button on the toolbar, or pressing the <Enter> key on the numeric keypad,
- □ Change the value of the **Delay Multiplier** knob using the Operating tool.
- □ Move the cursor to the knob.
- □ When the cursor changes to the Operating tool icon, press the mouse button and drag to the value you want.
- □ Set the value to `1`.

8. Change the color of the **Delay Multiplier** knob using the Coloring tool.

- □ Click the background square in the **Coloring** button and select a color from the color picker.



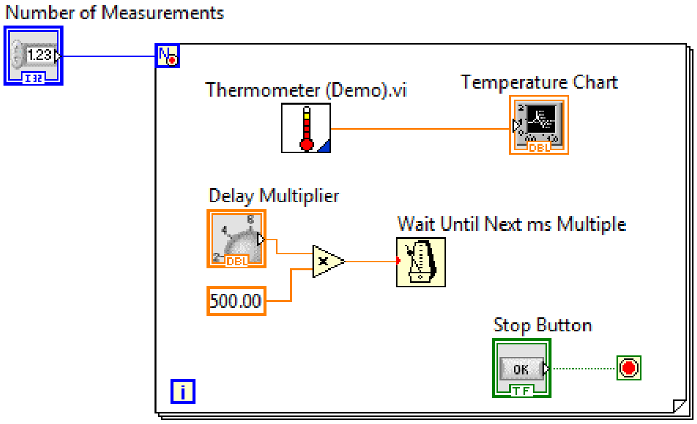
- □ When the cursor changes to a paintbrush, click the **Delay Multiplier** knob.
- □ Click the **Automatic Tool Selection** button again to turn on automatic tool selection.

9. Try changing the value of objects, resizing objects, and renaming objects until you are comfortable with using these tools.

10. Open the block diagram of the VI.

Figure 2-7 shows an example of the block diagram as it appears after your modifications. Steps 11 through 12 instruct you on how to update the block diagram to move the **Number of Measurements** terminal and wire the terminal to the count terminal of the For Loop.

Figure 2-7. Using Temperature VI Block Diagram



11. Move the **Number of Measurements** terminal using the Positioning tool.

- □ Move the cursor to the **Number of Measurements** terminal.
- □ Move the cursor over the terminal until the cursor changes to an arrow.
- □ Click and drag the terminal to the new location as shown in Figure 2-7.

12. Wire the **Number of Measurements** terminal to the count terminal of the For Loop using the Wiring tool.

    □ Move the cursor to the **Number of Measurements** terminal.

    □ Move the cursor to the right of the terminal, stopping when the cursor changes to a wiring spool.

    □ Click to start the wire.

    □ Move the cursor to the count (**N**) terminal of the For Loop.

    □ Click the count terminal to end the wire.

13. Try moving other objects, deleting wires and rewiring them, and wiring objects and wires together until you are comfortable with using these tools.

14. Automatically clean up the entire block diagram.

    □ Click the **Clean Up Diagram** button on the LabVIEW toolbar.

    □ Press <Ctrl-Z> to undo the clean-up.

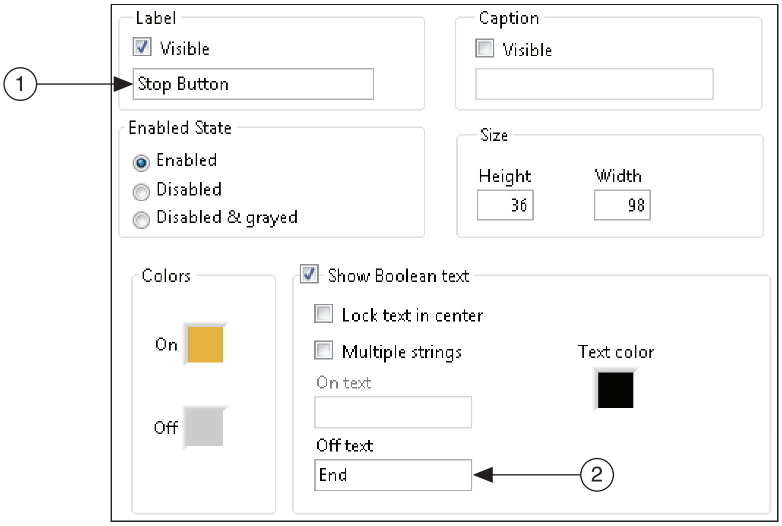
Tip You can also select specific objects to clean up, such as wires or individual nodes. <Shift>-click to select multiple objects and then click the **Clean Up Diagram** button. LabVIEW cleans up only the objects that you select and not the entire block diagram. Configure how LabVIEW cleans up objects by selecting **Tools»Options** from the menu, clicking the **Block Diagram** category, and changing the options in the **Block Diagram Cleanup** section.

15. Change the Boolean text of the **Stop** button.

Note Boolean controls and indicators have Boolean text labels in addition to their control labels. Boolean text labels change depending on the value of the control or indicator. The label for the control or indicator does not change depending on the value of the control or indicator.

□ Right-click the **Stop Button** terminal and select **Properties** from the shortcut menu. Set the properties as shown in Figure 2-8.

Figure 2-8. Changing the Boolean Text for the Stop Control



Label
☑ Visible
Stop Button

Caption
□ Visible

Enabled State
● Enabled
○ Disabled
○ Disabled & grayed

Size
Height 36
Width 98

Colors
On
Off

☑ Show Boolean text
□ Lock text in center
□ Multiple strings
On text
Off text
End
Text color

1 **Control label**—This text identifies the terminal of the Boolean control for programming purposes. This text does not appear on the front panel unless you select **Visible**.
2 **Boolean text**—This text appears only on the front panel, and by default, appears in the center of the Boolean control.

□ Click the **OK** button to close the dialog box.

□ Right-click the **Stop Button** terminal and select **Find Control** from the shortcut menu. Notice the control label is **Stop Button** and the button text is **End**.

Tip You can also double-click the **Stop Button** terminal to find the button control on the front panel.

16. Click the **Run** button to run the VI.

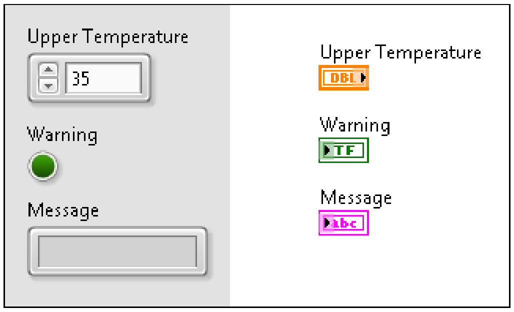
The time required to execute this VI is equivalent to **Number of Measurements** times **Delay Multiplier**. When the VI is finished executing, the **Temperature Chart** displays the data.

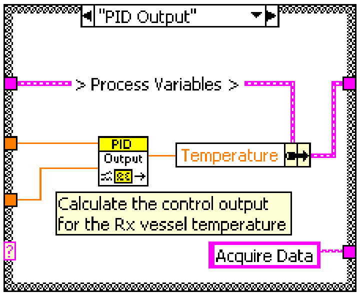
17. Close the VI and click the **Don’t Save** - **All** button. You do not need to save the VI.

End of Exercise 2-1

Demonstration: Making Code Readable

Block diagram comments can describe the function or operation of algorithms and explain the purpose of data that passes through wires.





Owned Labels	Free Labels
Explain data contents of wires and objects.	Describe algorithms.
Move with object.	Have fixed location.
Have transparent backgrounds.	Have pale yellow backgrounds.
Created by selecting **Visible Items»Label** from the shortcut menu.	Created by double-clicking in any open space.

Label Guidelines

- Use free labels to document algorithms and add reference information.
- Label structures to specify the main functionality.
- Label long wires to identify their use/contents.
- Label constants to specify the nature of the constant.
- To avoid clutter, document the Context Help window of subVIs or functions instead of labels.

Default Values

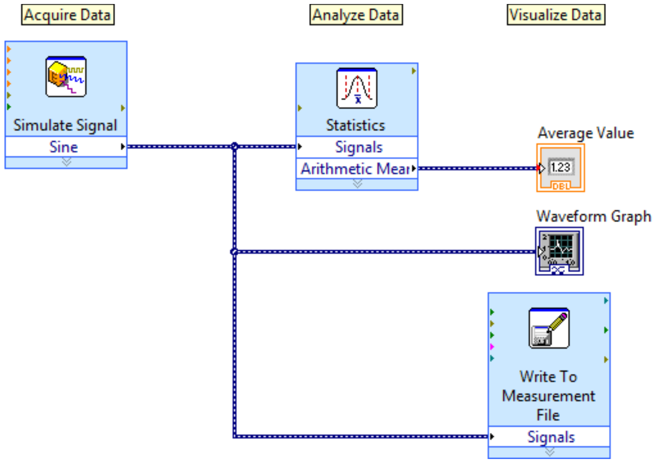
Setting a default value on a control or indicator can aid in future development and troubleshooting by giving the programmer an idea of the expected data for that object. Configure default values for controls and indicators by right-clicking on the object and selecting **Data Operations»Make Current Value Default** from the short-cut menu.

D. Building a Basic VI

Objective: Recognize Express VIs and be able to apply them appropriately.

Building a Basic VI

Express VIs are designed specifically for completing common, frequently used operations.



On the **Functions** palette, the Express VIs are grouped together in the **Express** category. Express VIs use the dynamic data type to pass data between Express VIs.

	VI	Icon	Description
Acquire	DAQ Assistant		Acquires data through a data acquisition device.You must use this Express VI frequently throughout this course.
	Instrument I/O Assistant		Acquires instrument control data, usually from a GPIB or serial interface.
	Simulate Signal		Generates simulated data such as a sine wave.
	Read from Measurement File		Reads a file that was created using the Write To Measurement File Express VI. It specifically reads LVM or TDM file formats. This Express VI does not read ASCII files.

	VI	Icon	Description
Analyze	Amplitude and Level Measurements		Performs voltage measurements on a signal. These include DC, rms, maximum peak, minimum peak, peak to peak, cycle average, and cycle rms measurements.
	Statistics		Calculates statistical data from a waveform. This includes mean, sum, standard deviation, and extreme values.
	Tone Measurements		Searches for a single tone with the highest frequency or highest amplitude. It also finds the frequency and amplitude of a single tone.
	Spectral Measurements	FFT	Performs spectral measurement on a waveform, such as magnitude and power spectral density.
	Filter		Processes a signal through filters and windows. Filters used include the following: Highpass, Lowpass, Bandpass, Bandstop, and Smoothing. Windows used include Butterworth, Chebyshev, Inverse Chebyshev, Elliptical, and Bessel.
Visualize	Write to Measurement File		Writes a file in LVM or TDMS file format.
	Build Text	1.3 LEC ↓ ↓ 1.3 LEC	Creates text, usually for displaying on the front panel window or exporting to a file or instrument.
	Front panel indicators	—	Examples include Waveform Chart, Waveform Graph, and XY Graph.

Activity 2-2: Program Architecture for Simple AAV VI

1. **Acquire**: Circle the Express VI that is best suited to acquire a sine wave from a data acquisition device.

	DAQ Assistant	The DAQ Assistant acquires data through a data acquisition device.
	Instrument I/O Assistant	The Instrument I/O Assistant acquires instrument control data, usually from a GPIB or serial interface.
	Simulate Signal	The Simulate Signal Express VI generates simulated data, such as a sine wave.

2. **Analyze**: Circle the Express VI that is best suited to determining the average value of the acquired data.

	Tone Measurements	The Tone Measurements Express VI finds the frequency and amplitude of a single tone.
	Statistics	The Statistics Express VI calculates statistical data from a waveform.
	Amplitude and Level Measurements	The Amplitude and Level Measurements Express VI performs voltage measurements on a signal.
	Filter	The Filter Express VI processes a signal through filters and windows.

3. **Visualize**: Circle the Express VIs and/or indicators that are best suited to displaying the data on a graph and logging the data to file.

	DAQ Assistant	The DAQ Assistant acquires data through a data acquisition device.
	Write to Measurement File	The Write to Measurement File Express VI writes a file in LVM or TDM file format.
	Build Text	The Build Text Express VI creates text, usually for displaying on the front panel window or exporting to a file or instrument.
	Waveform Graph	The waveform graph displays one or more plots of evenly sampled measurements.

Refer to the next page for answers to this quiz.

Activity 2-2: Program Architecture for Simple AAV VI - Answers

1. **Acquire**: Use the DAQ Assistant to acquire the sine wave from the data acquisition device.

2. **Analyze**: Use the Statistics Express VI to determine the average value of the sine wave. Because this signal is cyclical, you could also use the Cycle Average option in the Amplitude and Level Measurements Express VI to determine the average value of the sine wave.

3. **Visualize**: Use the Write to Measurement File Express VI to log the data and use the Waveform Graph to display the data on the front panel window.

Exercise 2-2 Simple AAV VI

Goal

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

Scenario

You need to acquire a sine wave for 0.1 seconds, determine and display the average value, log the data, and display the sine wave on a graph.

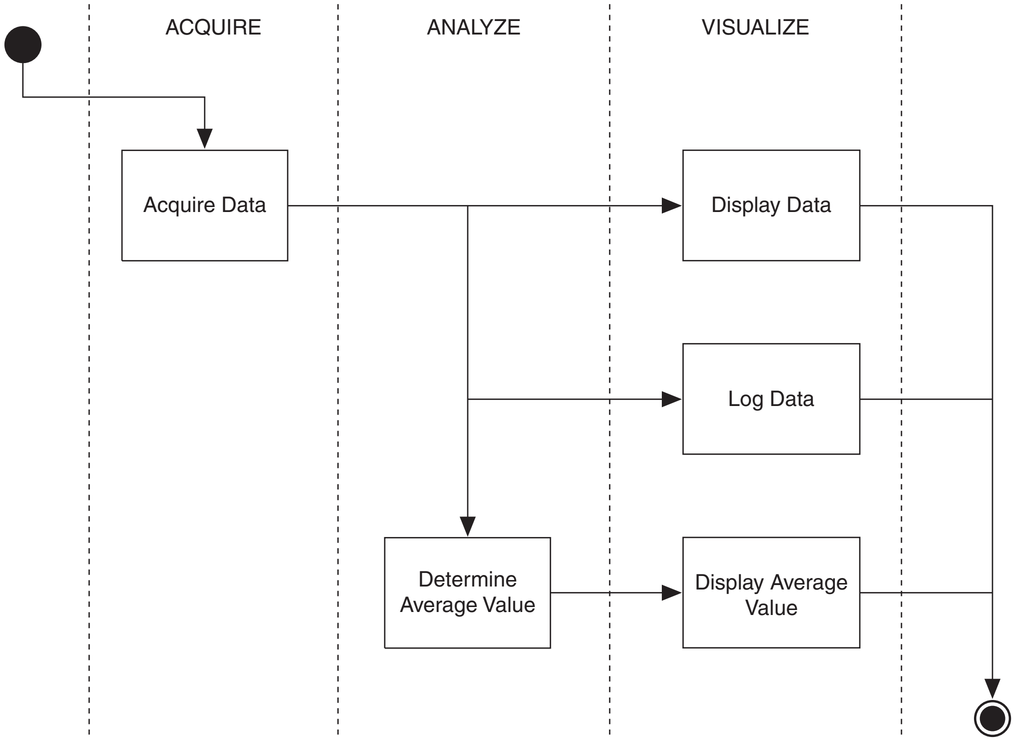
Design

The input for this problem is an analog channel of sine wave data. The outputs include a graph of the sine data, a file that logs the data, and an indicator that displays the average data value.

Flowchart

The flowchart in Figure 2-9 illustrates the data flow for this design.

Figure 2-9. Simple AAV VI Flowchart



Implementation

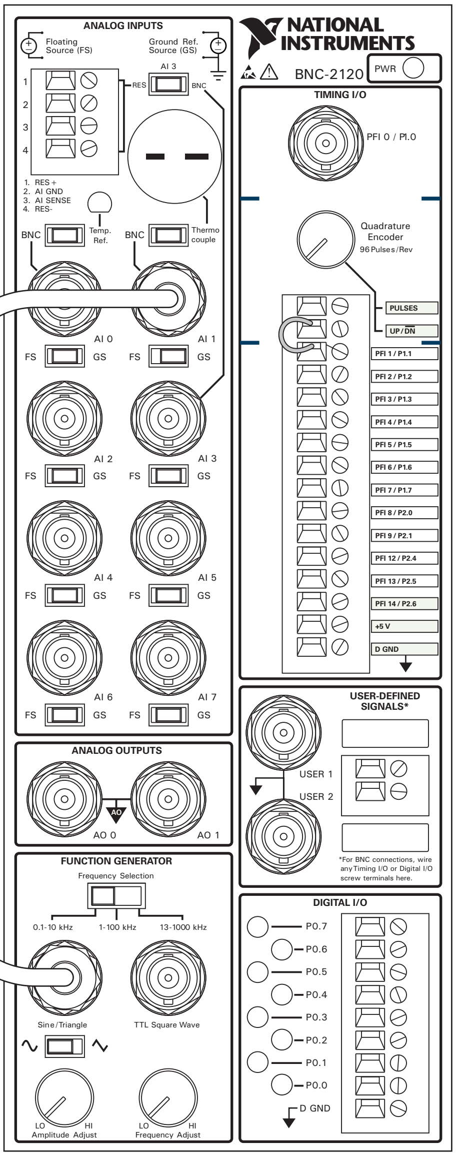
1. Prepare your hardware to generate a sine wave. If you are not using hardware, skip to step 2.

   □ Find the BNC-2120 and visually confirm that it is connected to the DAQ device in your computer.

   □ Using a BNC cable, connect the Analog In Channel 1 to the Sine Function Generator, as shown in Figure 2-10.

   □ Set the **Frequency Selection** switch and the **Frequency Adjust** knob to their lowest levels.

Figure 2-10. Connections for the BNC-2120



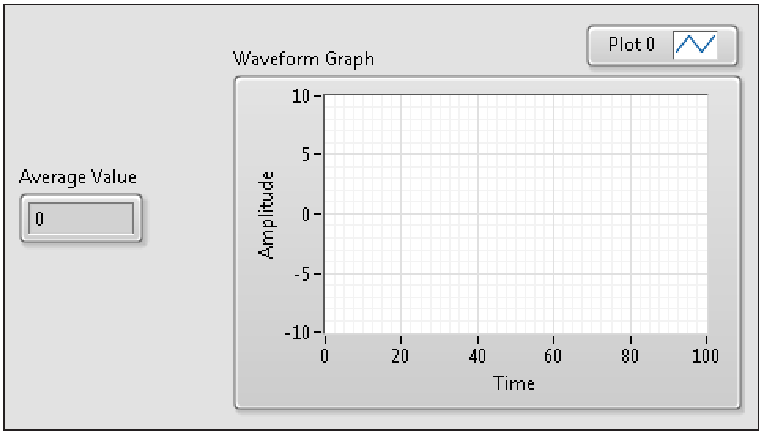
2. Open LabVIEW.

3. Create a blank project. Save the project as `Simple AAV.lvproj` in the `<Exercises>\LabVIEW Core 1\Simple AAV` directory.

4. Add a new VI to the project from the **Project Explorer** window and save the VI as `Simple AAV.vi` in the `<Exercises>\LabVIEW Core 1\Simple AAV` directory.

In steps 5 through 6 you will build a front panel similar to the one in Figure 2-11.

Figure 2-11. Simple AAV VI Front Panel Window



5. Add a waveform graph to the front panel window to display the acquired data.

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

   □ Type `Waveform` in the text box and double-click **Waveform Graph (Silver)** in the search results list.

   □ Place the graph in the front panel window.

6. Add a numeric indicator to the front panel window to display the average value.

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

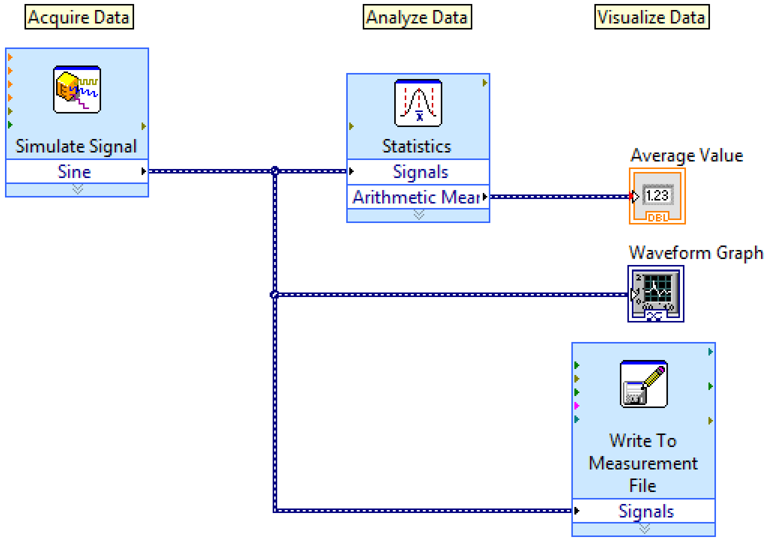
   □ Type `Numeric Indicator` in the text box and double-click **Numeric Indicator (Silver)** in the search results list.

   □ Place the indicator in the front panel window.

   □ Change the numeric indicator label to `Average Value`.

In the steps 7 through 14 you build a block diagram similar to the one in Figure 2-12.

Figure 2-12. Simple AAV VI Block Diagram



7. Open the block diagram of the VI by selecting **Window»Show Block Diagram**.

Note The terminals corresponding to the front panel window objects appear on the block diagram.

8. Acquire a sine wave for 0.1 seconds by following the instructions in Table 2-1. If you have hardware installed, follow the instructions in the **Hardware Installed** column to acquire the data using the DAQ Assistant. If you do not have hardware installed, follow the instructions in the **No Hardware Installed** column to simulate the acquisition using the Simulate Signal Express VI.

Table 2-1. Instructions for Acquiring or Simulating Data

Hardware Installed	No Hardware Installed
1. Press <Ctrl-Space> to open the **Quick Drop** dialog box.	1. Press <Ctrl-Space> to open the **Quick Drop** dialog box.
2. Type `DAQ Assist` in the text box and double-click **DAQ Assistant** in the search results list.	2. Type `Simulate Signal` in the text box and double-click **Simulate Signal** in the search results list.
3. Place the DAQ Assistant on the block diagram.	3. Place the Simulate Signal Express VI on the block diagram.
4. Wait for the DAQ Assistant dialog box to open.	4. Wait for the Simulate Signal dialog box to open.
5. Select **Acquire Signals»Analog Input» Voltage** for the measurement type.	5. Select **Sine** for the signal type.
6. Select **ai1** (analog input channel 1) for the physical channel.	6. Set the signal frequency to `100`.
7. Click the **Finish** button.	7. In the **Timing** section, set the **Samples per second (Hz)** to `1000`.

Table 2-1. Instructions for Acquiring or Simulating Data (Continued)

Hardware Installed	No Hardware Installed
8. In the **Timing Settings** section, select **N Samples** as the **Acquisition Mode**.	8. In the **Timing** section, deselect **Automatic** for the **Number of samples**.
9. In the **Timing Settings** section enter `100` in **Samples To Read**.	9. In the **Timing** section, set the **Number of samples** to `100`.
10. Enter `1000` in **Rate (Hz)**.	10. In the **Timing** section, select the **Simulate acquisition timing** option.
11. Click the **OK** button.	11. Click the **OK** button.

Tip Reading 100 samples at a rate of 1,000 Hz retrieves 0.1 seconds worth of data.

9. Use the Statistic Express VI to determine the average value of the data acquired.

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

   □ Type `statistics` in the text box and double-click **Statistics [NI_ExpressFull.lvlib]** in the search results list.

   □ Place the **Statistics** Express VI on the block diagram to the right of the DAQ Assistant (or Simulate Signal Express VI).

   □ Wait for the Statistics Express VI dialog box to open.

   □ Place a checkmark in the **Arithmetic mean** checkbox.

   □ Click the **OK** button.

10. Log the generated sine data to a LabVIEW Measurement File.

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

   □ Type `write to measurement` in the text box and double-click **Write to Measurement File** in the search results list.

   □ Place the **Write to Measurement File** Express VI on the block diagram below the Statistics Express VI.

   □ Wait for the Write to Measurement File Express VI dialog box to open.

   □ Leave all configuration settings in the Write to Measurement File dialog box as default.

   □ Click the **OK** button.

Note Future exercises do not detail the directions for finding specific functions or controls in the palettes. Use Quick Drop, the palette search feature, or the global search to locate functions and controls.

11. Wire the data from the DAQ Assistant (or Simulate Signal Express VI) to the Statistics Express VI.

   □ Place the mouse cursor over the **data** output of the DAQ Assistant (or the **Sine** output of the Simulate Signal Express VI) at the location where the cursor changes to the Wiring tool.

   □ Click the mouse button to start the wire.

- □ Place the mouse cursor over the **Signals** input of the Statistics Express VI and click the mouse button to end the wire.

12. Wire the data to the graph indicator.

- □ Place the mouse cursor over the **data** output wire of the DAQ Assistant (or the **Sine** output of the Simulate Signal Express VI) at the location where the cursor changes to the Wiring tool.
- □ Click the mouse button to start the wire.
- □ Place the mouse cursor over the **Waveform Graph** indicator and click the mouse button to end the wire.

13. Wire the **Arithmetic Mean** output of the Statistics Express VI to the **Average Value** numeric indicator.

- □ Place the mouse cursor over the **Arithmetic Mean** output of the Statistics Express VI at the location where the cursor changes to the Wiring tool.
- □ Click the mouse button to start the wire.
- □ Place the mouse cursor over the **Average Value** numeric indicator and click the mouse button to end the wire.

14. Wire the **data** output to the **Signals** input of the Write Measurement File Express VI.

- □ Place the mouse cursor over the **data** output wire of the DAQ Assistant (or the **Sine** output of the Simulate Signal Express VI) at the location where the cursor changes to the Wiring tool.
- □ Click the mouse button to start the wire.
- □ Place the mouse cursor over the **Signals** input of the Write Measurement File Express VI and click the mouse button to end the wire.

Note Future exercises do not detail the directions for wiring between objects.

15. Save the VI.

Test

1. Switch to the front panel window of the VI.
2. Set the graph properties to be able to view the sine wave.
   - □ Right-click the waveform graph and select **X Scale»Autoscale X** to remove the checkmark and disable autoscaling.
   - □ Use the labeling tool to change the last number on the Time scale of the waveform graph to `.1`.
3. Save the VI.
4. Click the **Run** button on the front panel toolbar to run the VI. The graph indicator should display a sine wave and the **Average Value** indicator should display a number around zero. If the VI does not run as expected, review the implementation steps.
5. Close the VI.

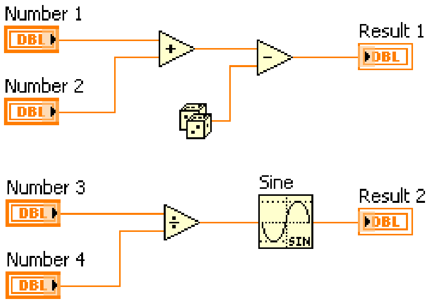
End of Exercise 2-2

Additional Resources

Learn More About	LabVIEW Help Topic
Terminal symbols for controls and indicators	*Numeric Data Types Table*
Using Express VIs	*Express VIs* *Working with Express VIs* *Acquiring or Simulating a Signal*
Visualizing data	*Types of Graphs and Charts*

Activity 2-3: Lesson Review

Refer to the figure below to answer the following quiz questions.



1. Which function executes first: Add or Subtract?
   a. Add
   b. Subtract
   c. Unknown

2. Which function executes first: Sine or Divide?
   a. Sine
   b. Divide
   c. Unknown

3. Which function executes first: Random Number, Divide or Add?
   a. Random Number
   b. Divide
   c. Add
   d. Unknown

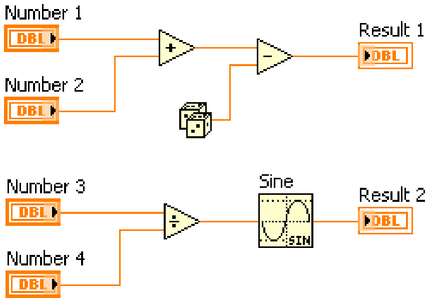
4. Which function executes last: Random Number, Subtract or Add?
   a. Random Number
   b. Subtract
   c. Add
   d. Unknown

5. If an input to a function is marked with a red dot (known as coercion dot), what does the dot indicate?
   a. Data was transferred into a structure.
   b. That input has not been wired
   c. The wire is broken
   d. The value passed into a node was converted to a different representation.

6. Which mechanical action causes a Boolean control in the FALSE state to change to TRUE when you click it and stay TRUE until LabVIEW has read the value?
   a. Switch Until Released
   b. Switch When Released
   c. Latch When Pressed
   d. Latch When Released

Activity 2-3: Lesson Review - Answers

Refer to the figure below to answer the following quiz questions.



1. Which function executes first: Add or Subtract?

    a. Add

    b. Subtract

    c. Unknown

2. Which function executes first: Sine or Divide?

    a. Sine

    b. Divide

    c. Unknown

3. Which function executes first: Random Number, Divide or Add?

    a. Random Number

    b. Divide

    c. Add

    d. Unknown

4. Which function executes last: Random Number, Subtract or Add?

    a. Random Number

    b. Subtract

    c. Add

    d. Unknown

5. If an input to a function is marked with a red dot (known as coercion dot), what does the dot indicate?

    a. Data was transferred into a structure.

    b. That input has not been wired

    c. The wire is broken

    d. The value passed into a node was converted to a different representation.

6. Which mechanical action causes a Boolean control in the FALSE state to change to TRUE when you click it and stay TRUE until LabVIEW has read the value?

    a. Switch Until Released

    b. Switch When Released

    c. Latch When Pressed

    d. Latch When Released

3 Troubleshooting and Debugging VIs

In this lesson you will learn how to use LabVIEW tools to debug and troubleshoot VIs and how to implement basic error handling techniques.

Topics

+ Correcting Broken VIs
+ Debugging Techniques
+ Error Handling

Exercises

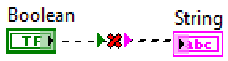
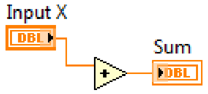
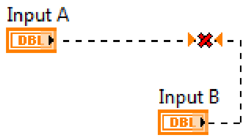
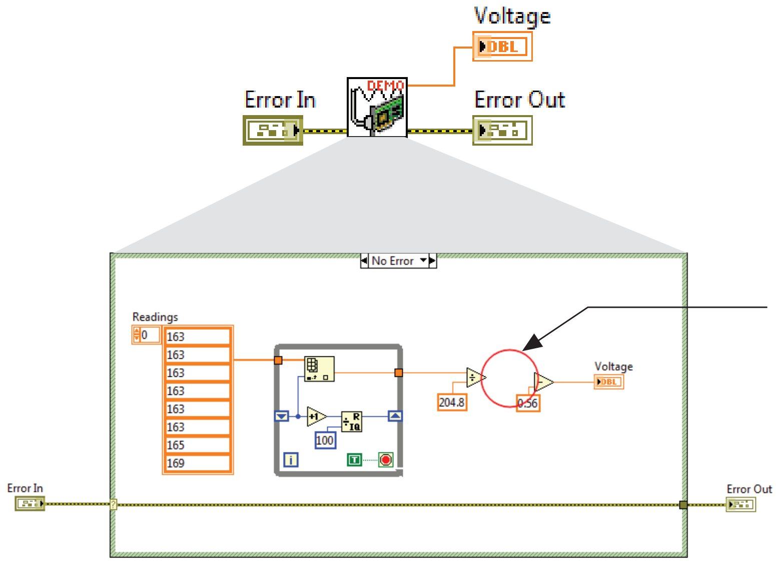
Exercise 3-1 Debugging

A. Correcting Broken VIs

Objective: Recognize an unexecutable VI and restate common problems.

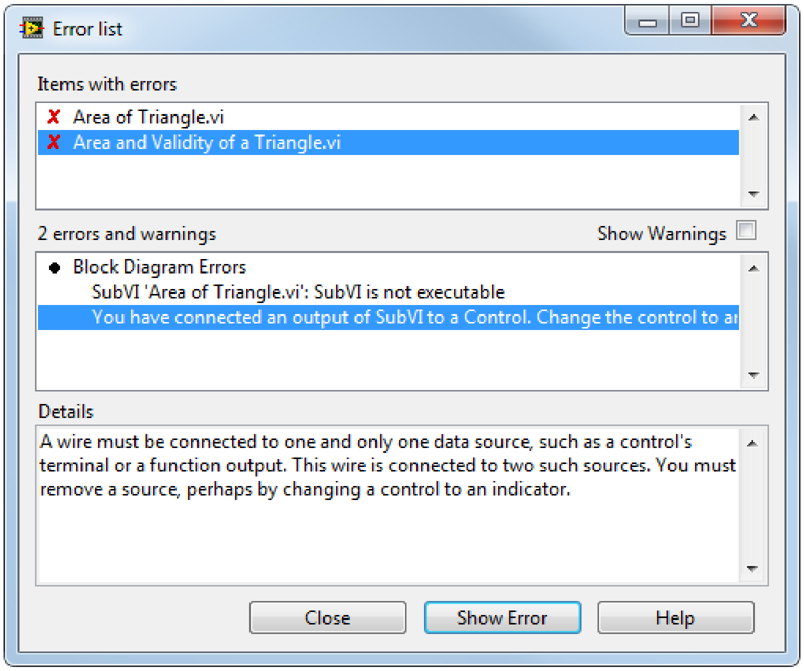
If the **Run** button appears broken when you finish wiring the block diagram, the VI is broken and cannot run.

Common Causes of Broken VIs

Example	Cause
	Broken wire due to mismatch of data types.
	Required terminal is unwired.
	Control wired to another control.
 Voltage DBL Error In DEMO Error Out No Error Readings 0 163 163 163 163 163 163 165 169 100 204.8 0.56 Voltage DBL Error In Error Out	Broken subVI due to unwired required terminal.

Identify Problems and Fix Broken VIs

Use the Error list dialog box to locate the source of errors and warnings, and get more detailed information about the problem.



Error list

Items with errors

- ✗ Area of Triangle.vi
- ✗ Area and Validity of a Triangle.vi

2 errors and warnings Show Warnings □

- Block Diagram Errors
  - SubVI 'Area of Triangle.vi': SubVI is not executable
  - You have connected an output of SubVI to a Control. Change the control to ar

Details

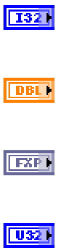
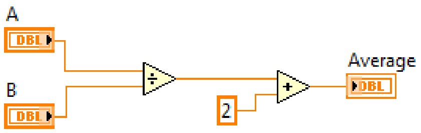
A wire must be connected to one and only one data source, such as a control's terminal or a function output. This wire is connected to two such sources. You must remove a source, perhaps by changing a control to an indicator.

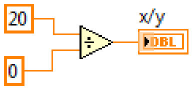
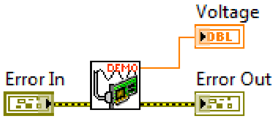
B. Debugging Techniques

Objective: Identify the LabVIEW tools for debugging and select the tool appropriate to the situation.

Debugging Tips

The following table includes a few common problems to look for when debugging your VIs.

	Is the numeric representation correct for your application?
	Do nodes execute in the correct order?

	Does the VI pass undefined data?
	Are there any hidden or unwired subVIs?

Job Aid

If a VI is not broken, but you get unexpected data, you can use the following checklist to identify and correct problems with the VI or the block diagram data flow:

☐ Wire the error in and error out parameters at the bottom of most built-in VIs and functions.

☐ Select **View»Error List** and place a checkmark in the **Show Warnings** checkbox to see all warnings for the VI.

☐ Triple-click a wire to highlight its entire path and to ensure that the wires connect to the proper terminals.

☐ Use the **Context Help** window to check the default values for each function and subVI on the block diagram.

☐ Use the **Find** dialog box to search for subVIs, text, and other objects to correct throughout the VI.

☐ Select **View»VI Hierarchy** to find unwired subVIs.

☐ Use execution highlighting to watch the data move through the block diagram.

☐ Single-step through the VI to view each action of the VI on the block diagram.

☐ Use the Probe tool to observe intermediate data values and to check the error output of VIs and functions, especially those performing I/O.

☐ Click the **Retain Wire Values** button on the block diagram toolbar to retain wire values for use with probes.

☐ Use breakpoints to pause execution, so you can single-step or insert probes.

☐ Suspend the execution of a subVI to edit values of controls and indicators, to control the number of times it runs, or to go back to the beginning of the execution of the subVI.

☐ Determine if the data that one function or subVI passes is undefined.

☐ If the VI runs more slowly than expected, confirm that you turned off execution highlighting in subVIs.

☐ Check the representation of controls and indicators to see if you are receiving overflow because you converted a floating-point number to an integer or an integer to a smaller integer.

☐ Determine if any For Loops inadvertently execute zero iterations and produce empty arrays.

☐ Verify you initialized shift registers properly unless you intend them to save data from one execution of the loop to another.

□ Check the cluster element order at the source and destination points.

□ Check the node execution order.

□ Check that the VI does not contain hidden subVIs.

□ Check the inventory of subVIs the VI uses against the results of **View»Browse Relationships»This VI's SubVIs** and **View»Browse Relationships»Unopened SubVIs** to determine if any extra subVIs exist. Also open the VI Hierarchy window to see the subVIs for a VI. To help avoid incorrect results caused by hidden VIs, specify that inputs to VIs are required.

Activity 3-1: Review Debugging Tools

Goal

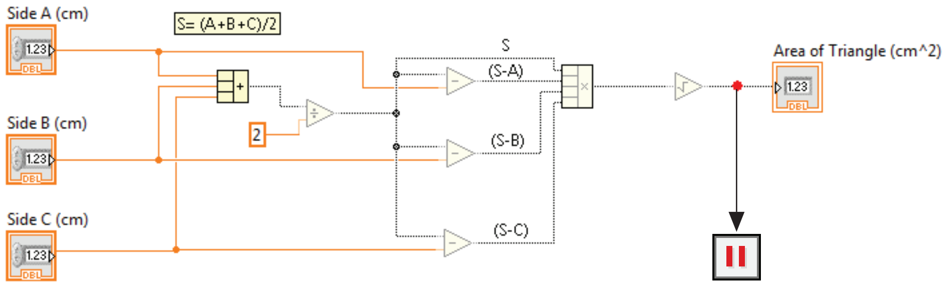
Select the LabVIEW debugging tool you would use for each of the following situations.

Execution Highlighting	Single Stepping	Probe
💡	↳□ □↱ □↑	P

Situation	Debugging Tool
Your VI is returning unexpected data and you want to see intermediate values on the wires to find out where the problem initiates.	**Probe**
You want to see how data moves from one node to the next on the block diagram.	
You want to view each action of the VI on the block diagram.	
This tool slows down how fast the VI runs so you can see the data flow through the block diagram.	
Your block diagram is complicated and you need to see the intermediate data values and check the error output of VIs and functions, especially those performing I/O.	
Suspend the execution of a subVI to edit values of controls and indicators, to control the number of times it runs, or to go back to the beginning of the execution of the subVI.	

Breakpoints

Use the Breakpoint tool to place a breakpoint on a VI, node, or wire and pause execution at that location.

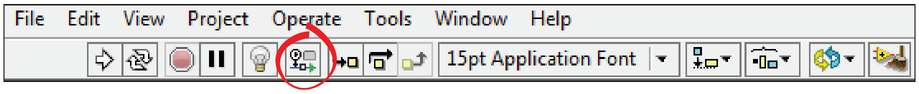


When you reach a breakpoint during execution, the VI pauses and the **Pause** button appears red. You can take the following actions:

- Single-step through execution using the single-stepping buttons
- Probe wires to check intermediate values
- Change the values of front panel controls.
- Click the **Pause** button to continue running to the next breakpoint or until the VI finishes running.

Retain Wire Values

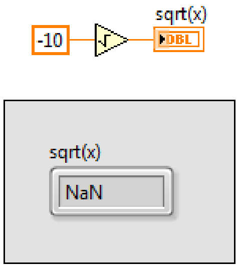
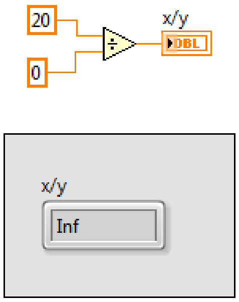
Retain Wire Values enables you to probe wire values after execution completes. This option uses more memory because LabVIEW is storing data for each wire as it runs.



Tip Always remember to turn this feature off when you are done debugging to free memory.

Undefined or Unexpected Data

Floating-point operations return the following two symbolic values that indicate faulty computations or meaningless results.

NaN (not a number)		a floating-point value that invalid operations produce, such as taking the square root of a negative number
Inf (infinity)		a floating-point value that valid operations produce, such as dividing a number by zero

Exercise 3-1 Debugging

Goal

Use the debugging tools built into LabVIEW.

Description

The VIs in this exercise check the validity of a triangle and then calculate the area. For a triangle to be valid, all three sides must have a length that is greater than zero. The subVI in this exercise uses Heron's formula to calculate the area of a triangle. You can use this method when you know the lengths of all three sides of a triangle.

Heron's formula

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

where

$$s = \frac{a+b+c}{2}$$

The default values, which you will use to debug and test this VI, are Side A = 6, Side B = 8, Side C = 10. Therefore the correct values are as follows:

$$s = \frac{6+8+10}{2} = 12$$

$$A = \sqrt{12 \times 6 \times 4 \times 2} = 24$$

You might want to refer to this calculation as you debug the VI.

Complete the following sections to identify and fix edit-time and run-time issues. Use single-stepping and execution highlighting to step through the VI. Use breakpoints and probes to determine if the calculations are correct and figure out where an error originates.

Edit-Time Errors

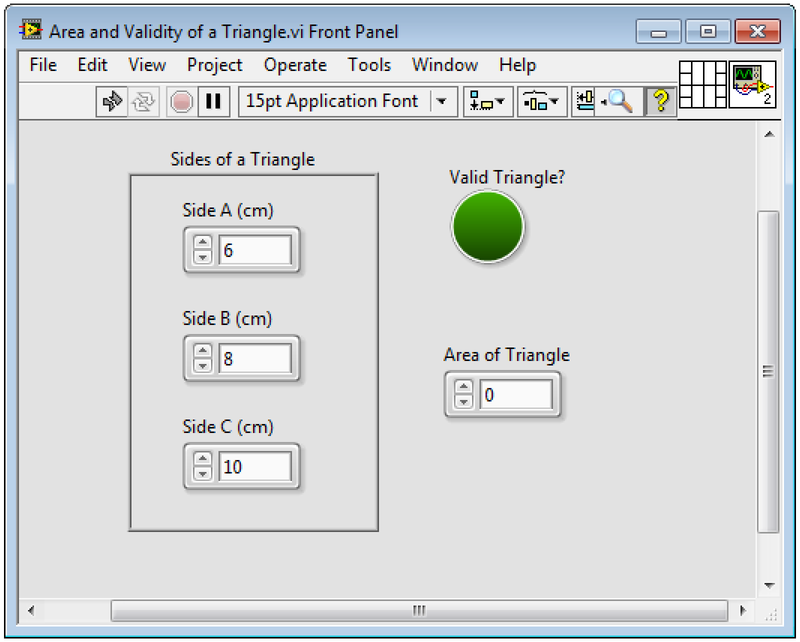
Locate and correct errors that prevent the VI from running.

1. Open and examine the area and validity of a triangle VI.

   □ Open `Debug.lvproj` in the `<Exercises>\LabVIEW Core 1\Debugging` directory.

   □ Open **Area and Validity of a Triangle VI** from the **Project Explorer** window.

Figure 3-1. Area and Validity of a Triangle VI Front Panel



Side A (cm) 6

Side B (cm) 8

Side C (cm) 10

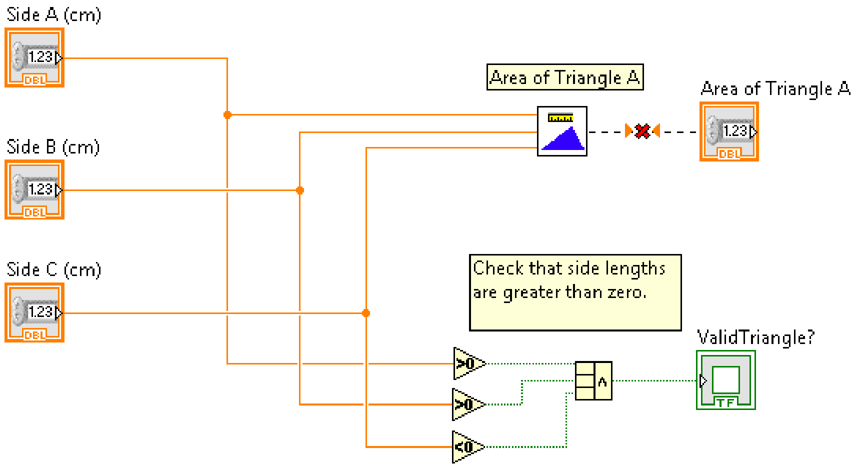
Area of Triangle 0

   □ Notice the **Run** button on the toolbar appears broken, indicating that the VI is broken and cannot run.

2. Display and examine the block diagram of Area and Validity of a Triangle VI shown in Figure 3-2.

   This VI takes input values for each of the three sides of a triangle, passes the values into a subVI that determines the area, and checks that the values entered are valid for a triangle.

Figure 3-2. Area and Validity of a Triangle VI Block Diagram



3. Find and fix each error.

   - ☐ Click the broken **Run** button to display the Error list window, which lists all the errors.
   - ☐ Select an error description in the Error list window. The **Details** section describes the error and in some cases recommends how to correct the error.
   - ☐ Click the **Help** button to display a topic in the *LabVIEW Help* that describes the error in detail and includes step-by-step instructions for correcting the error.
   - ☐ Click the **Show Error** button or double-click the error description to highlight the area on the block diagram that contains the error.
   - ☐ Use the Error list window to fix each error.

   Tip For errors in the Area of a Triangle subVI, double-click to open it. In the Area of Triangle VI, notice that the formula for calculating the area of a triangle requires the sum of the sides be divided by 2. Right-click the **y** input of the Divide function and select **Create»Constant** and enter a value of `2`.

4. Save both VIs.

Run-Time Errors

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

1. Test the VI.

   □ Display the front panel by clicking it or by selecting **Window»Show Front Panel**.

   □ Use the default values for each side. These values are valid measurements for a triangle.

   □ Run the VI.

   □ Notice that although the numbers you entered are valid, the LED is not illuminated and the Area of a Triangle indicator displays NaN.

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

   □ Display the block diagram.

   □ Click the **Highlight Execution** button on the toolbar to enable execution highlighting.

   □ Click the **Retain Wire Values** button on the toolbar so you can see the last value passed on a wire.

   □ Run the VI.

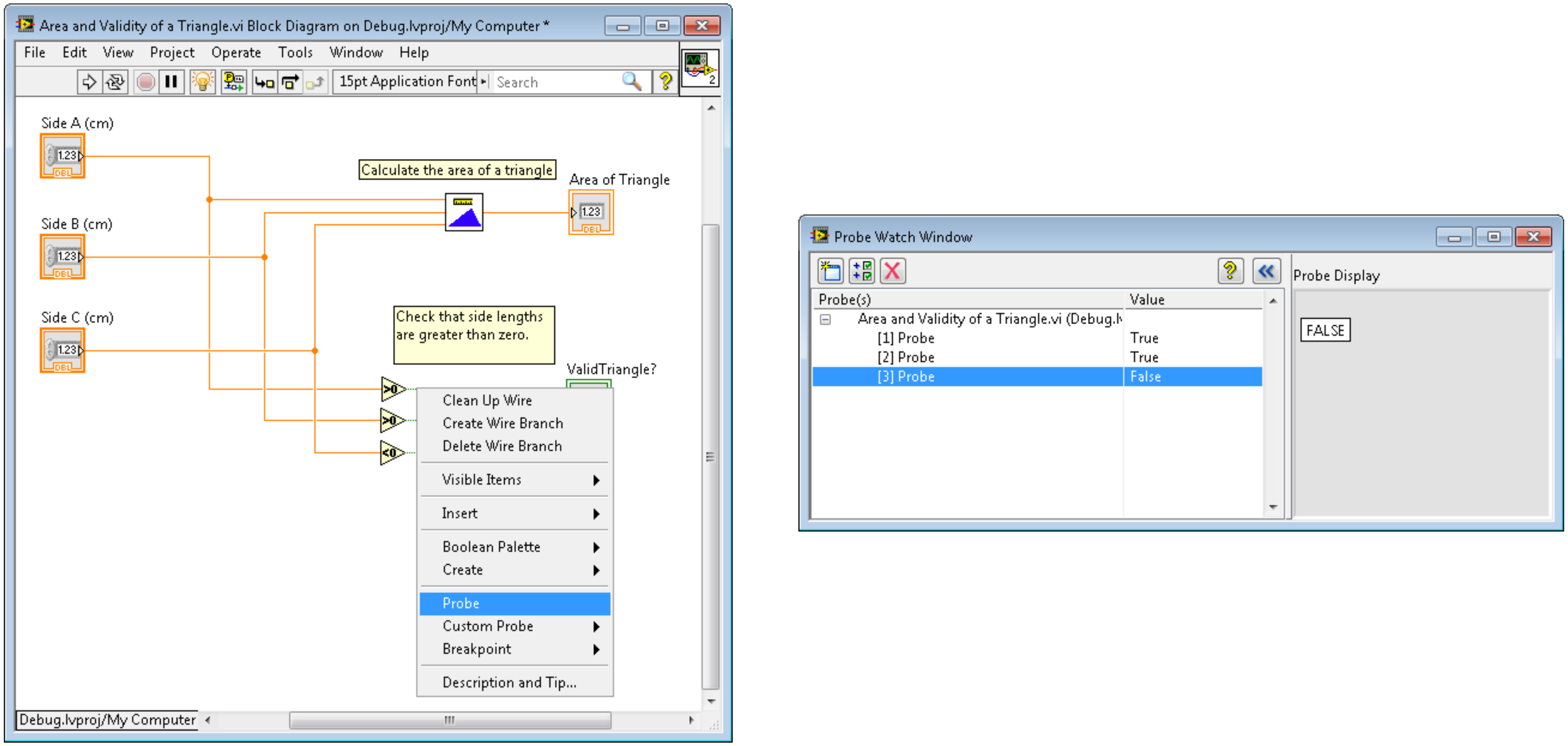
   Notice that you can see how data flows through the wires. At the output of each node, you can see the data value displays momentarily. Because you have enabled the Retain Wire Values button, you can probe the last value in the wire.

3. Probe the wire values.

   □ Right-click each of the input wires to the Compound Arithmetic Function and select **Probe**. This displays the Probe Watch Window.

   □ Notice that one of the wire values is **`False`** as shown in Figure 3-3.

Figure 3-3. Probe Wires



Probe(s)	Value
Area and Validity of a Triangle.vi (Debug.l\	
[1] Probe	True
[2] Probe	True
[3] Probe	False

   □ Because you are checking to see that all three sides of the triangle have positive lengths, either the input value is invalid or the logic is incorrect. The input values were all positive numbers, so that means the logic is incorrect.

   Notice that the node returning a value of **`False`** is a Less than Zero? function, but this section of code should be checking to see if the value is greater than zero.

   □ Right-click the **Less than Zero?** function and select **Replace»Comparison Palette»Greater than Zero?**.

4. Test the VI.

   □ Run the VI.

   □ Notice that all the probe values are all `True`.

   □ Display the front panel. Notice that the Valid Triangle? LED is illuminated, but the Area of Triangle indicator is still returning NaN.

   □ The area of the triangle is calculated in the subVI, so you must continue debugging in the Area of a Triangle subVI.

5. Continue debugging the subVI.

   □ Display the block diagram of the Area and Validity of a Triangle VI.

   □ Click the **Step Into** button to start single-stepping through the VI. Execution highlighting shows the flow of data on the block diagram from one node to another. Nodes blink to indicate they are ready to execute.

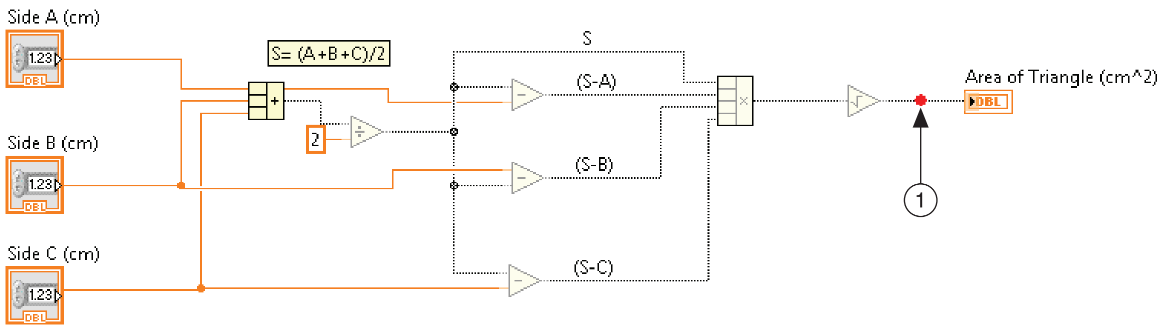
   □ Click the **Step Over** button after each node to step through the entire block diagram until you get to the subVI. Each time you click the **Step Over** button, the current node executes and pauses at the next node.

   □ When you get to the subVI, click the **Step Into** button to open the block diagram of the Area of Triangle subVI. The subVI is paused.

   □ Turn on Execution Highlighting and Retain Wire Values in the subVI.

□ Right-click the output of the Square Root function and select **Breakpoint»Set Breakpoint** as shown in Figure 3-4.

Figure 3-4. Set Breakpoint



1 The breakpoint stops the VI after the Square Root node executes and before the value is output to the Area of Triangle indicator.

6. Click the red pause button to resume the execution of the VI.

   □ The VI continues executing until the breakpoint and then pauses again.

7. Examine the values on the wires

   □ Move the cursor to hover over the input wire of the Square Root function. You should see a tip strip with a value of -576. You cannot take the square root of a negative number, which is why the Area of Triangle indicator returns NaN.

Tip If you cannot see the tip strip, you can click the wire to open the Probe Watch window to see the value.

   □ Hover over other wires or use the Probe Watch window to examine other intermediate values.

   □ Notice that the value on the (S-B) wire is also a negative number. If you look more closely, you notice that the inputs for the subtract function are reversed.

   □ Click the **Abort** button to stop the VI.

   □ Switch the inputs for the (S-B) Subtract function.

Tip Press <Ctrl> and click one of the inputs to switch them. When you press <Ctrl> and hover over an input, you see the cursor change.

8. Save the VI.

9. Test the Area of Triangle VI.

   □ Run the VI again.

   □ Check the intermediate values as the VI runs or hover over the wires after it pauses at the breakpoint and verify that the values returned are correct. The square root function should return a value of 24.

   □ Right-click the breakpoint and select **Breakpoint»Clear Breakpoint**.

   □ Turn off execution highlighting in the Area of Triangle VI and the Area and Validity of a Triangle VI.

   □ Save both VIs.

Test

1. Test the Area and Validity of a Triangle VI using the values for Side A, Side B, and Side C in Table 3-1. For each set of test values, record the area you get when you run the VI.

Table 3-1. Area and Validity of a Triangle Test Values

Side A	Side B	Side C	Area
24	30	18	
12	12	12	
15	20	25	

Table 3-2. Area and Validity of a Triangle Test Values - Area Answers

Side A	Side B	Side C	Area
24	30	18	**216**
12	12	12	**62.35**
15	20	25	**150**

2. Save and close the VI when you are finished testing.

End of Exercise 3-1

C. Error Handling

Objective: Describe the difference between automatic and manual error handling.

Error Handling Review

Error Handling Anticipation, detection, and resolution of warnings and errors

Automatic Error Handling At run time, LabVIEW suspends execution, highlights node where error occurred, and displays Error dialog box

Manual Error Handling You control when dialog boxes appear, propagate errors through error in/error out clusters, terminate error chain with Simple Error Handler

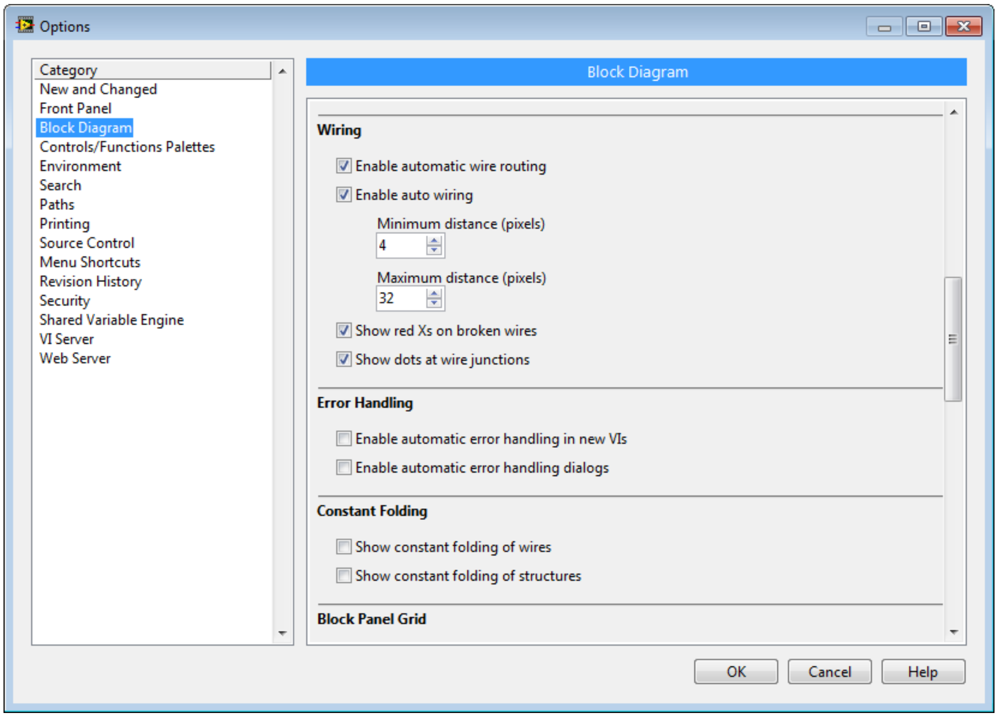
Demonstration: Automatic vs. Manual Error Handling

- Manual error handling—Use Simple Error Handler to display errors
- Automatic error handling—Highlights the node that caused the error

Enabling automatic error handling does not override manual error handling. If the error cluster is wired and the VI uses the Simple Error Handler, then LabVIEW defaults to manual error handling.

Disable Automatic Error Handling

To ensure that LabVIEW doesn't implement automatic error handling by default, disable it for all new VIs. Change the configuration settings in the **Tools»Options** dialog box.



Wiring

☑ Enable automatic wire routing

☑ Enable auto wiring

Minimum distance (pixels) 4

Maximum distance (pixels) 32

☑ Show red Xs on broken wires

☑ Show dots at wire junctions

Error Handling

☐ Enable automatic error handling in new VIs

☐ Enable automatic error handling dialogs

Constant Folding

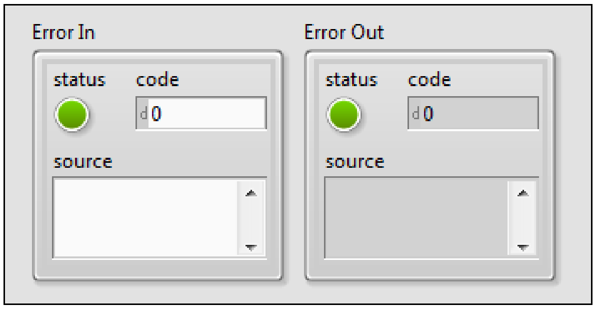
☐ Show constant folding of wires

☐ Show constant folding of structures

Block Panel Grid

Error Clusters

VIs and functions return errors in one of two ways—with numeric error codes or with an error cluster. Typically, functions use numeric error codes, and VIs use an error cluster, usually with error inputs and outputs. Use the error cluster controls and indicators to create error inputs and outputs in subVIs.



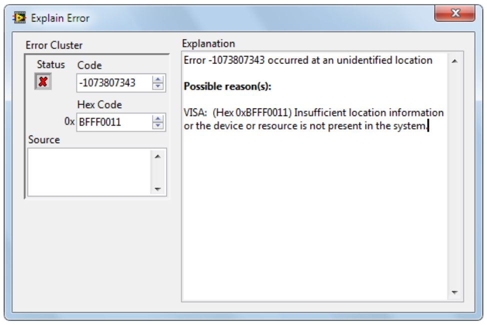
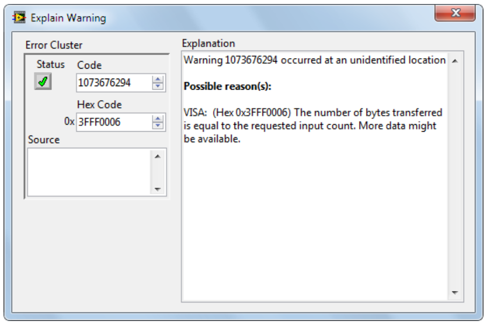
Error In
status ● code d0
source

Error Out
status ● code d0
source

1 **status**—a Boolean value that reports TRUE if an error occurs
2 **code**—a 32-bit signed integer that identifies the error numerically. A non-zero error code is coupled with a status of FALSE signals a warning rather than an error
3 **source**—a string that identifies where the error occurred

Errors and Warnings

When an error occurs, open the **Explain Error** dialog box to see more information about the error.

Errors	Warnings
 Explain Error Error Cluster Status ☒ Code -1073807343 Hex Code 0x BFFF0011 Source Explanation Error -1073807343 occurred at an unidentified location **Possible reason(s):** VISA: (Hex 0xBFFF0011) Insufficient location information or the device or resource is not present in the system. Status = TRUE Code = Non-zero	 Explain Warning Error Cluster Status ✓ Code 1073676294 Hex Code 0x 3FFF0006 Source Explanation Warning 1073676294 occurred at an unidentified location **Possible reason(s):** VISA: (Hex 0x3FFF0006) The number of bytes transferred is equal to the requested input count. More data might be available. Status = FALSE Code = Non-zero
More severe	Less severe
Passed to the next node without executing that part of the code	Node executes normally, but it is important to monitor warnings during development to ensure proper behavior of your application

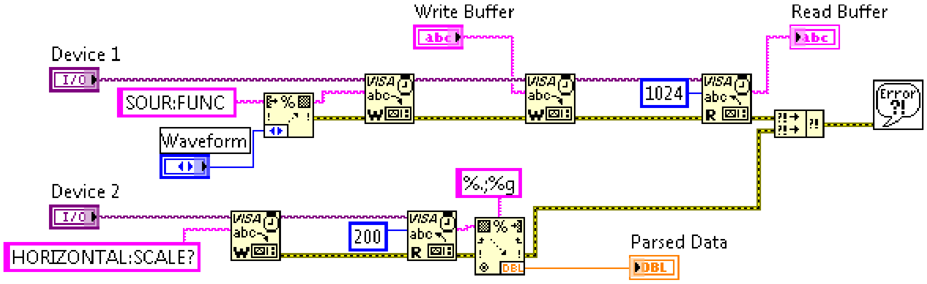
Merge Errors

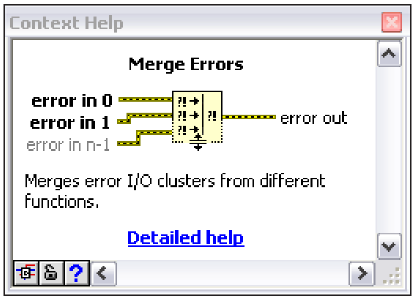
The Merge Errors Function:

- Returns the first error found. If no error is found, it returns the first warning.
- Does not concatenate errors.

Use Merge Errors to

- Propagate errors along wires.
- Merge errors from different wire paths.





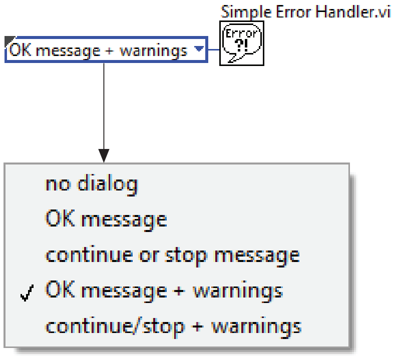
Merge Errors

Merges error I/O clusters from different functions.

At the end of your application after all error sources are merged into one error cluster, you must report errors to the user using the Simple Error Handler VI or another error reporting mechanism.

Errors and Warnings Recommendations

By default, the Simple Error Handler VI displays a dialog with a description of any errors that occurred and does not report warnings. However, the Simple Error Handler VI can be configured for other error handling behavior.



Additional Resources

Learn More About	LabVIEW Help Topic
Correcting Broken VIs	*Error List Window*
	Correcting Broken VIs
	Using Wires to Link Block Diagram Objects
Debugging	*Debugging Techniques*
	Execution Highlighting
	Single-Stepping through a VI
	Using the Probe Tool
	Probe Watch Window
	Managing Breakpoints
	Breakpoint Manager Window
	Preventing Undefined Data
Error Handling	*Disabling Debugging Tools*
	Using Error Clusters
	Error Codes and Messages
	Merge Error Function
	Displaying Warnings
	Simple Error Handler VI
	Handling Errors

Activity 3-2: Lesson Review

Answer the following questions and then discuss your answers with the class.

1. Which of the following will result in a broken run arrow?
   a. A subVI is broken
   b. The diagram includes a divide by zero
   c. A required subVI input is unwired
   d. A Boolean terminal is wired to a numeric indicator

2. Which of the following are components and data types of the error cluster?
   a. Status: Boolean
   b. Error: String
   c. Code: 32-bit integer
   d. Source: String

3. All errors have negative error codes and all warnings have positive error codes.
   a. True
   b. False

4. Merge Errors function concatenates error information from multiple sources.
   a. True
   b. False

Activity 3-2: Lesson Review - Answers

1. Which of the following will result in a broken run arrow?
   - **a. A subVI is broken**
   - b. The diagram includes a divide by zero
   - **c. A required subVI input is unwired**
   - **d. A Boolean terminal is wired to a numeric indicator**

2. Which of the following are components and data types of the error cluster?
   - **a. Status: Boolean**
   - b. Error: String
   - **c. Code: 32-bit integer**
   - **d. Source: String**

3. All errors have negative error codes and all warnings have positive error codes.
   - a. True
   - **b. False**

4. Merge Errors function concatenates error information from multiple sources.
   - a. True
   - **b. False**

4 Using Loops

In this lesson you will learn to recognize the different components of a LabVIEW loop structure and how to apply a For Loop or a While Loop appropriately.

Topics

- Loops Review
- While Loops
- For Loops
- Timing a VI
- Data Feedback in Loops
- Plotting Data

Exercises

Exercise 4-1	Pass Data Through Tunnels
Exercise 4-2	Calculating Average Temperature
Exercise 4-3	Temperature Monitor VI—Plot Multiple Temperatures

A. Loops Review

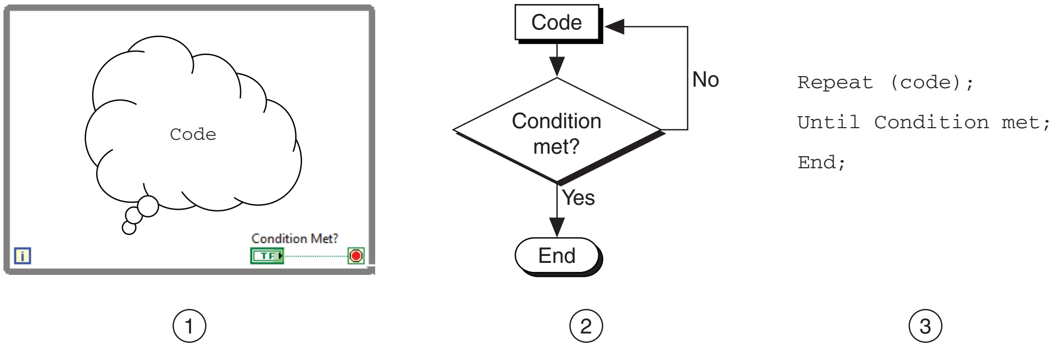
Objective: Recognize loop structures and explain how to use them.

While Loops—Review

- Similar to Do Loop or a Repeat-Until Loop
- Repeats code segment until a condition is met
- Always execute at least once
- Iteration terminal—output that contains the number of complete iterations

i

- Iteration terminal always starts at zero
- Conditional terminal sets the condition for stopping the loop



```
Repeat (code);
Until Condition met;
End;
```

1	LabVIEW While Loop	2	Flowchart	3	Pseudo Code

For Loops—Review

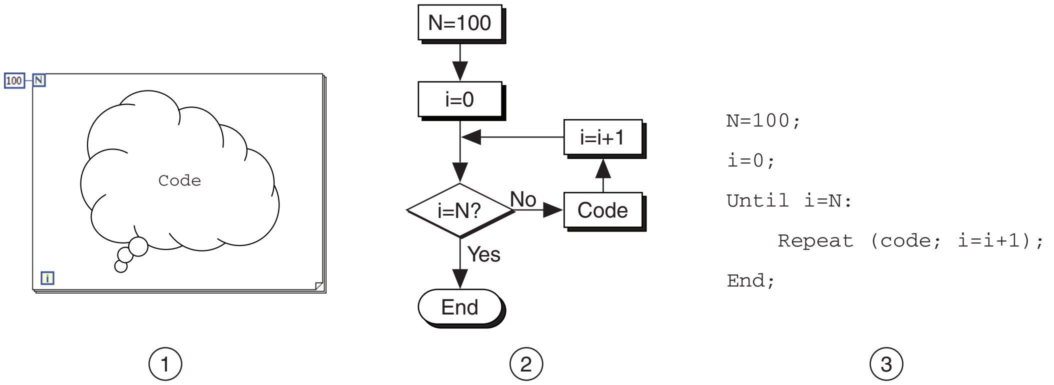
- Repeats code a certain number of times
- The value in the count terminal (an input terminal) indicates how many times to repeat the subdiagram in the For Loop.

N

- Can execute zero times
- Iteration terminal—output that contains the number of complete iterations

i

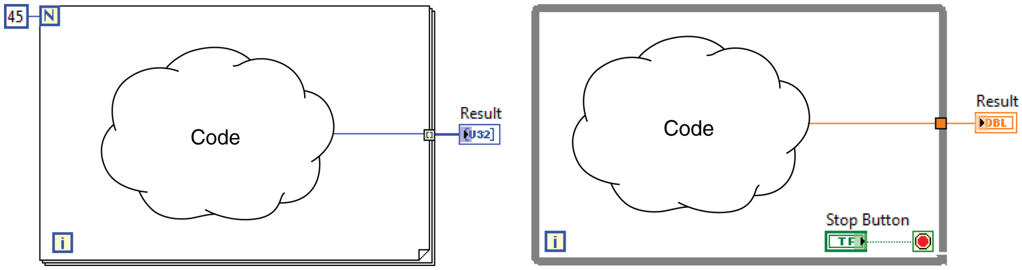
- Iteration terminal always starts at zero



```
N=100;
i=0;
Until i=N:
    Repeat (code; i=i+1);
End;
```

1 LabVIEW For Loop 2 Flowchart 3 Pseudo Code

While Loop/For Loop Comparison



While Loop	For Loop
Stops executing only if the value at the conditional terminal meets the condition	Executes a set number of times
Must execute at least once	Can execute zero times
Tunnels automatically output the last value	Tunnels automatically output an array of data

Activity 4-1: While Loops vs. For Loops

Goal

Determine when to use a While Loop and when to use a For Loop.

Description

For the following scenarios, decide whether to use a While Loop or a For Loop.

Scenario 1

Acquire pressure data in a loop that executes once per second for one minute.

1. If you use a While Loop, what is the condition that you need to stop the loop?
2. If you use a For Loop, how many iterations does the loop need to run?
3. Is it easier to implement a For Loop or a While Loop?

Scenario 2

Acquire pressure data until the pressure is greater than or equal to 1400 psi.

1. If you use a While Loop, what is the condition that you need to stop the loop?
2. If you use a For Loop, how many iterations does the loop need to run?
3. Is it easier to implement a For Loop or a While Loop?

Scenario 3

Acquire pressure and temperature data until both values are stable for two minutes.

1. If you use a While Loop, what is the condition that you need to stop the loop?
2. If you use a For Loop, how many iterations does the loop need to run?
3. Is it easier to implement a For Loop or a While Loop?

Scenario 4

Output a voltage ramp starting at zero, increasing incrementally by 0.5 V every second, until the output voltage is equal to 5 V.

1. If you use a While Loop, what is the condition that you need to stop the loop?
2. If you use a For Loop, how many iterations does the loop need to run?
3. Is it easier to implement a For Loop or a While Loop?

Activity 4-1: While Loops vs. For Loops - Answers

Scenario 1

Acquire pressure data in a loop that executes once per second for one minute.

1. If you use a While Loop, what is the condition that you need to stop the loop?

   While Loop: Time = 1 minute

2. If you use a For Loop, how many iterations does the loop need to run?

   For Loop: 60 iterations

3. Is it easier to implement a For Loop or a While Loop?

   Both are possible

Scenario 2

Acquire pressure data until the pressure is greater than or equal to 1400 psi.

1. If you use a While Loop, what is the condition that you need to stop the loop?

   While Loop: Pressure = 1400 psi

2. If you use a For Loop, how many iterations does the loop need to run?

   For Loop: unknown

3. Is it easier to implement a For Loop or a While Loop?

   A While Loop. Although you can add a conditional terminal to a For Loop, you still need to wire a value to the count terminal. Without more information, you do not know the appropriate value to wire to the count terminal.

Scenario 3

Acquire pressure and temperature data until both values are stable for two minutes.

1. If you use a While Loop, what is the condition that you need to stop the loop?

   While Loop: [(Last Temperature = Previous Temperature) for 2 minutes or more] and [(Last Pressure = Previous Pressure) for 2 minutes or more]

2. If you use a For Loop, how many iterations does the loop need to run?

   For Loop: unknown

3. Is it easier to implement a For Loop or a While Loop?

   A While Loop. Although you can add a conditional terminal to a For Loop, you still need to wire a value to the count terminal. Without more information, you do not know the appropriate value to wire to the count terminal.

Scenario 4

Output a voltage ramp starting at zero, increasing incrementally by 0.5 V every second, until the output voltage is equal to 5 V.

1. If you use a While Loop, what is the condition that you need to stop the loop?

   While Loop: Voltage = 5 V

2. If you use a For Loop, how many iterations does the loop need to run?

   For Loop: 11 iterations (Including the two end points, count the iteration for each value - 0, 0.5, 1.0, 1.5, ... 4.5, 5.0.)

3. Is it easier to implement a For Loop or a While Loop?

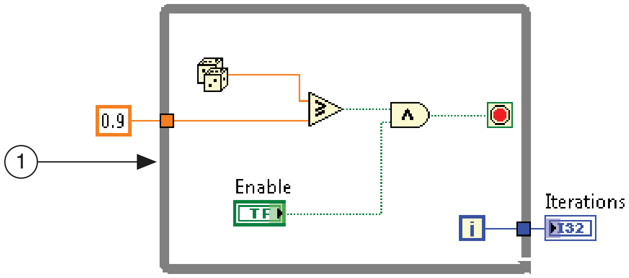
   Both are possible.

B. While Loops

Objective: Recognize tunnels and explain their purpose on a loop structure and demonstrate how to use error checking and error handling inside a loop.

Tunnels

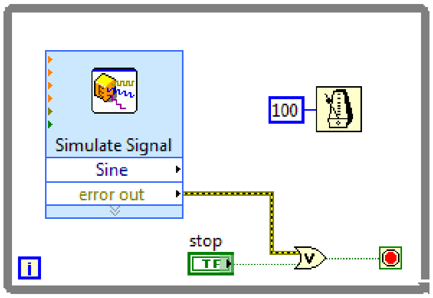
Tunnels transfer data into and out of structures. Data pass out of a loop after the loop terminates.



1 The loop executes only after data arrive at the tunnel.

Error Checking and Error Handling

You can wire an error cluster to the conditional terminal to stop the iteration of the loop. If you wire the error cluster to the conditional terminal, only the TRUE or FALSE value of the **status** parameter of the error cluster passes to the terminal. If an error occurs, the loop stops.



1 Use the error cluster and a stop button to determine when to stop the loop.

Exercise 4-1 Pass Data Through Tunnels

Goal

Use a While Loop and an iteration terminal and pass data through a tunnel.

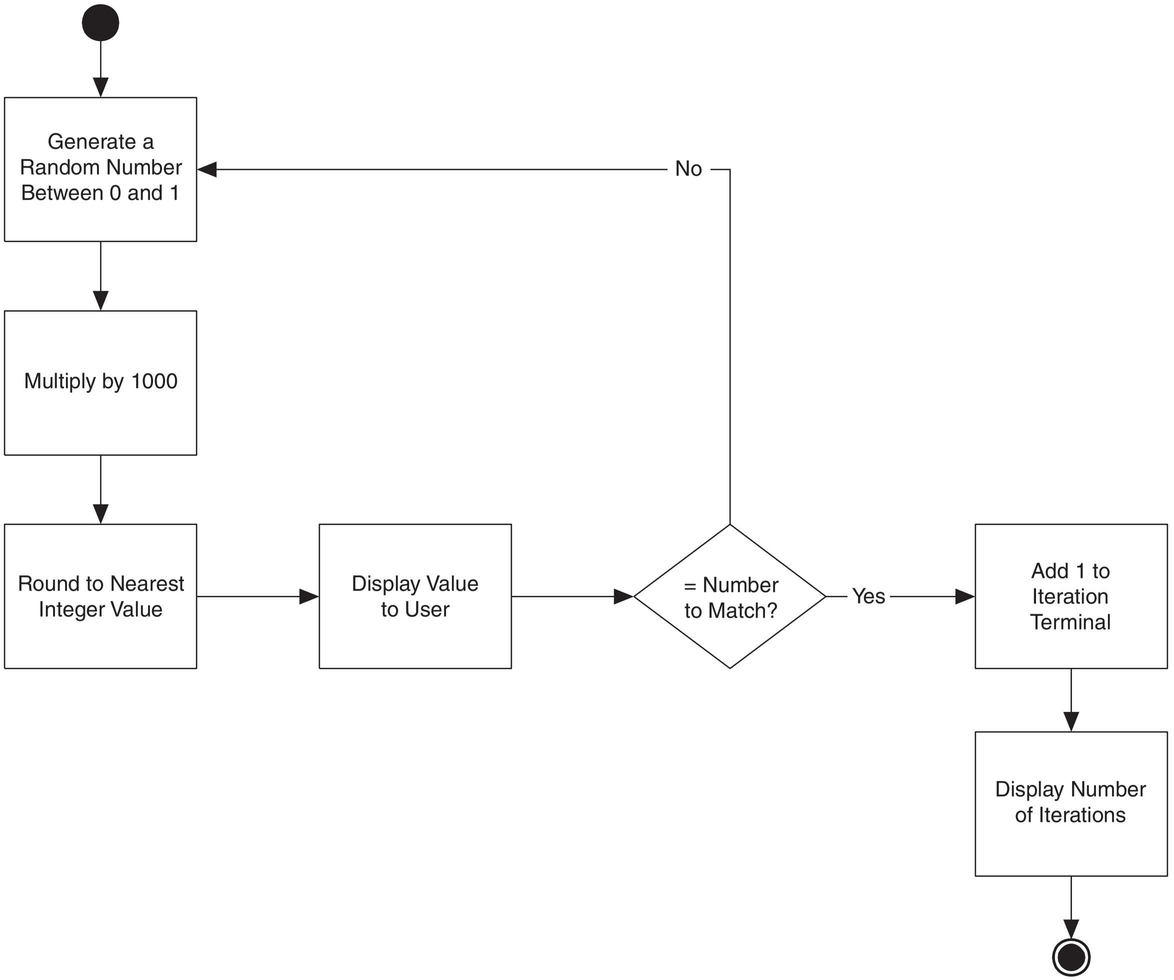
Scenario

Create a VI that continuously generates random numbers between 0 and 1000 until it generates a number that matches a number selected by the user. Determine how many random numbers the VI generated before generating the matching number.

Design

Use the following flowchart and input/output list to create the VI for this exercise. The flowchart in Figure 4-1 illustrates the data flow for this design.

Figure 4-1. Auto Match Flowchart


Generate a Random Number Between 0 and 1
Multiply by 1000
Round to Nearest Integer Value
Display Value to User
= Number to Match?
No
Yes
Add 1 to Iteration Terminal
Display Number of Iterations

Inputs and Outputs

The following table describes the inputs and outputs for this exercise.

Table 4-1. Auto Match VI Inputs and Outputs

Type	Name	Properties
Numeric control	`Number to Match`	Double-precision, floating-point between 0 and 1000, coerce to nearest whole number, default value = 50
Numeric indicator	`Current Number`	Double-precision, floating-point
Numeric indicator	`Number of Iterations`	Integer

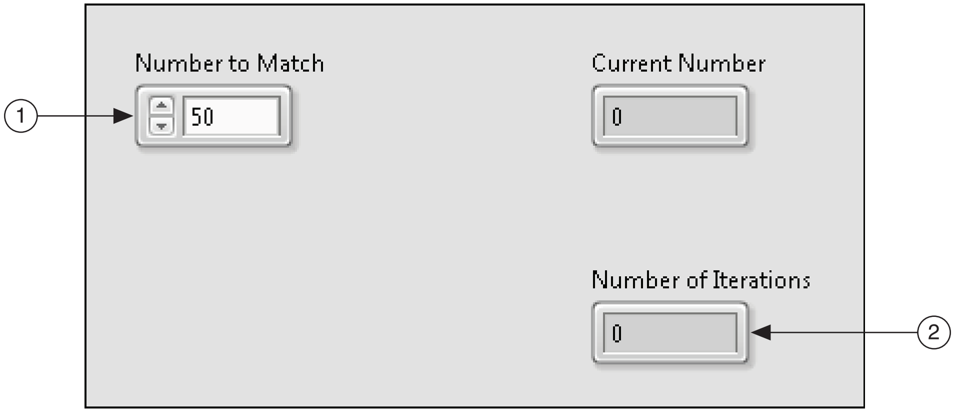
Implementation

1. Create a blank project and save it as `Auto Match.lvproj` in the `<Exercises>LabVIEW Core 1\Auto Match` directory.

2. Create a new VI in the project and save it as `Auto Match.vi` in the same directory as the project.

3. Build the front panel shown in Figure 4-2.

Figure 4-2. Auto Match VI Front Panel

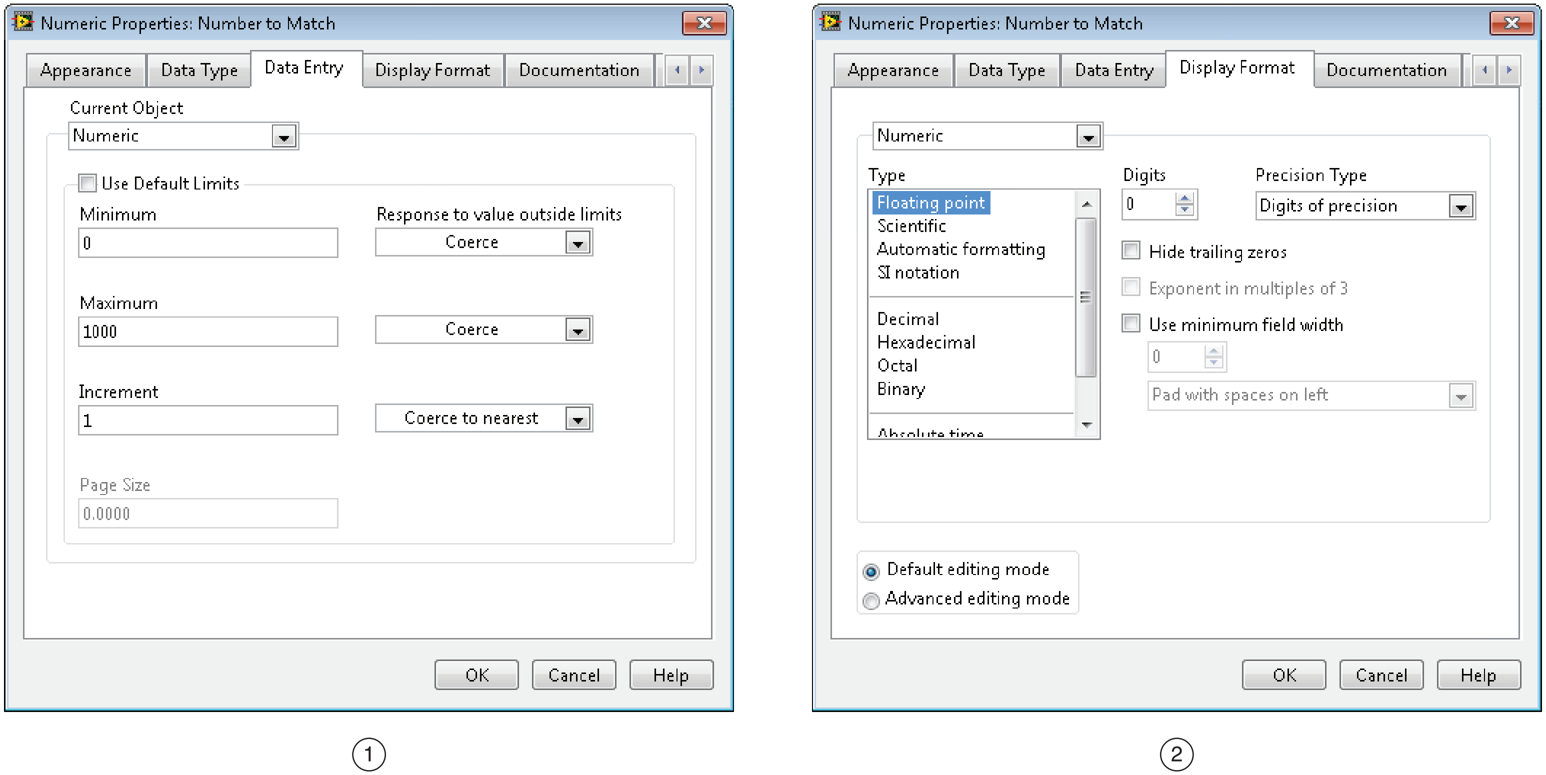


1 Set the default value of the Number to Match control to 50—Enter `50` in the **Number to Match** control and then right-click the control and select **Data Operations»Make Current Value Default**.
2 Set Number of Iterations indicator to output a signed, long integer—Right-click the indicator and select **Representation»I32**.

4. Set the properties for the **Number to Match** control so that the data type is a 32-bit unsigned integer, the data range is from 0 to 1000, the increment value is 1, and the digits of precision is 0.

   - ☐ Right-click the **Number to Match** control and select **Representation»U32** from the shortcut menu.
   - ☐ Right-click the **Number to Match** control and select **Data Entry** from the shortcut menu. Set the properties on the **Data Entry** and **Display Format** tabs as shown in Figure 4-3.

Figure 4-3. Number to Match Numeric Properties



Numeric Properties: Number to Match

Current Object: Numeric

☐ Use Default Limits

Minimum	Response to value outside limits
0	Coerce
Maximum	
1000	Coerce
Increment	
1	Coerce to nearest
Page Size	
0.0000	

Numeric Properties: Number to Match

Numeric

Type: Floating point

Digits: 0

Precision Type: Digits of precision

☐ Hide trailing zeros

☐ Exponent in multiples of 3

☐ Use minimum field width

0

Pad with spaces on left

● Default editing mode

○ Advanced editing mode

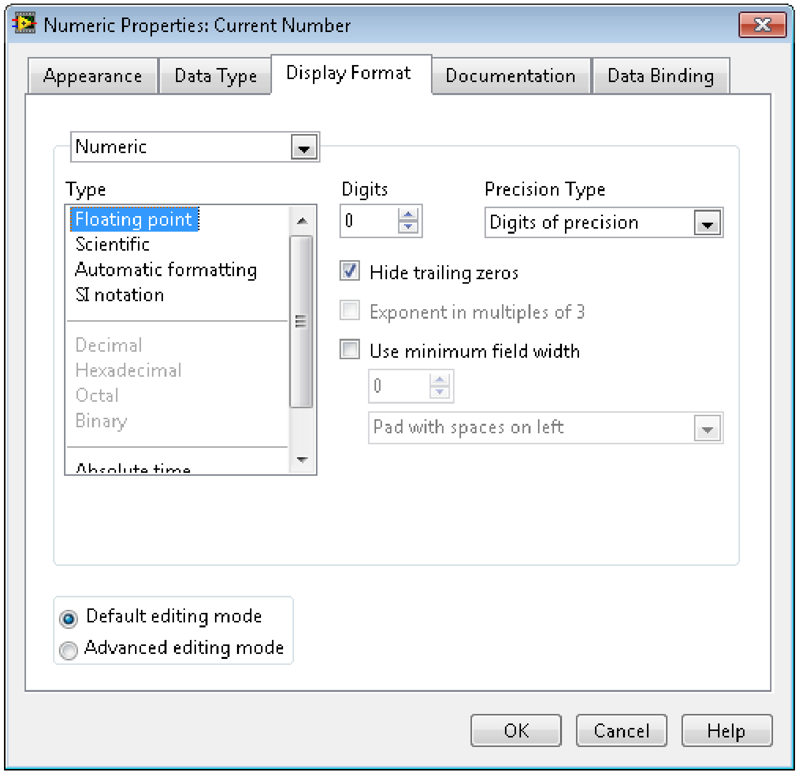
1 **Number to Match**—Data Entry Properties	2 Number to Match—Display Format Properties

5. Set the representation of the **Current Number** indicator to an unsigned, 32-bit integer and set the digits of precision for the **Current Number** output to 0.

   □ Right-click the **Current Number** indicator and select **Representation»U32** from the shortcut menu.

   □ Right-click the **Current Number** indicator and select **Display Format**. Set the properties as shown in Figure 4-4.

Figure 4-4. Current Number Indicator Display Format Properties



Numeric Properties: Current Number

Appearance | Data Type | Display Format | Documentation | Data Binding

Numeric

Type: Floating point

Digits: 0

Precision Type: Digits of precision

☑ Hide trailing zeros

□ Exponent in multiples of 3

□ Use minimum field width

0

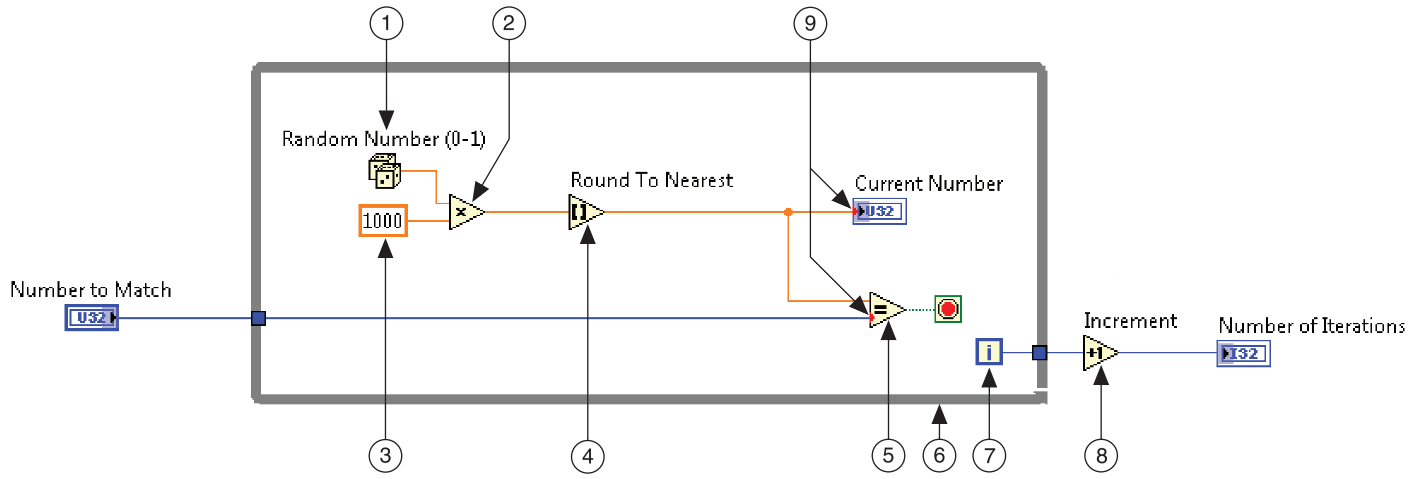
Pad with spaces on left

● Default editing mode

○ Advanced editing mode

6. Create the block diagram shown in Figure 4-5.

Figure 4-5. Auto Match VI Block Diagram



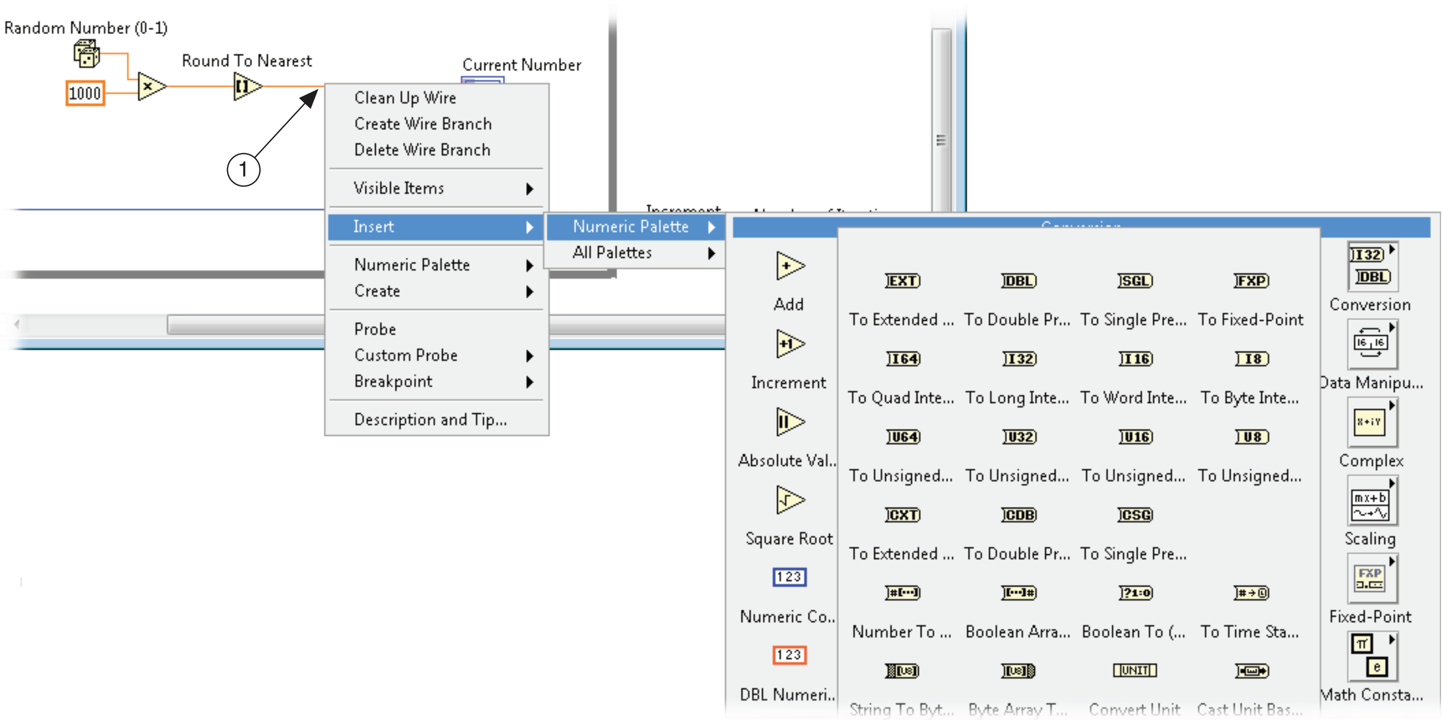
1 **Random Number (0–1)**—Generates a random number between 0 and 1.
2 **Multiply**—Multiplies the random number by the **y** input to produce a random number between 0 and y.
3 **Numeric Constant**—Right-click the **y** input of the Multiply function and select **Create»Constant**. Enter a value of `1000`. Because the Random Number (0-1) function generates a double-precision, floating point number between 0 and 1, multiplying the number by 1000 produces a range of numbers between 0 and 1000.
4 **Round To Nearest**—Rounds the random number to the nearest integer.
5 **Equal?**—Compares the random number with Number to Match and returns FALSE if the numbers are not equal. Otherwise, it returns TRUE.
6 **While Loop**—Repeats the algorithm until the Equal? function returns TRUE because the Equal? function is wired to the conditional terminal, which is set to Stop if True.
7 Iteration terminal—Each time the loop executes, the iteration terminal increments by one.
8 **Increment**—Adds 1 to the While Loop count because the iteration starts at 0.
9 Coercion dots—Red coercion dots appear on block diagram nodes when you connect a wire of one numeric type to a terminal of a different numeric type. In this case, the output from Round To Nearest is a double-precision, floating point but Current Number is an integer.

Tip Coercion dots can cause a VI to use more memory and increase its run time, so try to keep data types consistent in the VIs you create.

7. Update the VI to remove the coercion dots.

   □ Right-click the wire coming from the Round To Nearest function and select **Insert»Numeric Palette»Conversion»To Unsigned Long Integer** as shown in Figure 4-6. This inserts the To Unsigned Long Integer function on the wire.

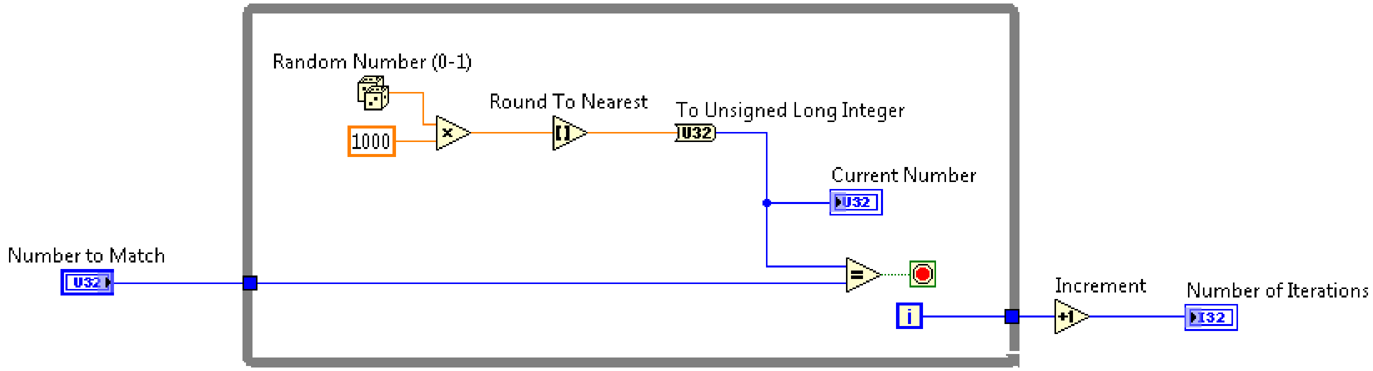
Figure 4-6. Inserting the To Unsigned Long Integer Function on a Wire



1 Right-click the wire to display the shortcut menu.

8. Notice that converting the output from the Round To Nearest function removes all the coercion dots on the block diagram, as shown in Figure 4-7.

Figure 4-7. Completed Auto Match VI



9. Display the front panel.

10. Right-click the **Current Number** indicator and select **Advanced»Synchronous Display**.

Note If synchronous display is enabled, then every time the block diagram sends a value to the **Current Number** indicator, the block diagram stops executing until the front panel has updated the value of the indicator. In this exercise, you enable the synchronous display, so you can see the **Current Number** indicator get updated repeatedly on the front panel. Typically, the synchronous display is disabled to increase execution speed since you usually do not need to see every single updated value of an indicator on the front panel.

11. Save the VI.

Test

1. Change the number in the **Number to Match** control to a number that is in the data range, which is 0 to 1000 with an increment of 1.

2. Run the VI.

3. Change the **Number to Match** value and run the VI again. Current Number updates at every iteration of the loop because it is inside the loop. Number of Iterations updates upon completion because it is outside the loop.

4. To see how the VI updates the indicators, enable execution highlighting.

   □ On the block diagram toolbar, click the **Highlight Execution** button to enable execution highlighting.

5. Run the VI and observe the data flow.

6. Turn off execution highlighting to quickly finish executing the VI.

7. Try to match a number that is outside the data range.

   □ Change the **Number to Match** value to a number that is out of the data range, 0 - 1000.

   □ Run the VI.

   □ Notice LabVIEW coerces the out-of-range value to the nearest value in the data range you specified in step 4 of the Implementation section.

8. Close the VI.

End of Exercise 4-1

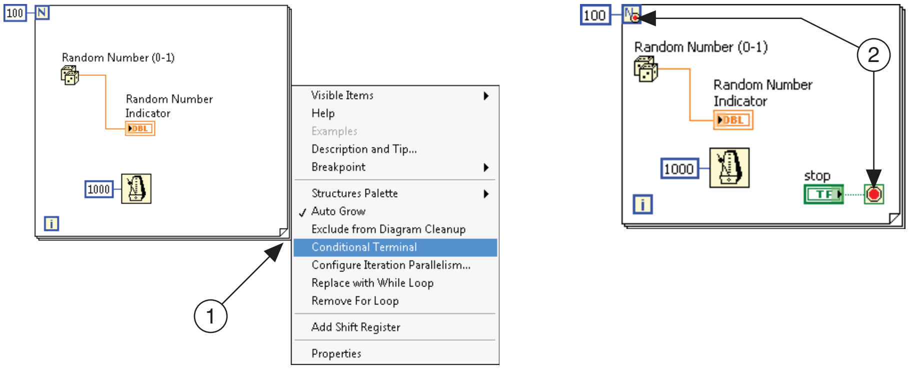
C. For Loops

Objective: Demonstrate how to add a conditional terminal to a For Loop and describe how numeric conversion occurs on the For Loop count terminal.

Conditional Terminal

You can add a conditional terminal to configure a For Loop to stop when a Boolean condition or an error occurs. A For Loop with a conditional terminal executes until the condition occurs or until all iterations are complete, whichever happens first.

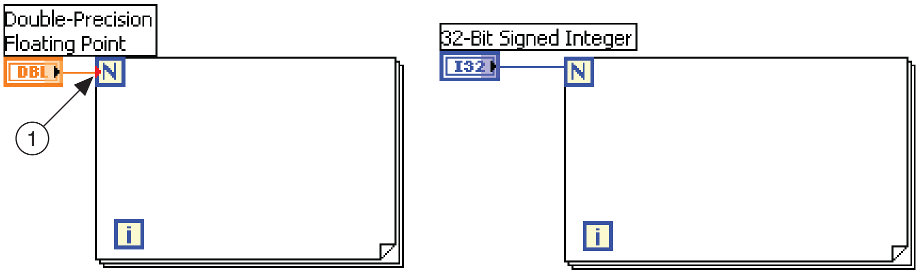
The following For Loop generates a random number every second until 100 seconds has passed or until the user clicks the **stop** button.



1 Right-click the loop border and select Conditional Terminal from the shortcut menu.
2 Red glyph appears in the count terminal and a conditional terminal in the lower right corner.

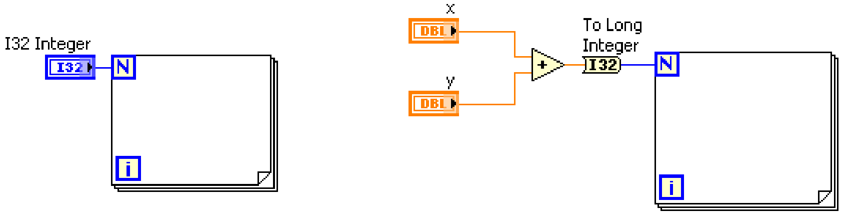
Count Terminal Numeric Conversion

If you wire a double-precision, floating-point numeric value to the count terminal, LabVIEW converts the numeric value to a 32-bit signed integer.



1 Coercion Dot

For better performance, avoid coercion by using matching data types or programmatically converting to matching data types.

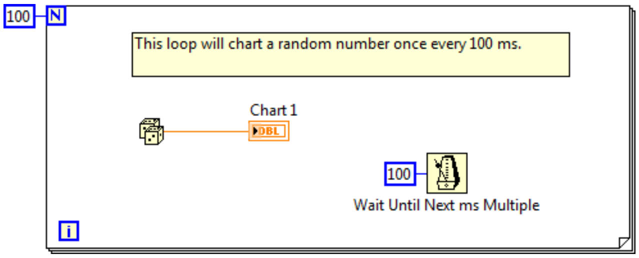


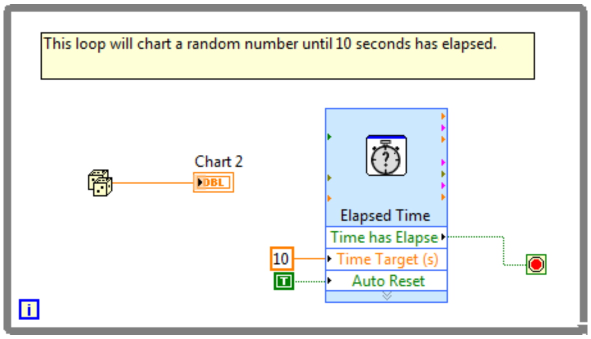
D. Timing a VI

Objective: Identify scenarios that require loop timing and apply the appropriate function

Why Do You Need Timing in a VI?

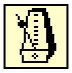
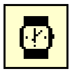
When a loop finishes executing an iteration, it immediately begins executing the next iteration, unless it reaches a stop condition. If you are acquiring data, and you want to acquire the data once every 10 seconds, you need a way to time the loop iterations so they occur once every 10 seconds. You also want to time a loop to provide the processor with time to complete other tasks, such as processing the user interface.


This loop will chart a random number once every 100 ms.


This loop will chart a random number until 10 seconds has elapsed.

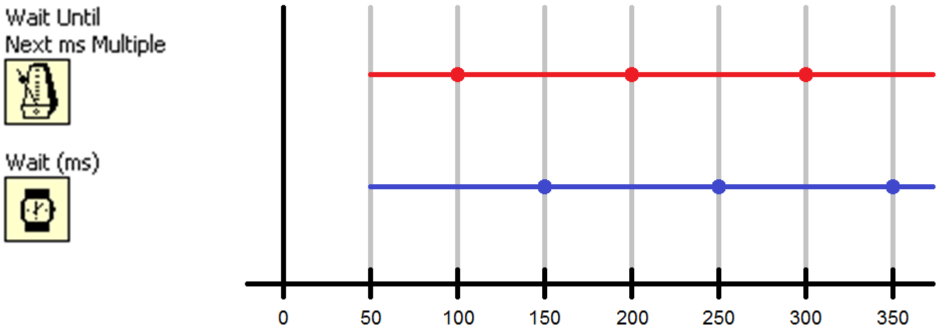
Wait Functions Inside a Loop

Use a wait function inside a loop to accomplish the following actions:

Wait Function	Behavior
Wait Until Next ms Multiple 	Monitors a millisecond counter and waits until the millisecond counter reaches a multiple of the amount you specify. This function is synced to the system clock.
Wait (ms) 	Waits until the millisecond counter counts to an amount equal to the input you specify
Time Delay Express VI Time Delay Delay Time (s) error in (no error) error out	Similar to the Wait (ms) function with the addition of built-in error clusters.

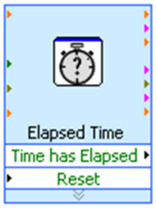
Wait Function Timing

The following illustration compares the differences between two common timing functions: Wait Until Next ms Multiple and Wait (ms). This timing diagram assumes that the wait functions begin running immediately for each loop iteration and that the loop is ready to iterate as soon as the wait function finishes.



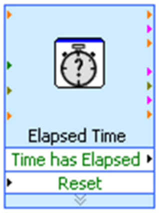
Elapsed Time Express VI

Determines how much time elapses after some point in your VI and keeps track of time while the VI continues to execute.



Demonstration: Wait Chart VI

Compare and contrast using a Wait function and the Elapsed Time Express VI for software timing.

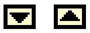


E. Data Feedback in Loops

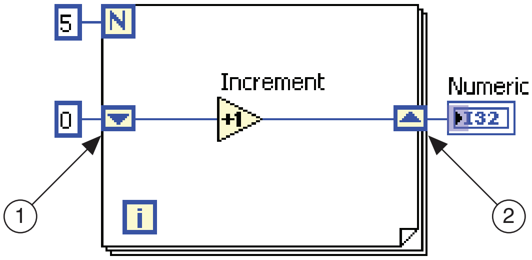
Objective: Apply shift registers when appropriate and predict the correct value at different iterations of the loop.

Introduction to Shift Registers

Shift registers store data values from previous iterations of a loop in LabVIEW.



Shift registers are similar to static variables in text-based programming languages.



1	Stores data at beginning of iteration	2	Stores data on completion of iteration

Multimedia: Using Shift Registers

When you create a shift register, the shift register reflects the data type you wire to the terminals. If you do not initialize the register, the loop uses the value written to the register when the loop last executed or it uses the default value for the data type if the loop has never executed. You can add more than one shift register to a loop for applications such as averaging data points.

Complete the multimedia module, *Using Shift Registers*, available in the `<Exercises>\LabVIEW Core 1\Multimedia\` folder.

Demonstration: Creating Shift Registers

Replace tunnels with shift registers when you need to transfer values from one loop iteration to the next. If you convert a tunnel with auto-indexing enabled to a shift register on a While Loop, the wire to any node outside the loop breaks because shift registers cannot auto-index

Exercise 4-2 Calculating Average Temperature

Goal

Use a While Loop and shift registers to average data.

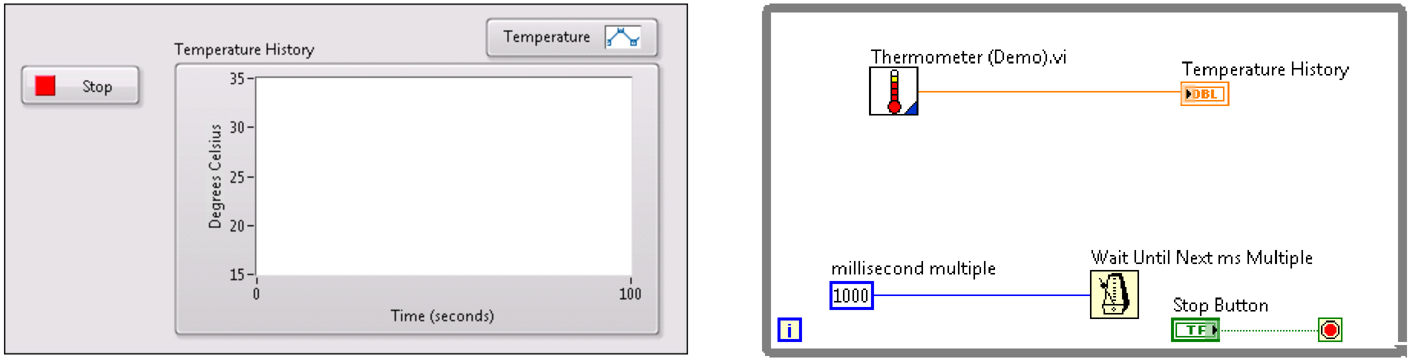
Scenario

The Temperature Monitor VI acquires and displays temperature. Modify the VI to average the last five temperature measurements and display the running average on the waveform chart.

Design

Figure 4-8 shows the Temperature Monitor VI front panel and block diagram.

Figure 4-8. Temperature Monitor VI Front Panel and Block Diagram



To modify this VI, you need to retain the temperature values from the previous four iterations of the While Loop and average the values. To accomplish this, you modify this VI as follows:

- Use a shift register with additional elements to retain data from the previous four iterations.
- Initialize the shift register with a reading from the simulated thermometer.
- Calculate and chart only the average temperature.

Implementation

1. Test the VI.

   □ Open Temperature Monitor.lvproj in the <Exercises>\LabVIEW Core 1\Temperature Monitor directory.

   □ Open **Temperature Monitor VI** from the **Project Explorer** window.

   □ Run the VI. Notice the variation in the simulated temperature reading.

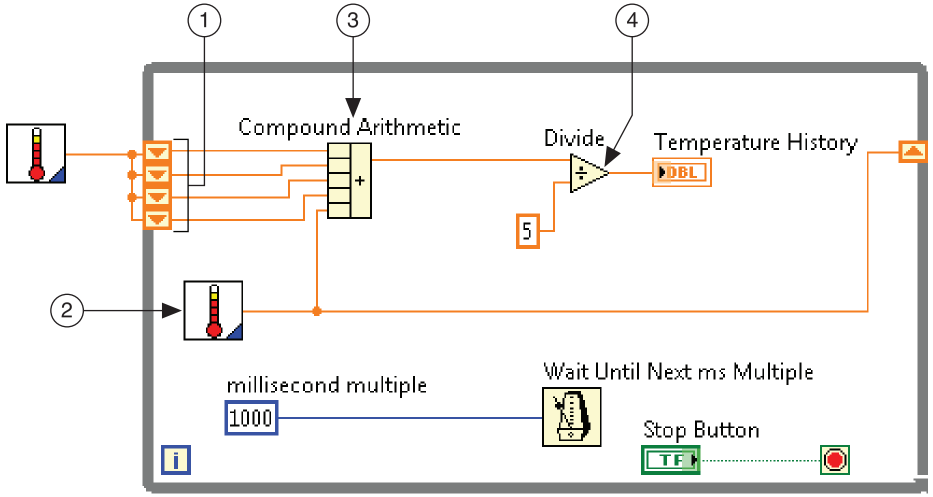
2. Stop the VI by clicking the **Stop** button on the front panel.

3. Modify the VI to reduce the number of temperature spikes.

   □ Display the block diagram.

   □ Modify the block diagram as shown in Figure 4-9.

Figure 4-9. Temperature Monitor VI Block Diagram—Average Temperature



1 **Shift Registers**—Stacked shift registers collect multiple temperature readings. Right-click the border of the While Loop and select **Add Shift Register**. Drag the lower resizing handle of the shift register to display four shift registers.

2 Create a copy of the Thermometer (Demo) VI—Press <Ctrl> while dragging the subVI outside the While Loop to create a copy. The Thermometer (Demo) VI returns one temperature measurement and initializes the left shift registers before the loop starts.

3 **Compound Arithmetic**—Returns the sum of the current temperature and the four previous temperature readings. Resize the function to have five terminals.

4 **Divide**—Returns the average of the last five temperature readings.

Note You can create stacked shift register terminals on the left side of a loop to remember multiple previous iterations and carry those values to the next iterations. This technique is useful for averaging data points. Stacked shift registers can occur only on the left side of the loop because the right terminal transfers the data generated from only the current iteration to the next iteration.

4. Save the VI.

Test

1. Run the VI.

   During each iteration of the While Loop, the Thermometer (Demo) VI takes one temperature measurement. The VI adds this value to the last four measurements stored in the left terminals of the shift register. The VI divides the result by five to find the average of the five measurements—the current measurement plus the previous four. The VI displays the average on the waveform chart. Notice that the VI initializes the shift register with a temperature measurement.

2. Stop the VI by clicking the **Stop** button on the front panel.

3. Save and close the VI and the project.

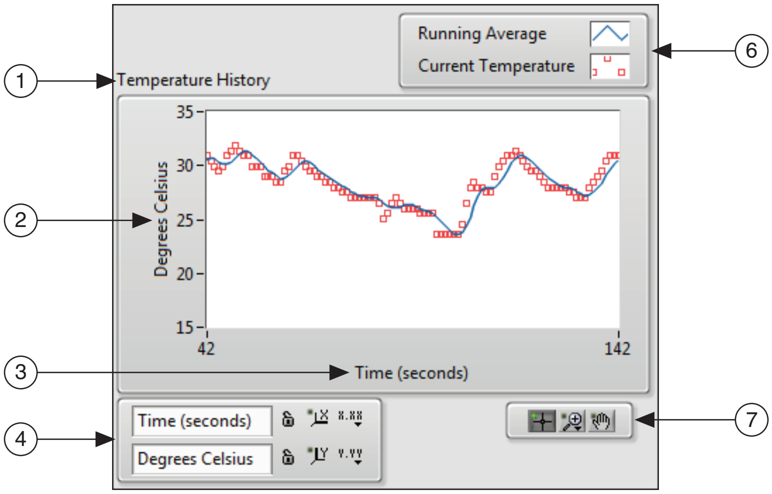
End of Exercise 4-2

F. Plotting Data

Objective: Use data feedback in a loop to plot waveform charts.

Waveform Chart

The waveform chart is a special type of numeric indicator that displays one or more plots of data typically acquired at a constant rate. Waveform charts can display single or multiple plots.



1	Label	3	X-scale	5	Graph Palette
2	Y-scale	4	Scale Legend	6	Plot Legend

Waveform Chart Properties

Waveform charts include options that you can use to customize appearance, convey more information, or highlight data.

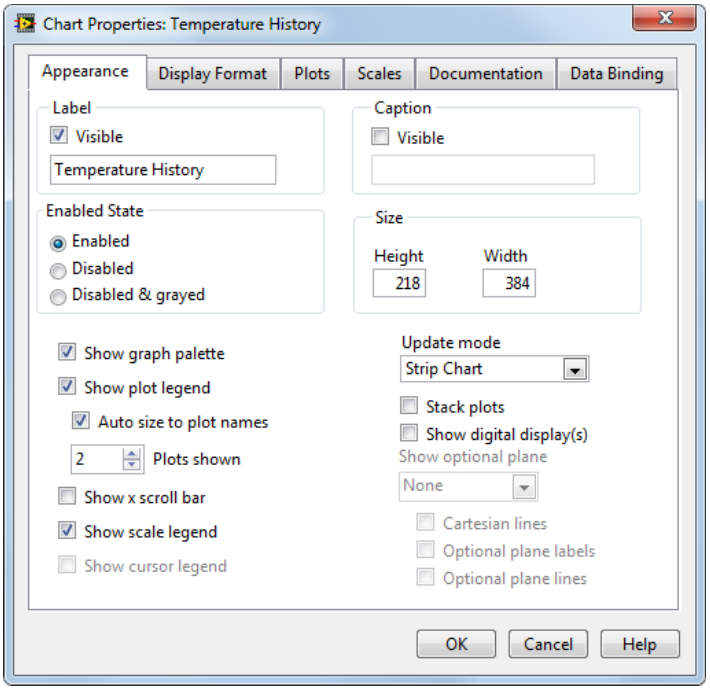


Chart Properties: Temperature History

Appearance | Display Format | Plots | Scales | Documentation | Data Binding

Label
☑ Visible
Temperature History

Caption
☐ Visible

Enabled State
● Enabled
○ Disabled
○ Disabled & grayed

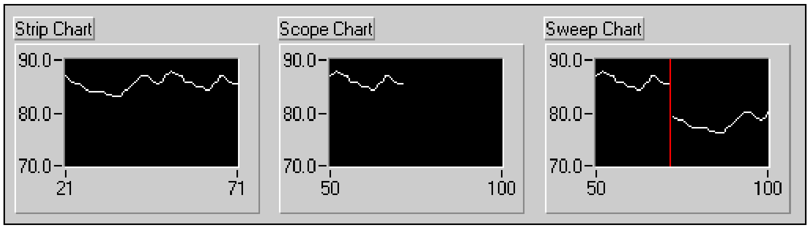
Size
Height 218
Width 384

☑ Show graph palette
☑ Show plot legend
☑ Auto size to plot names
2 Plots shown
☐ Show x scroll bar
☑ Show scale legend
☐ Show cursor legend

Update mode
Strip Chart
☐ Stack plots
☐ Show digital display(s)
Show optional plane
None
☐ Cartesian lines
☐ Optional plane labels
☐ Optional plane lines

You can configure how the chart updates to display new data. The chart uses the following modes to display data:

- **Strip Chart**—Shows running data continuously scrolling from left to right across the chart with old data on the left and new data on the right.
- **Scope Chart**—Shows one item of data, such as a pulse or wave, scrolling partway across the chart from left to right.
- **Sweep Chart**—Works similarly to a scope chart except it shows the old data on the right and the new data on the left separated by a vertical line.



Exercise 4-3 Temperature Monitor VI—Plot Multiple Temperatures

Goal

Plot multiple data sets on a single waveform chart and customize the chart view.

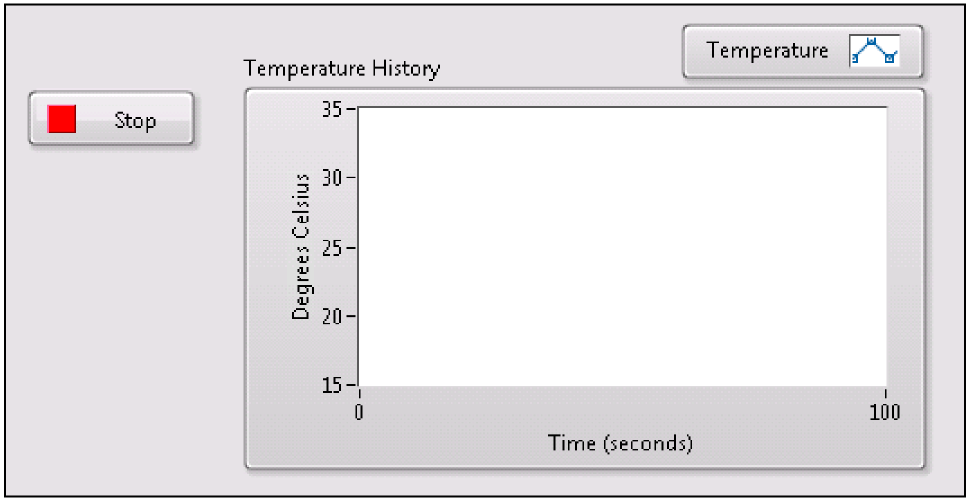
Scenario

Modify the VI from Exercise 4-2 to plot both the current temperature and the running average on the same chart. In addition, allow the user to examine a portion of the plot while the data is being acquired.

Design

Figure 4-10 shows the front panel for the existing Temperature Monitor VI and Figure 4-11 shows the block diagram.

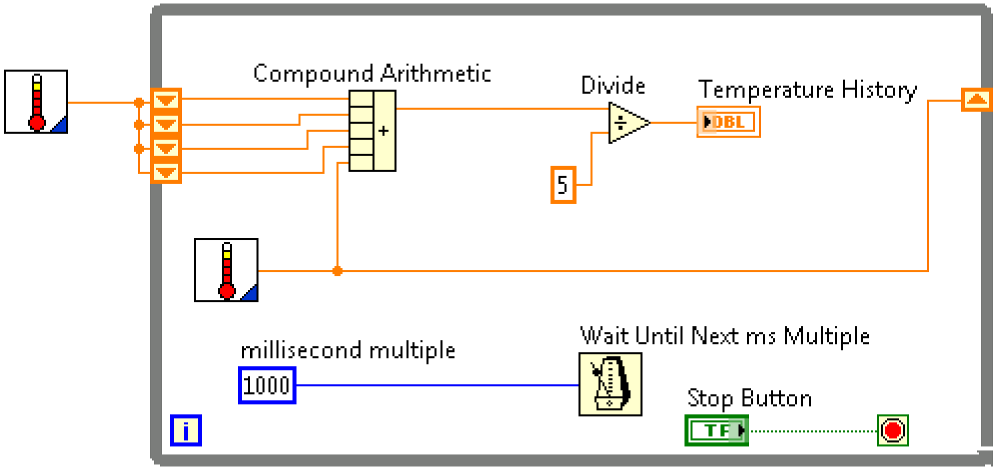
Figure 4-10. Temperature Monitor VI Front Panel



To allow the user to examine a portion of the plot while the data is being acquired, display the scale legend and the graph palette for the waveform chart. Also, expand the legend to show additional plots.

To modify the block diagram in Figure 4-11, you must modify the chart terminal to accept multiple pieces of data. Use a Bundle function to combine the average temperature and the current temperature into a cluster to pass to the **Temperature History** terminal.

Figure 4-11. Original Temperature Monitor VI Block Diagram



Implementation

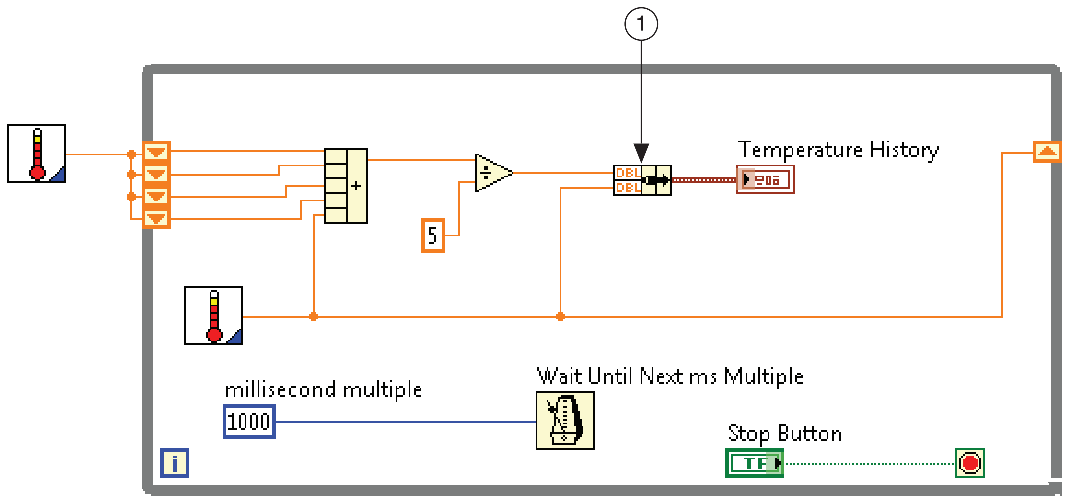
1. Open the Temperature Monitor VI you created in Exercise 4-2.

   □ Open `Temperature Monitor.lvproj` in the `<Exercises>\LabVIEW Core 1\Temperature Monitor` directory.

   □ Open **Temperature Monitor.vi** from the **Project Explorer** window.

2. Modify the block diagram so that it resembles Figure 4-12.

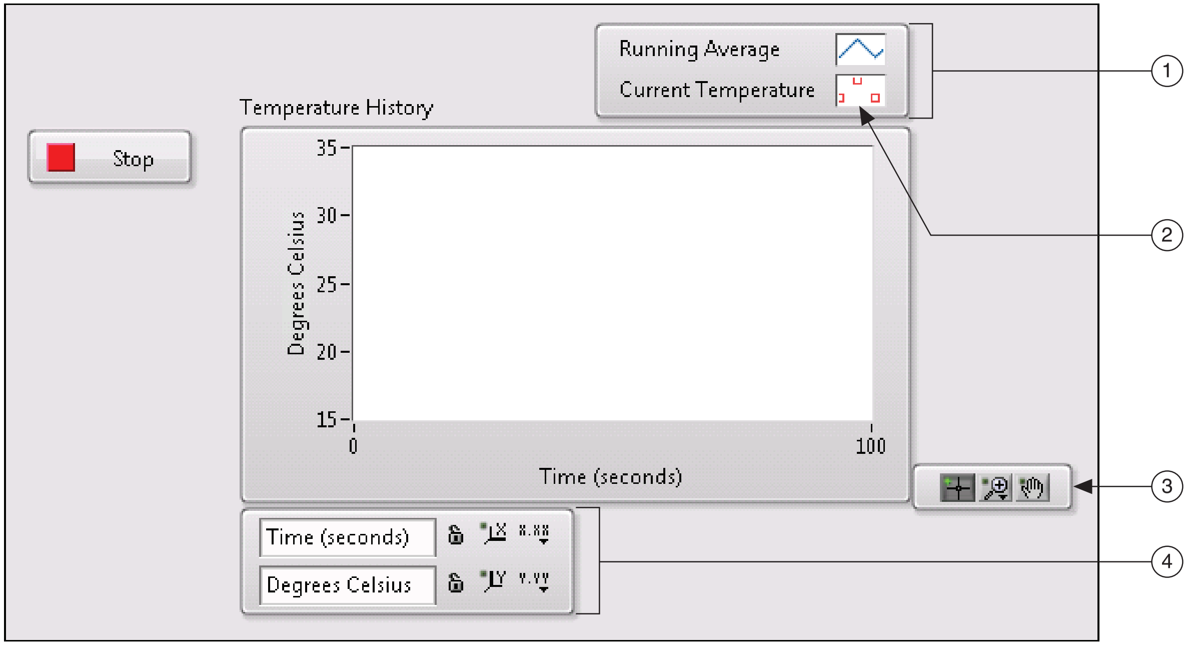
Figure 4-12. Temperature Monitor VI Block Diagram—Plotting Multiple Temperatures



1 **Bundle**—Passes the current temperature and average temperature to the Temperature History chart.

3. Modify the front panel so that it resembles Figure 4-13.

Figure 4-13. Temperature Monitor VI Front Panel—Plotting Multiple Temperatures



1 **Show both plots in the plot legend**—Use the positioning tool to resize the plot legend to show two objects. Double-click the label to edit the plot names. The order of the plots listed in the plot legend is the same as the order of the items wired to the Bundle function on the block diagram.

2 **Change the plot type of Current Temperature**—Use the Operating tool to select the plot in the plot legend. Click the plot icon, select **Common Plots** from the menu, and choose the plot you want.

3 **Display Graph Palette**—Right-click the **Temperature History** chart and select **Visible Items»Graph Palette**.

4 **Display Scale Legend**—Right-click the **Temperature History** chart and select **Visible items»Scale Legend**.

4. Save the VI.

Test

1. Run the VI. Use the tools in the scale legend and the graph palette to examine the data as it generates.

2. Close the VI and project when you are finished.

End of Exercise 4-3

Additional Resources

Learn More About	LabVIEW Help Topic
Using shift registers	*Shift Registers: Passing Values between Loop Iterations* *Passing Multiple Values to the Next Loop Iteration*

Activity 4-2: Lesson Review

1. Which structure must run at least one time?
   a. While Loop
   b. For Loop

Activity 4-2: Lesson Review - Answers

1. Which structure must run at least one time?

   a. While Loop

   b. For Loop

5 Creating and Leveraging Data Structures

In this lesson you will learn about arrays, clusters, and type definitions and be able to identify applications where using these data structures can be beneficial.

Topics

+ Arrays
+ Common Array Functions
+ Polymorphism
+ Auto-Indexing
+ Clusters
+ Type Definitions

Exercises

Exercise 5-1	Manipulating Arrays
Exercise 5-2	Temperature Warnings VI—Clusters
Exercise 5-3	Temperature Warnings VI—Type Definition

A. Arrays

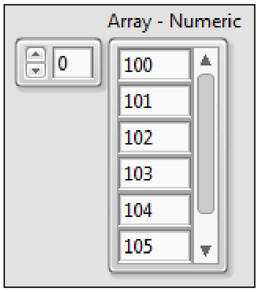
Objective: Identify when to use arrays and learn how to create and initialize arrays.

Arrays

Array Collection of data elements that are of the same type.

Elements The data that make up the array. Elements can be numeric, Boolean, path, string, waveform, and cluster data types.

Dimension Length, height, or depth of the array. Arrays can have one or more dimensions and as many as $(2^{31})-1$ dimensions.

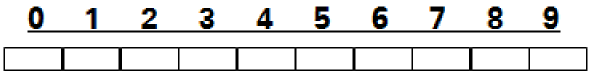


Note Array indexes in LabVIEW are zero-based. The index of the first element in the array, regardless of its dimension, is zero.

1D and 2D Examples

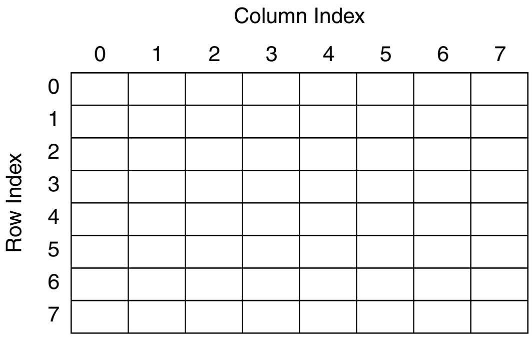
Arrays can have multiple dimensions.

- 1D array:



For example, LabVIEW represents a text array that lists the twelve months of the year as a 1D array of strings with twelve elements. Index is zero-based, which means the range is 0 to *n* - 1, where *n* is the number of elements in the array. For example, *n* = 12 for the twelve months of the year, so the index ranges from 0 to 11. March is the third month, so it has an index of 2.

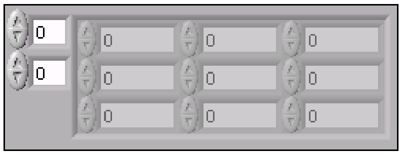
- 2D array:



For example, LabVIEW represents a table of data with rows and columns as a 2D array.

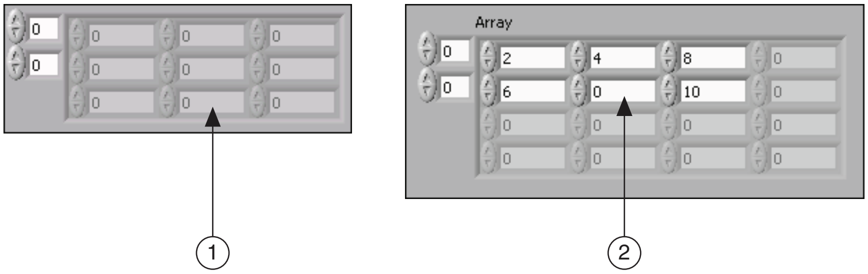
2D Arrays

A 2D array stores elements in a grid. It requires a column index and a row index to locate an element, both of which are zero-based.



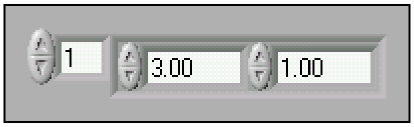
Initializing Arrays

An uninitialized array contains a fixed number of dimensions but no elements. An initialized defines the number of elements in each dimension and the contents of each element.

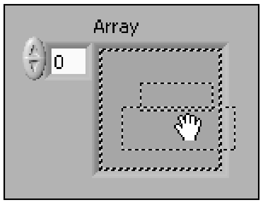


1	Uninitialized array	2	Initialized array

Demonstration: Viewing Arrays



Create an array control or indicator on the front panel by adding an array shell to the front panel, as shown in the following front panel, and dragging a data object or element, which can be a numeric, Boolean, string, path, refnum, or cluster control or indicator, into the array shell.



To create an array constant on the block diagram, select an array constant on the **Functions** palette, place the array shell on the block diagram, and place a string constant, numeric constant, a Boolean constant, or cluster constant in the array shell.

Restrictions

You cannot create arrays of arrays. However, you can use a multidimensional array or create an array of clusters where each cluster contains one or more arrays. Also, you cannot create an array of subpanel controls, tab controls, .NET controls, ActiveX controls, charts, or multi-plot XY graphs.

B. Common Array Functions

Objective: Create and manipulate arrays using built-in array functions.

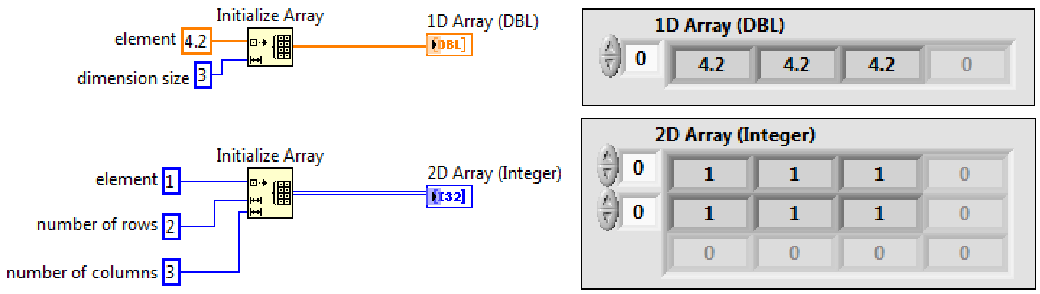
Multimedia: Common Array Functions

Functions you can use to manipulate arrays are located on the **Array** palette.

Complete the multimedia module, *Common Array Functions*, available in the `<Exercises>\LabVIEW Core 1\Multimedia\` folder.

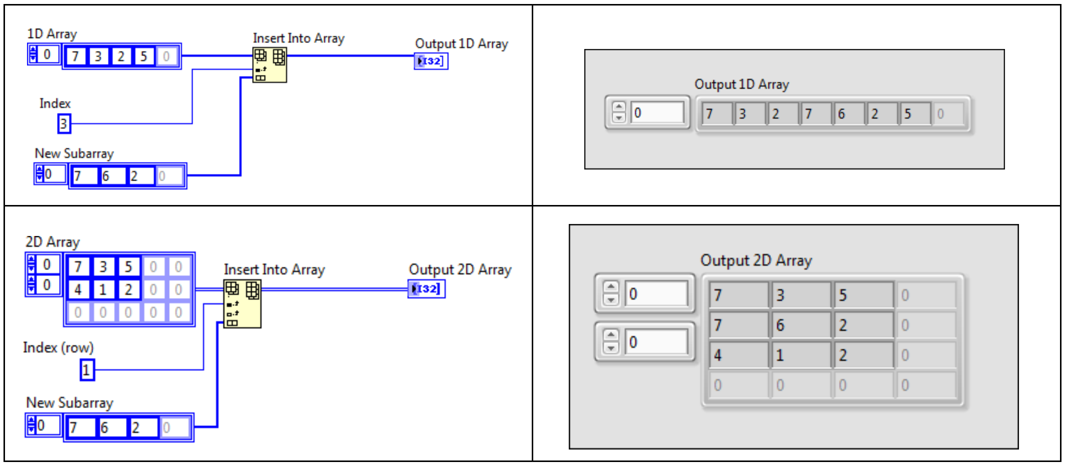
Initialize Array

Creates an n-dimensional array in which every element is initialized to the value of **element**.



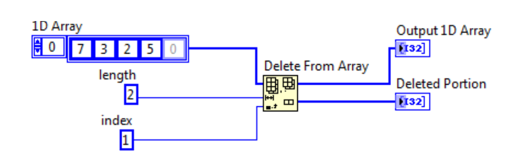
Insert Into Array

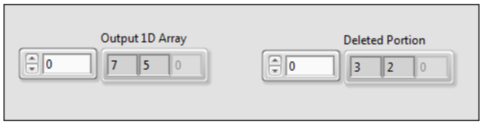
Inserts an element or subarray at the point you specify in index.

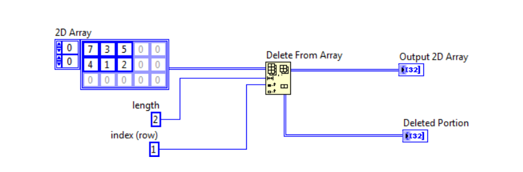


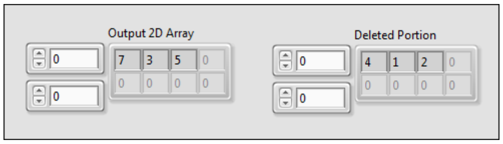
Delete From Array

Deletes an element or subarray from **n-dim array** of **length** elements starting at **index**. Returns the edited array in **array w/ subset deleted** and the deleted element or subarray in **deleted portion**.



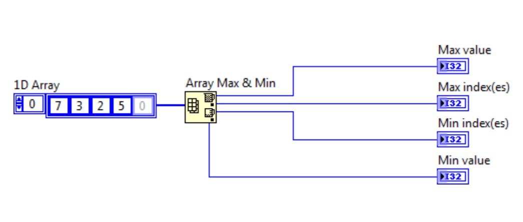

Output 1D Array
0 7 5 0
Deleted Portion
0 3 2 0

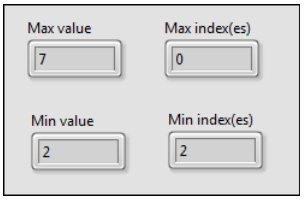


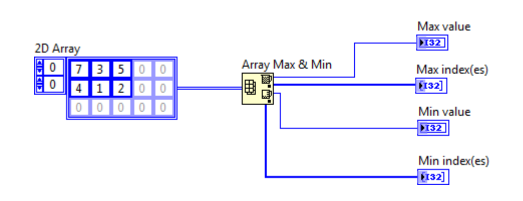

Output 2D Array
0 0
7 3 5 0
0 0 0 0
Deleted Portion
0 0
4 1 2 0
0 0 0 0

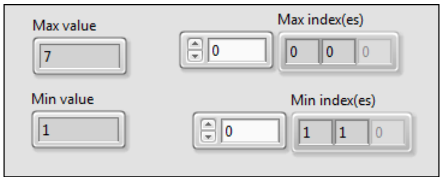
Array Max & Min

Returns the maximum and minimum values in array, along with the indexes for each value.



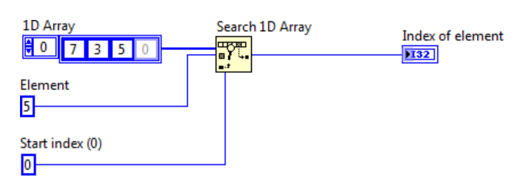

Max value
7
Max index(es)
0
Min value
2
Min index(es)
2

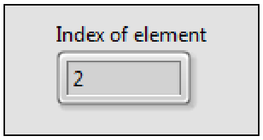


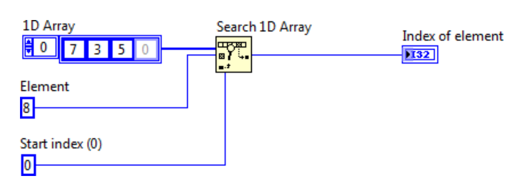

Max value
7
Max index(es)
0 0 0 0
Min value
1
Min index(es)
0 1 1 0

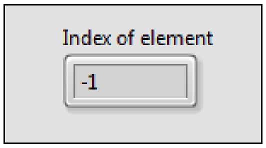
Search 1D Array

Searches for an element in a 1D array starting at start index. Because the search is linear, you need not sort the array before calling this function. LabVIEW stops searching as soon as the element is found.




Index of element
2



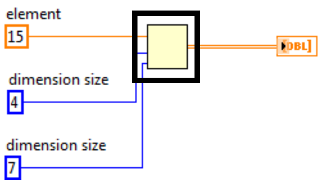
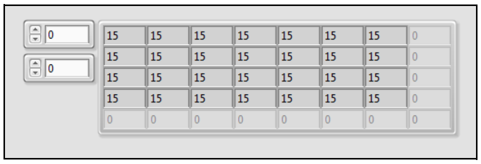
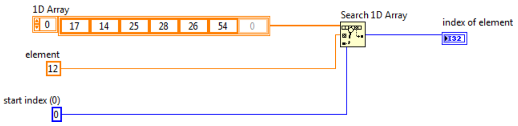
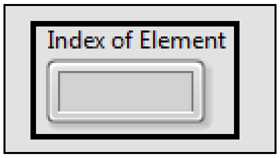
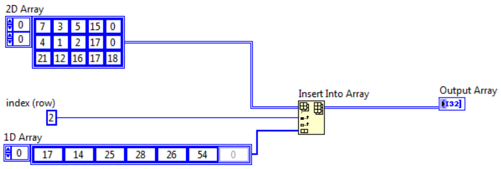
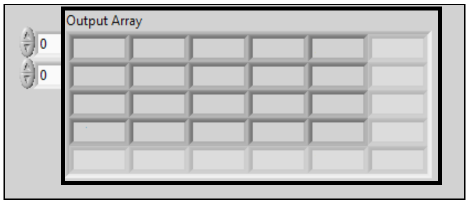
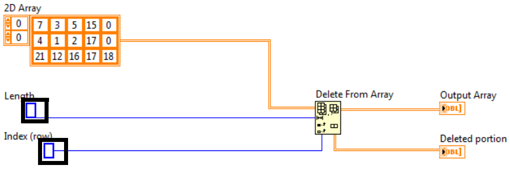
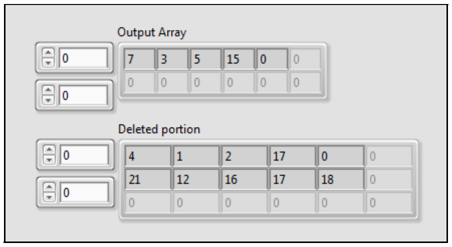

Index of element
-1

Activity 5-1: Using Array Functions

Goal: Complete the highlighted portion in each VI

Description

Each of the VIs shown has missing information. Determine what belongs in the highlighted section.

Block Diagram	Result
 element 15 dimension size 4 dimension size 7 DBL	 0 0 15 15 15 15 15 15 15 0 15 15 15 15 15 15 15 0 15 15 15 15 15 15 15 0 15 15 15 15 15 15 15 0 0 0 0 0 0 0 0 0
 1D Array 0 17 14 25 28 26 54 0 element 12 start index (0) 0 Search 1D Array index of element I32	
 2D Array 0 0 7 3 5 15 0 4 1 2 17 0 21 12 16 17 18 index (row) 2 1D Array 0 17 14 25 28 26 54 0 Insert Into Array Output Array I32	
 2D Array 0 0 7 3 5 15 0 4 1 2 17 0 21 12 16 17 18 Length Index (row) Delete From Array Output Array DBL Deleted portion DBL	 Output Array 0 0 7 3 5 15 0 0 0 0 0 0 0 0 Deleted portion 0 0 4 1 2 17 0 0 21 12 16 17 18 0 0 0 0 0 0 0

C. Polymorphism

Objective: Understand the ability of various VIs to accept input data of different data types.

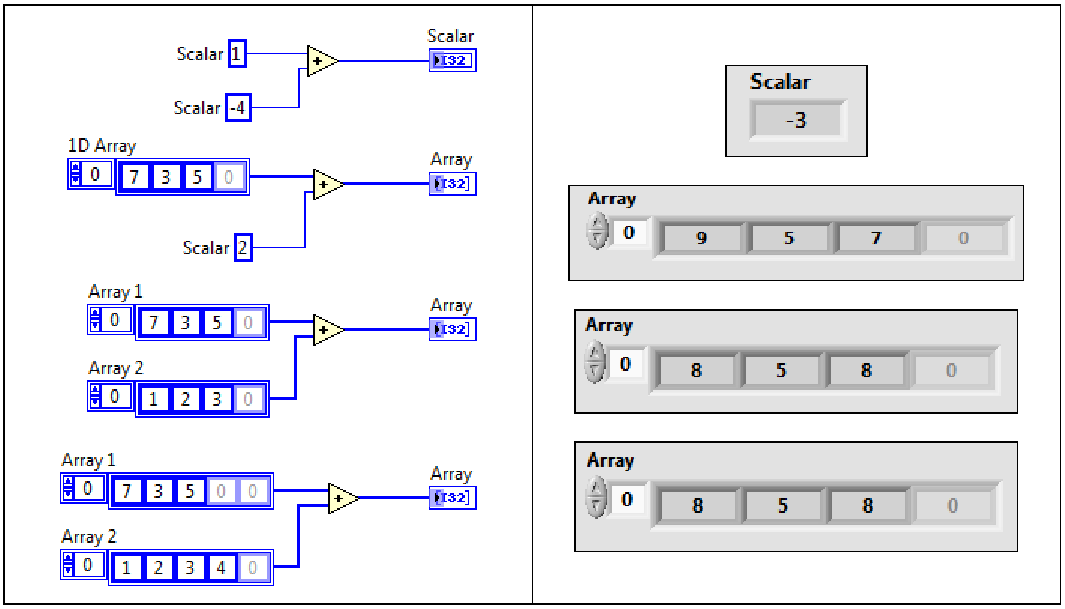
Polymorphism

Polymorphism The ability of VIs and functions to automatically adapt to accept input data of different data types.

Functions are polymorphic to varying degrees—none, some, or all of their inputs can be polymorphic.

Arithmetic Functions Are Polymorphic

LabVIEW arithmetic functions are polymorphic.



D. Auto-Indexing

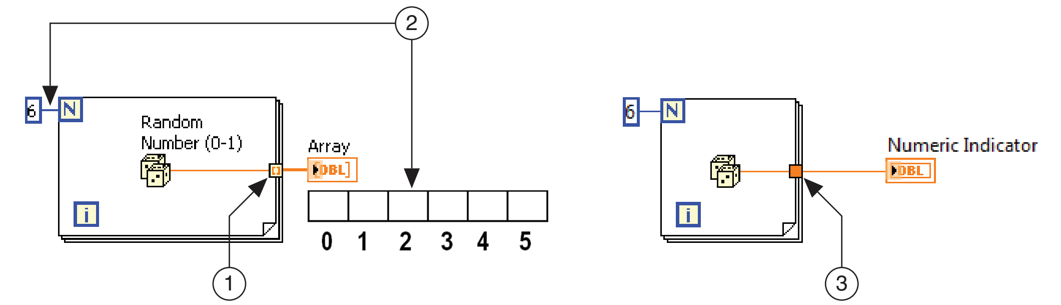
Objective: Use auto-indexed inputs and outputs to create graphs and arrays.

Auto-indexing The ability to automatically process every element in an array.

Auto-Indexing

If you wire an array to or from a For Loop or While Loop, you can link each iteration of the loop to an element in that array by enabling auto-indexing. The tunnel image changes from a solid square to the image to indicate auto-indexing.

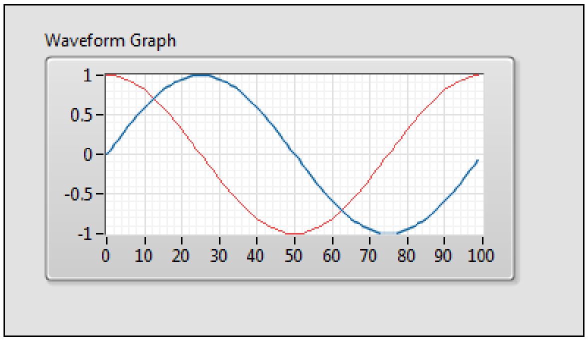
[]

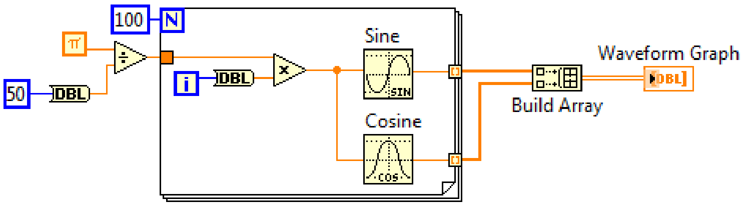


1 Right-click the tunnel and select **Enable Indexing** or **Disable Indexing** to toggle the state of the tunnel.
2 Auto-indexed output arrays are always equal in size to the number of iterations.
3 Only one value (the last iteration) is passed out of the loop when auto-indexing is disabled.

Waveform Graphs

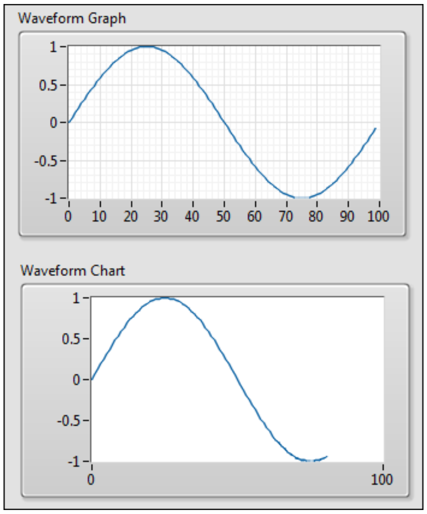
A waveform graph collects the data in an array and then plots the data to the graph.

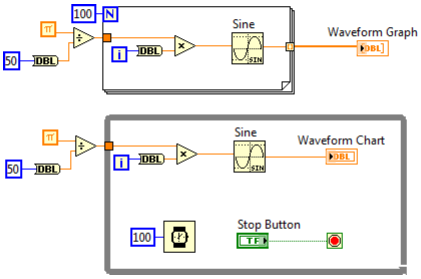




Charts vs. Graphs—Single-Plot

Charts are generally used inside the While Loop and graphs are generally outside the While Loop.

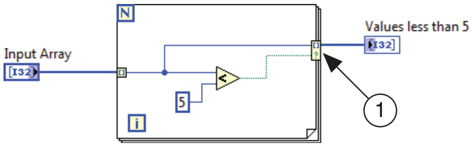




Auto-Indexing with a Conditional Tunnel

You can determine what values LabVIEW writes to the loop output tunnel based on a condition you specify.

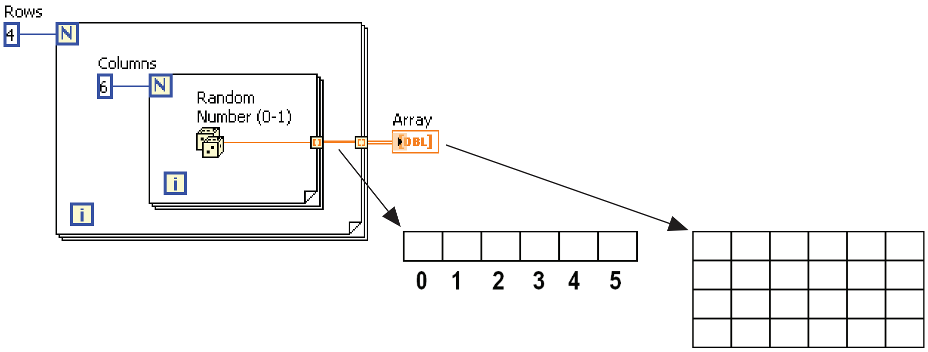
For example, consider the following block diagram. The array **Input Array** contains the following elements: 7, 2, 0, 3, 1, 9, 5, and 7. Because of the conditional tunnel, the **Values less than 5** array contains only the elements 2, 0, 3, and 1 after this loop completes all iterations.



1 Right-click the loop output tunnel and selecting **Tunnel Mode»Conditional** from the shortcut menu.

Creating Two-Dimensional Arrays

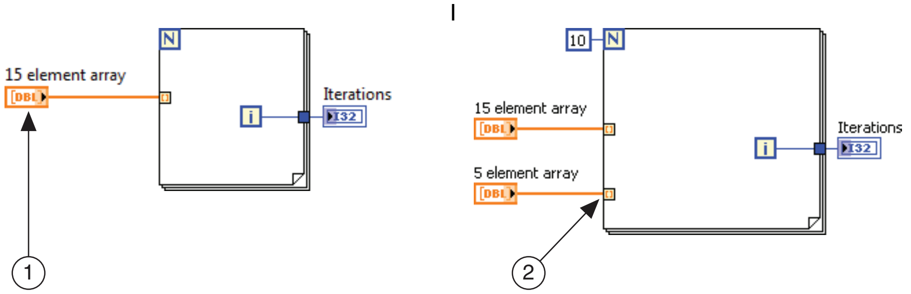
Use two For Loops, nested one inside the other, to create a 2D array.



1 The inner loop creates the column elements and the outer loop creates the row elements.

Auto-Indexing Input

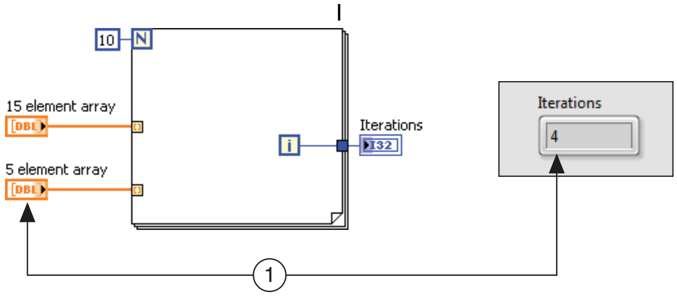
Use an auto-indexing input array to perform calculations on each element in an array. If you wire an array to an auto-indexing tunnel on a For Loop, you do not need to wire the count (N) terminal.



1 The For Loop executes the number of times equal to the number of elements in the array.
2 If the iteration count terminal is wired and arrays of different sizes are wired to auto-indexed tunnels, the actual number of iterations becomes the smallest of the choices.

Auto-Indexing Input—Different Array Sizes

If the iteration count terminal is wired and arrays of different sizes are wired to auto-indexed tunnels, the actual number of iterations becomes the smallest of the choices.



1 The For Loop iterates 5 times and because the iterations are zero-based, the output is 4.

Exercise 5-1 Manipulating Arrays

Goal

Manipulate arrays using various LabVIEW functions.

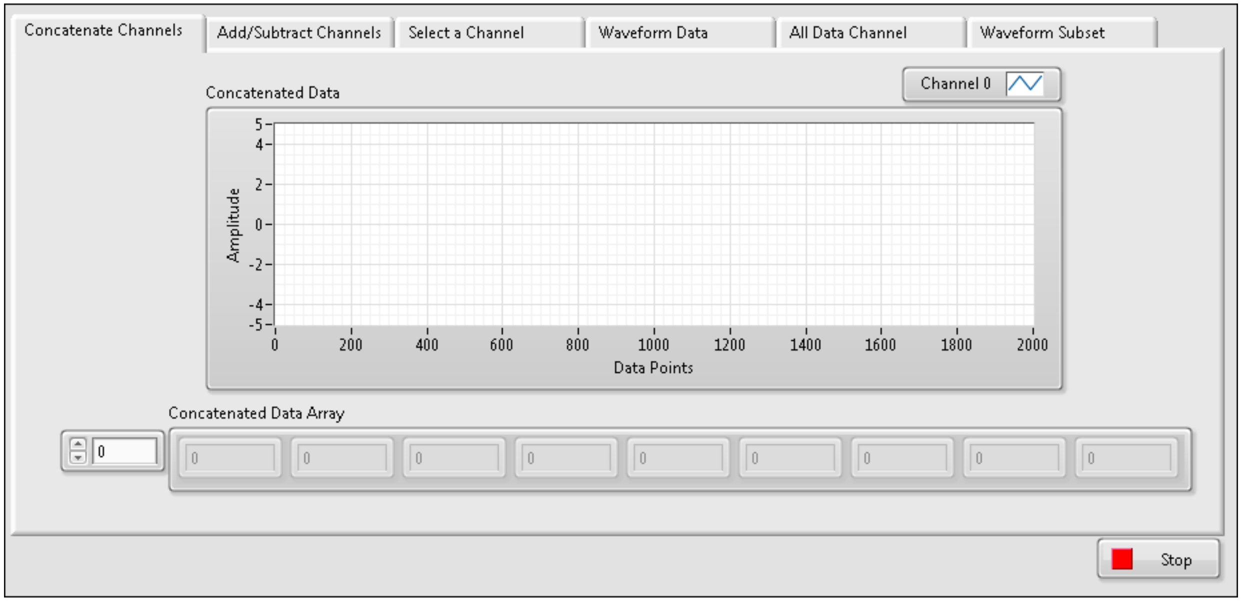
Description

You are given a VI and asked to enhance it for a variety of purposes. The front panel of this VI is built. You complete the block diagram to practice several different techniques to manipulate arrays.

Implementation

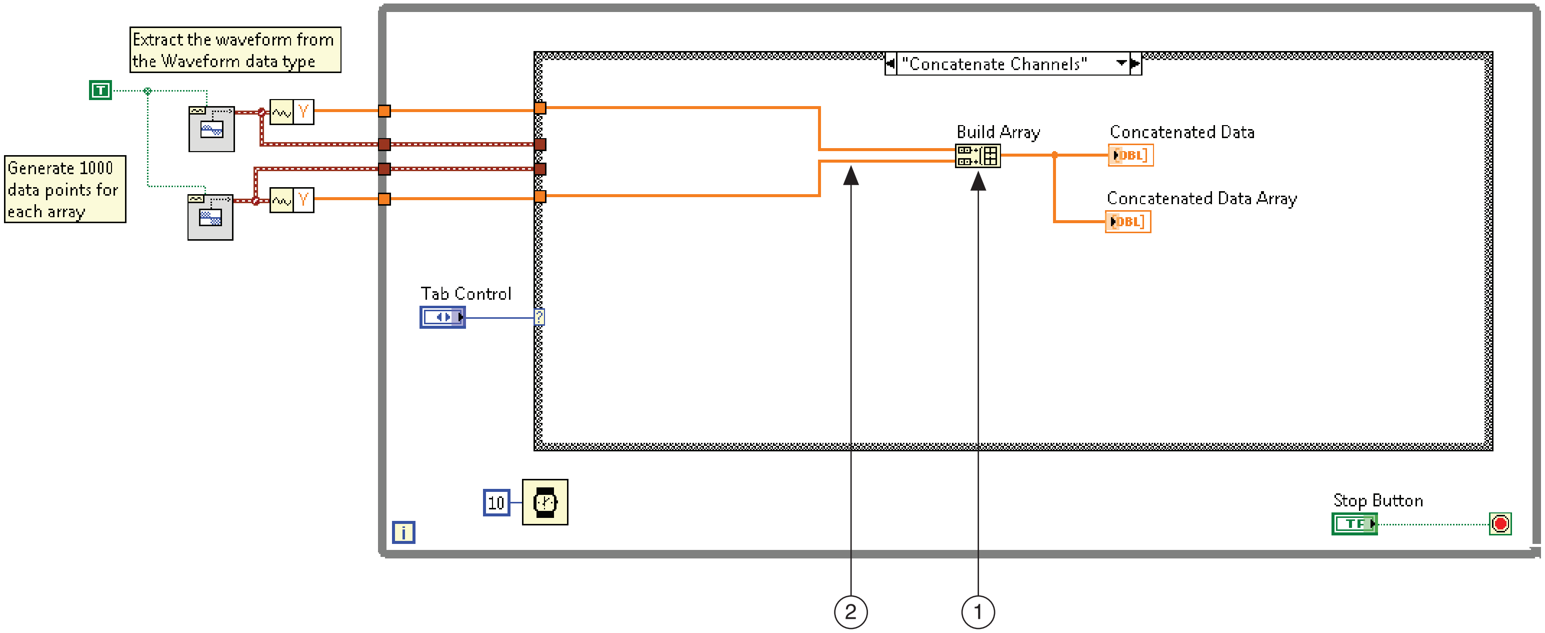
1. Open `Manipulating Arrays.lvproj` in the `<Exercises>\LabVIEW Core 1\Manipulating Arrays` directory.
2. Open **Array Manipulation** VI from the **Project Explorer** window. The front panel, shown in Figure 5-1, is already built for you.

Figure 5-1. Array Manipulation VI Front Panel



3. Open the block diagram and complete each of the cases that correspond to the tabs on the front panel as shown in Figures 5-2 through 5-8.

Figure 5-2. Array Manipulation VI—Concatenate Channels Case



1 **Build Array**—Expand this node to accept two inputs, and then right-click and select **Concatenate inputs** from the shortcut menu.
2 Wire the sine wave and square wave outputs to the Build Array function to create a 1D array with both waveforms.

4. Switch to the front panel and test the Concatenate Channels case.

   □ On the front panel, click the **Concatenate Channels** tab.

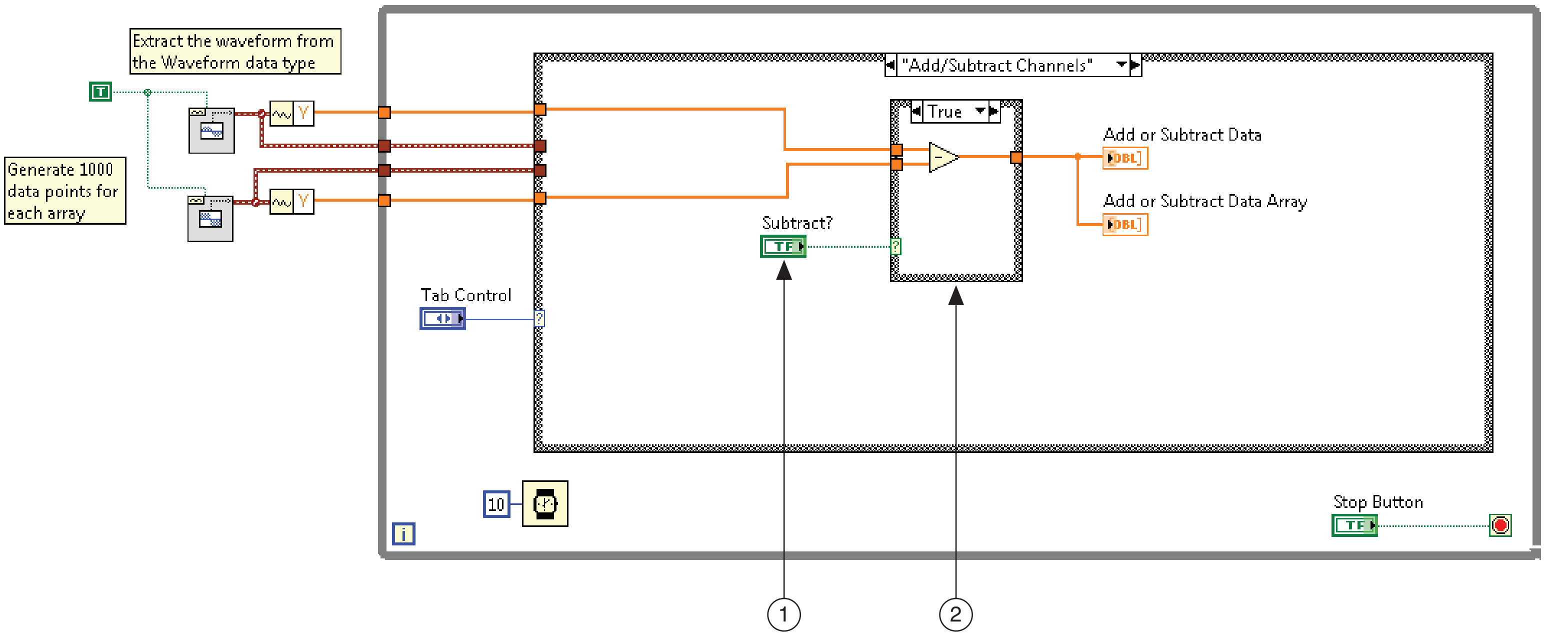
   □ Run the VI and notice that the sine wave is concatenated with a square wave.

5. Stop the VI.

6. Switch to the block diagram and select the Add/Subtract Channels case.

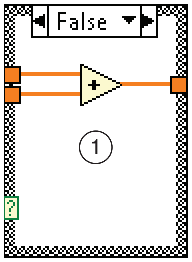
7. Complete the Add/Subtract Channels case as shown in Figure 5-3 and Figure 5-4.

Figure 5-3. Array Manipulation VI—Add/Subtract Channels True Case


Extract the waveform from the Waveform data type
Generate 1000 data points for each array
"Add/Subtract Channels"
True
Add or Subtract Data
DBL
Add or Subtract Data Array
DBL
Subtract?
TF
Tab Control
10
i
Stop Button
TF
1
2

1 **Subtract?**—Wire this to the case selector terminal so that the correct case executes when you click the Subtract? button on the front panel.
2 **Case Structure**—Place a Subtract function in the True case, so that the VI subtracts the elements of the array when the Subtract? button on the front panel is pressed.

Figure 5-4. Array Manipulation VI—Add/Subtract Channels False Case



1 When the value of the Subtract? Boolean control is False, the array elements are added.

Note This case demonstrates polymorphic functionality by adding and subtracting elements of the array.

8. Switch to the front panel and test the Add/Subtract Channels case.

   □ On the front panel, click the **Add/Subtract Channels** tab.

   □ Run the VI.

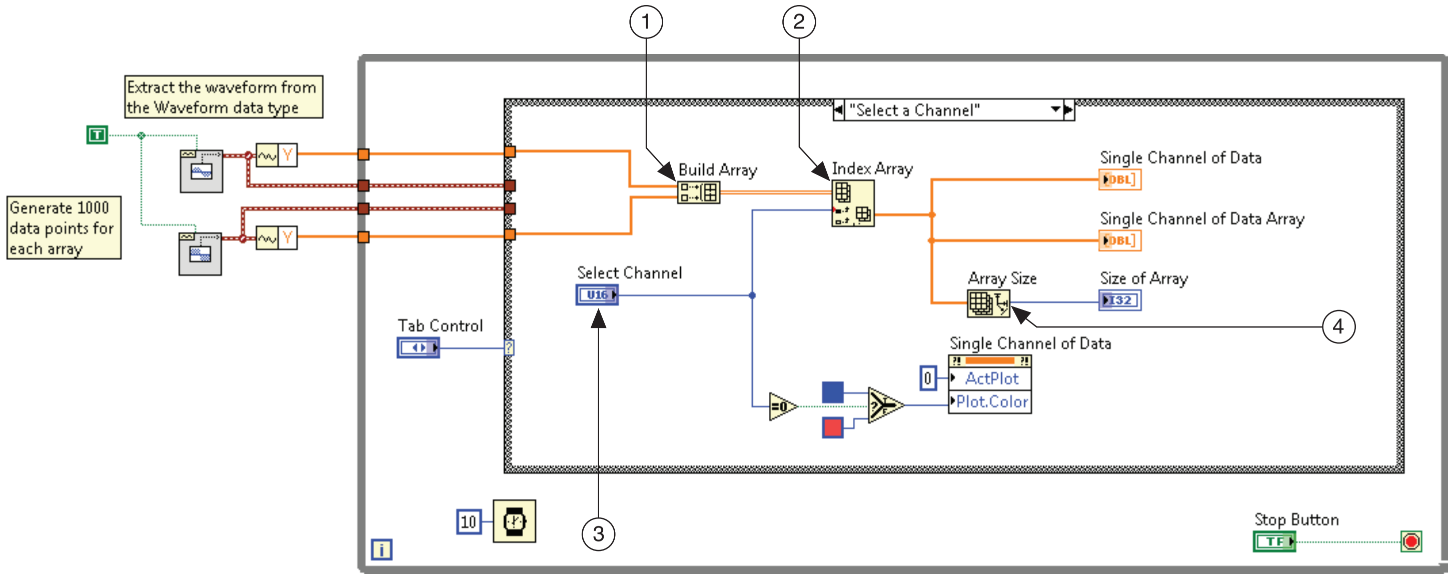
   □ Click the **Subtract?** button and observe the behavior of subtracting the square wave from the sine wave.

9. Stop the VI.

10. Switch to the block diagram and select the Select a Channel case.

11. Complete the Select a Channel case as shown in Figure 5-5.

Figure 5-5. Array Manipulation VI—Select a Channel



1 **Build Array**—Combines the sine and square waves into one 2D array.
2 **Index Array**—Extracts row 0 or 1 from the 2D array. The output from this function is a 1D array and is the waveform you select with the **Select Channel** control. The waveform is displayed on the **Single Channel of Data** Waveform Graph and the **Single Channel of Data Array** indicator.
3 **Select Channel**—Wire to the **row** input of the Index Array function.
4 **Array Size**—Because you are using a 1D array, this function outputs a scalar value.

Note The Select a Channel case uses a property node to change the color of the graph plot. You learn about Property Nodes *LabVIEW Core 2*.

12. Switch to the front panel and test the Select a Channel case.

    □ On the front panel, click the **Select a Channel** tab.

    □ Run the VI.

    □ Switch between Channel 0 and Channel 1 and notice the different values shown in the **Single Channel of Data Array** indicator.

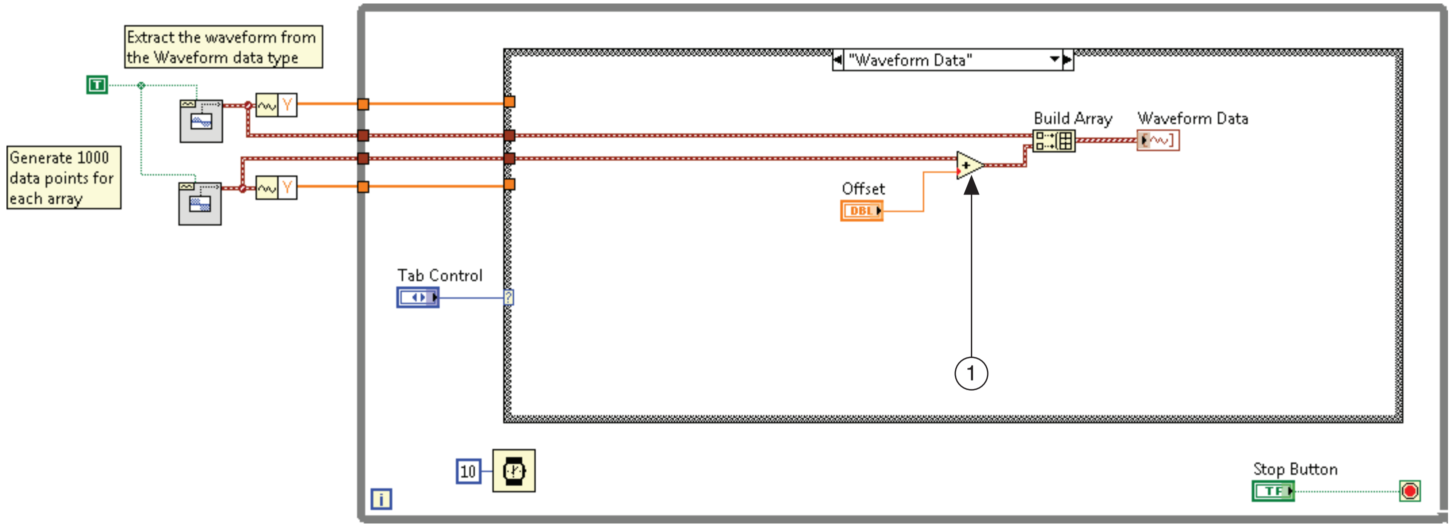
13. Stop the VI.

14. Switch to the block diagram and select the Waveform Data case.

15. Complete the Waveform Data case block diagram as shown in Figure 5-6.

    The waveform datatype is a special kind of cluster that contains additional timing information about the waveform.

Figure 5-6. Array Manipulation VI—Waveform Data


Extract the waveform from the Waveform data type
Generate 1000 data points for each array
"Waveform Data"
Build Array
Waveform Data
Offset
DBL
Tab Control
1
10
Stop Button
TF

1 **Add**—Uses the value from the **Offset** control to modify the value of the waveform in the waveform datatype. Notice the value from the **Offset** control must be coerced to be used with the waveform datatype.

Note Polymorphism is the ability of VIs and functions to automatically adapt to accept input data of different data types, including arrays, scalars, and waveforms. VIs and functions are polymoprhic to varying degrees.

16. Switch to the front panel and test the Waveform Data case.

    □ On the front panel, click the **Waveform Data** tab.

    □ Run the VI.

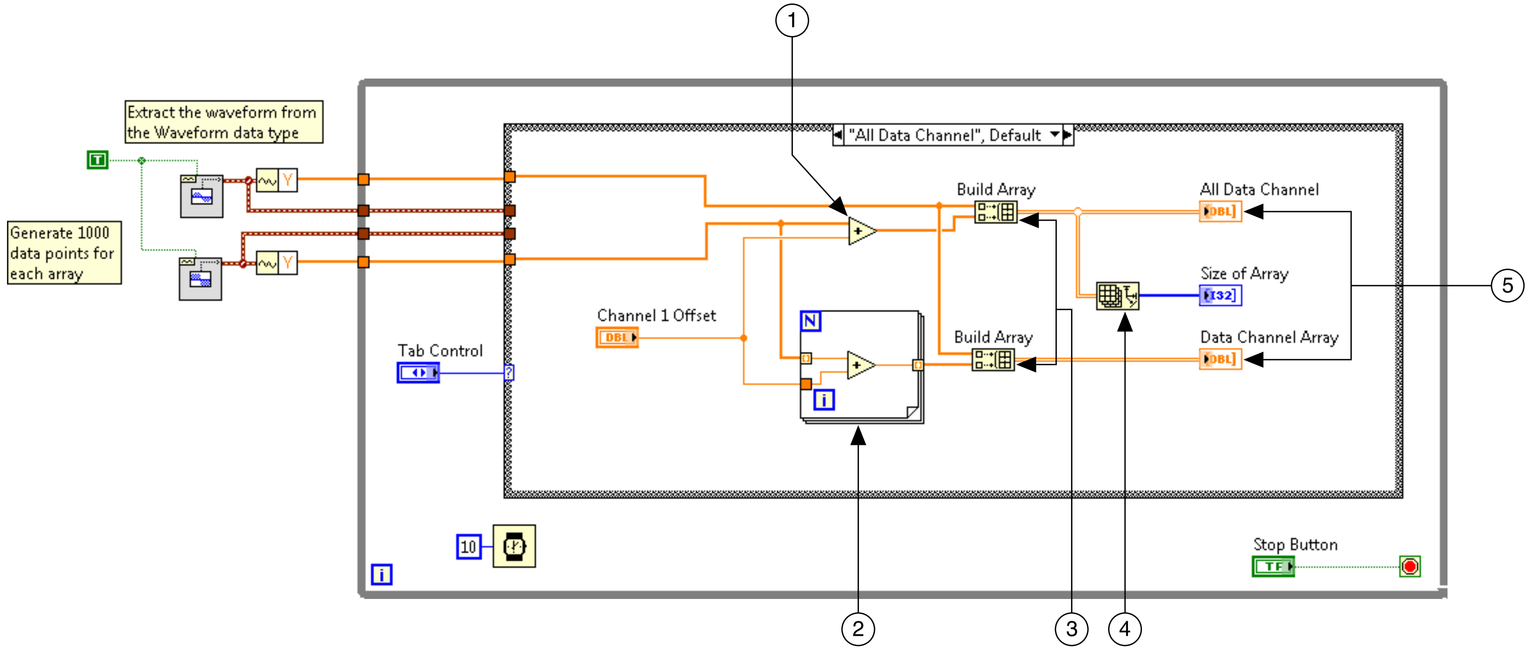
    □ Change the value of the **Offset** control and notice the square wave move on the **Waveform Data** chart.

17. Stop the VI.

18. Switch to the block diagram and select the All Data Channel case.

19. Complete the All Data Channel case as shown in Figure 5-7.

Figure 5-7. Array Manipulation VI—All Data



1 **Add**—Modify the same data in one array by adding the value of the Channel 1 Offset to each element of the array.
2 **For Loop**—Extracts each element of the array using auto indexing so that the Add function in the For Loop can add the scalar value.
3 **Build Array**—Takes the two 1D arrays and builds a 2D array. Each 1D array becomes a row in the 2D array.
4 **Array Size**—Outputs a 1D array where each element shows the size of each dimension. In this exercise, you have 2 elements of data for the number of rows and columns.
5 **All Data Channel** and **Data Channel Array** indicators display the same data.

Note The polymorphic functionality of LabVIEW functions allows you to perform the same operation on each element without extracting the array elements, as you do with the two Add functions in the All Data Channel case.

20. Switch to the front panel and test the All Data Channel case.

    □ On the front panel, click the **All Data Channel** tab.

    □ Run the VI.

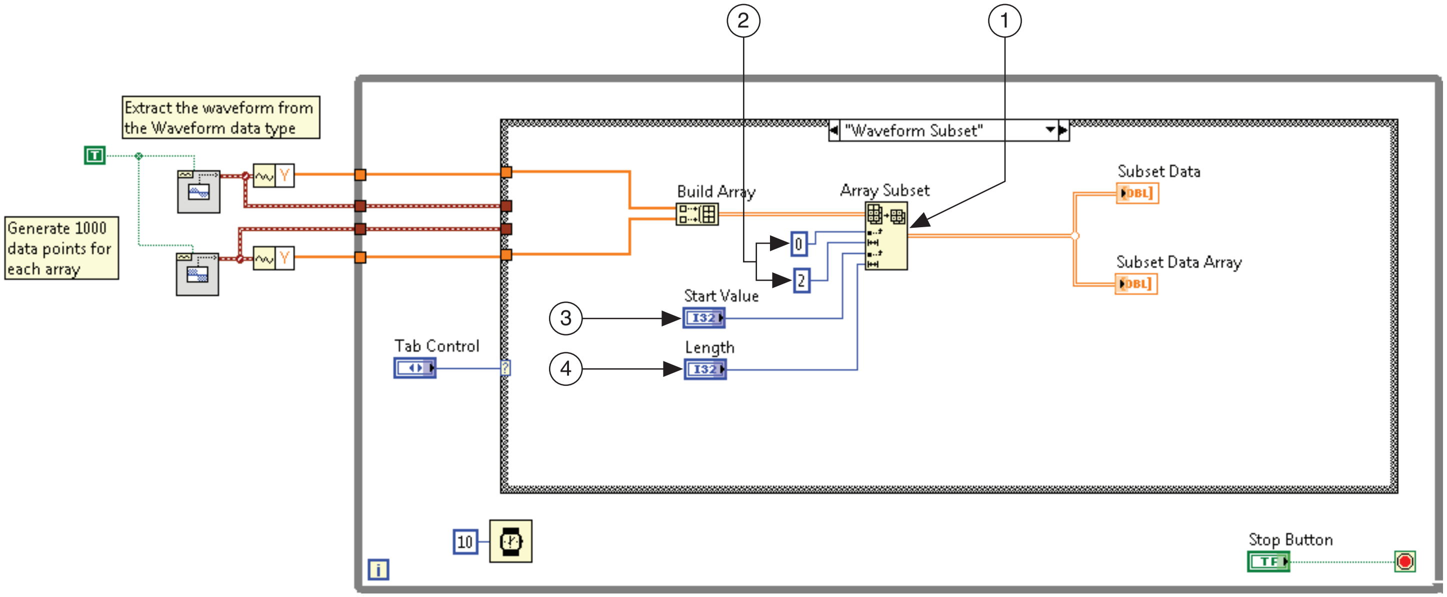
    □ Change the value of the **Channel 1 Offset** control and observe the behavior.

21. Stop the VI.

22. Switch to the block diagram and select the Waveform Subset case.

23. Complete the Waveform Subset case as shown in Figure 5-8.

Figure 5-8. Array Manipulation VI—Waveform Subset



1 **Array Subset**—Extracts a subset of an existing array. In this exercise, you use this function to zoom in on a subset of the waveform you generated.
2 **Numeric Constant**—These constants specify that the function extract the first two rows starting at element 0.
3 **Start Value**—Sets the start index. The default value is set to start at element 0.
4 **Length**—Sets the number of elements to extract. The default value is set to output 1000 elements.

24. Switch to the front panel and test the Waveform Subset case.

    - □ On the front panel, click the **Waveform Subset** tab.
    - □ Run the VI.
    - □ Change value of the **Start Value** and **Length** sliders and notice that the **Subset Data** waveform graph x-axis starts at zero and finishes at the number of elements in the new array. The x-axis starts at zero because the VI creates a brand new array and the graph does not know where the data was located in the original array.

25. Stop the VI.

Using the NI Example Finder to Learn More about Arrays

Use the NI Example Finder to browse or search examples installed on your computer or on the NI Developer Zone at `ni.com/zone`. Example VIs can show you how to use specific functions and programming concepts such as arrays and polymorphism.

Complete the following steps to use the NI Example finder to locate example VIs that demonstrate different ways to use the Array function.

1. Select **Help»Find Examples** to start the NI Example Finder.
2. Click the **Search** tab and enter the keyword `array`.
3. Click the **Search** button to find VIs using that keyword.
4. Click one of the example VIs in the search results list and read the description.
5. Double-click an example VI to open it.
6. Read through the comments on the front panel and block diagram to learn more about what this example VI demonstrates.
7. Run the example, examine the different cases, and click the **Stop** button to exit.
8. Close the VIs and the NI Example Finder when you are finished.

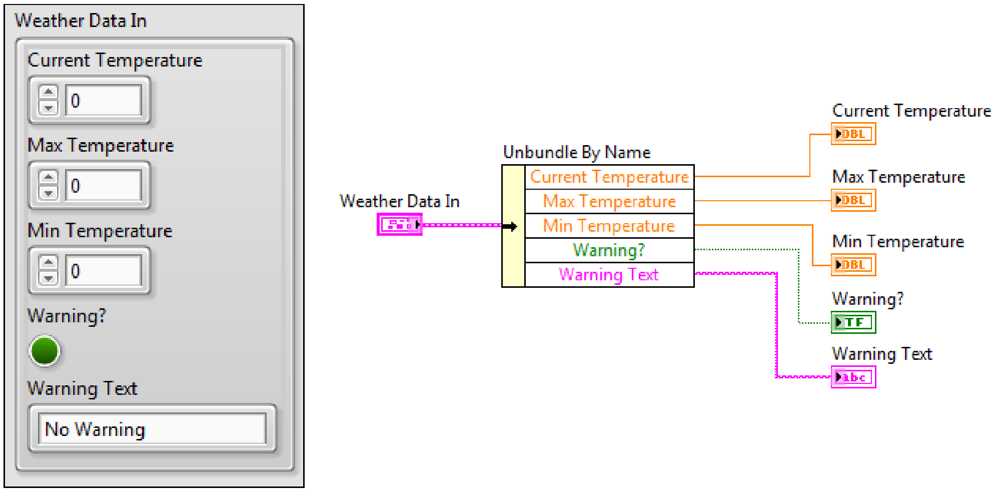
End of Exercise 5-1

E. Clusters

Objective: Identify when to use clusters and be able to create them.

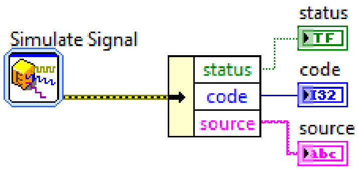
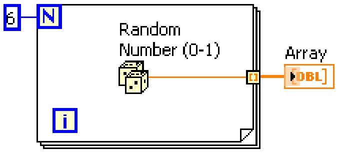
Clusters

Clusters Clusters group data elements of mixed types



A cluster is similar to a record or a struct in text-based programming languages.An example of a cluster is the LabVIEW error cluster, which combines a Boolean value, a numeric value, and a string.

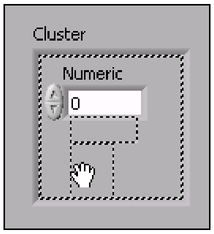
Clusters vs. Arrays

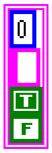
Cluster	Array
 Simulate Signal status code source status TF code I32 source abc	
• Mixed data types • Fixed size	• Vary in size • Contain only one data type

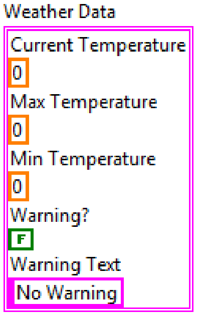
Demonstration: Create a Cluster Control

Create a cluster control or indicator on the front panel by adding a cluster shell to the front panel and dragging a data object or element into the shell. The element can be a numeric, Boolean, string, path, refnum, array, or cluster control or indicator.

Resize the cluster shell by dragging the cursor while you place the cluster shell.



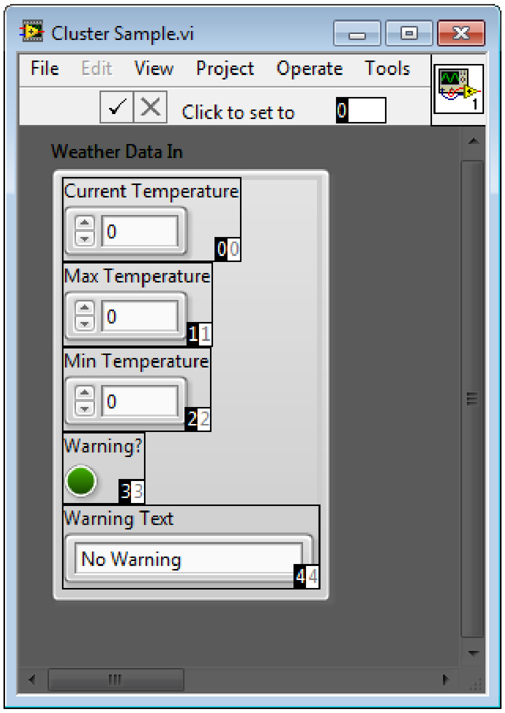




Cluster Order

Cluster elements have a logical order unrelated to their position in the shell. The cluster order determines the order in which the elements appear as terminals on the Bundle and Unbundle functions on the block diagram.

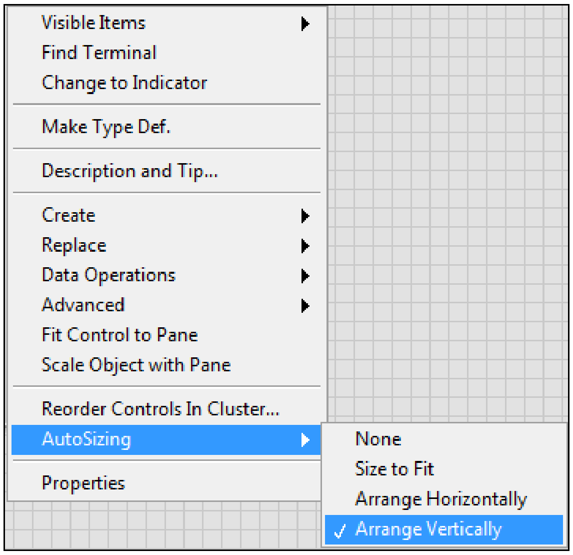
You can view and modify the cluster order by right-clicking the cluster border and selecting Reorder Controls In Cluster from the shortcut menu.



Autosizing Clusters

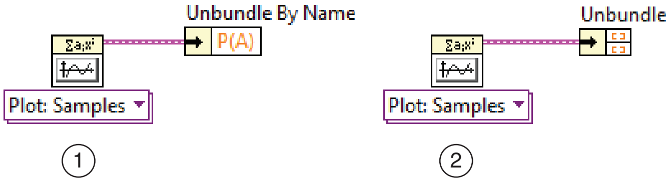
Autosizing helps you arrange elements in clusters.NI recommends the following:

- Arrange cluster elements vertically.
- Arrange elements compactly.
- Arrange elements in their preferred order.



Disassembling Clusters

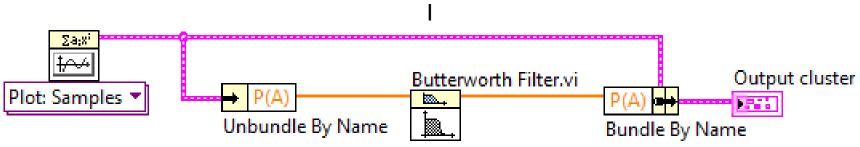
Use the Unbundle and Unbundle By Name functions to return individual cluster elements.

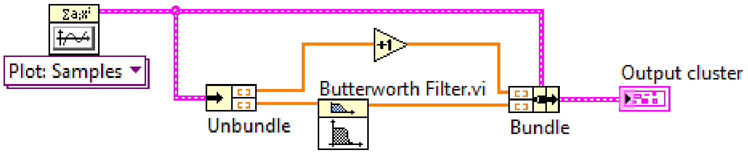


1 Unbundle By Name—Returns the cluster elements whose names you specify. The number of output terminals does not depend on the number of elements in the input cluster.
2 Unbundle—Splits a cluster into its individual elements

Modifying a Cluster

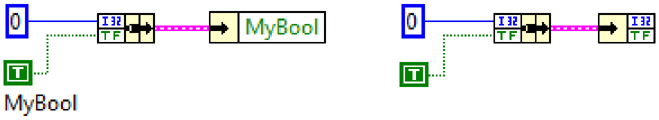
Use Bundle By Name whenever possible to access elements in a cluster. Use Bundle when some or all cluster elements are unnamed.





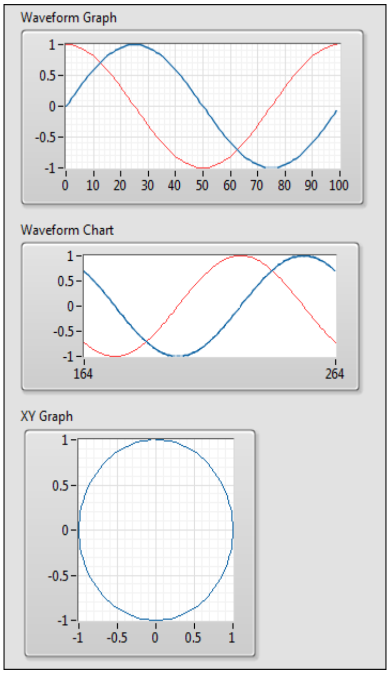
Demonstration: Creating a Cluster on the Block Diagram

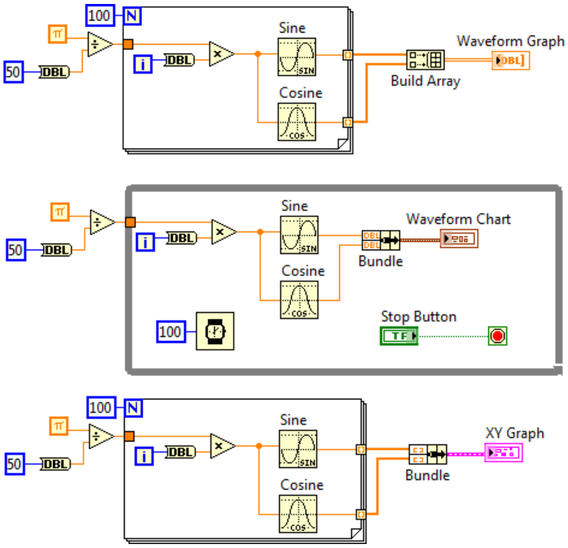
Use the Bundle function to programmatically create a cluster on a block diagram. If the elements that are bundled have labels, you can access them using the Unbundle By Name function. Otherwise use the Unbundle function.



Multi-plot Graphs/Charts and XY Graph

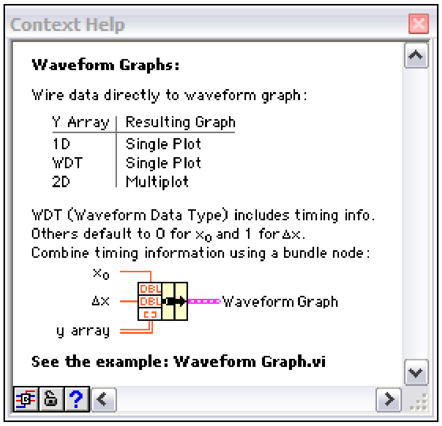
The Bundle function is often used to create multi-chart plot charts and XY plots. The Build Array function is used to create multi-plot waveform graphs.





Plotting Data

Use the Context Help window to determine how to wire multi-plot data to Waveform Graphs, Charts and XY Graphs.



Context Help

Waveform Graphs:

Wire data directly to waveform graph:

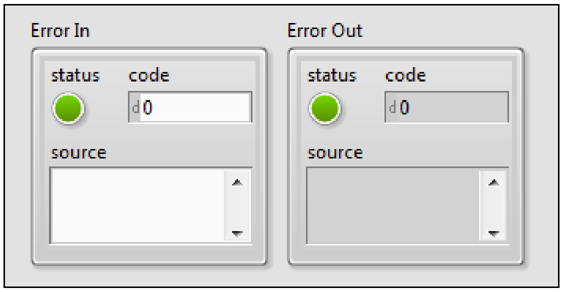
Y Array	Resulting Graph
1D	Single Plot
WDT	Single Plot
2D	Multiplot

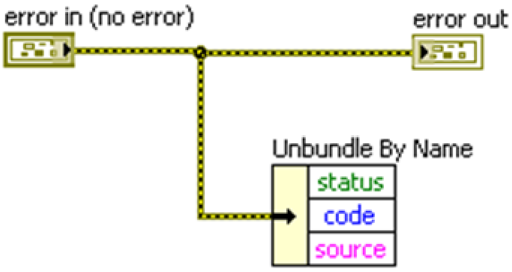
WDT (Waveform Data Type) includes timing info.
Others default to 0 for x_0 and 1 for Δx.
Combine timing information using a bundle node:

See the example: Waveform Graph.vi

Error Clusters

LabVIEW contains a custom cluster called the error cluster. LabVIEW uses error clusters to pass error information.





Exercise 5-2 Temperature Warnings VI—Clusters

Goal

Create a cluster datatype containing the data to be passed around an application and, in the process, create scalable, readable code.

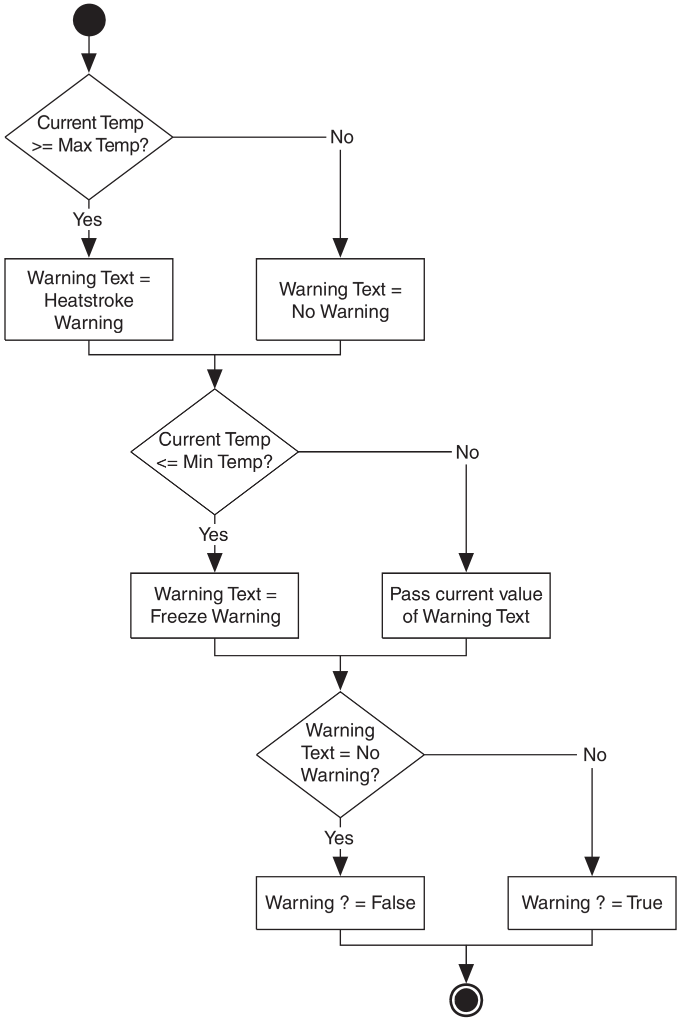
Scenario

Another developer has created a VI that displays temperature warnings. This VI is part of the temperature weather station project studied throughout this course. Your task is to update this VI to use clusters instead of individual terminals for inputs and outputs.

Design

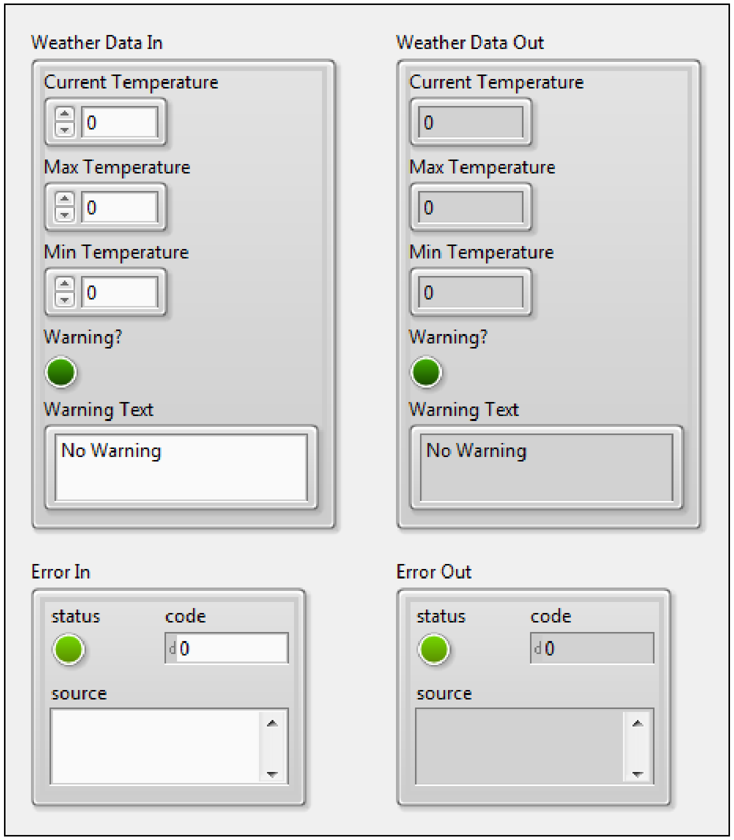
The flowchart in Figure 5-9 illustrates the data flow for the design of the Temperature Warnings VI.

Figure 5-9. Temperature Warnings VI Flowchart


Current Temp >= Max Temp?
Yes
No
Warning Text = Heatstroke Warning
Warning Text = No Warning
Current Temp <= Min Temp?
Yes
No
Warning Text = Freeze Warning
Pass current value of Warning Text
Warning Text = No Warning?
Yes
No
Warning ? = False
Warning ? = True

Create a cluster which contains the data used by the Temperature Warnings VI. You modify the Temperature Warnings VI to receive and return data in the form of that same cluster as shown in Figure 5-10. The modified VI works in a more modular fashion with other subVIs in the overall application.

Figure 5-10. Temperature Warnings VI with Clusters Front Panel



Weather Data In
Current Temperature 0
Max Temperature 0
Min Temperature 0
Warning? ●
Warning Text No Warning

Weather Data Out
Current Temperature 0
Max Temperature 0
Min Temperature 0
Warning? ●
Warning Text No Warning

Error In
status ●
code d 0
source

Error Out
status ●
code d 0
source

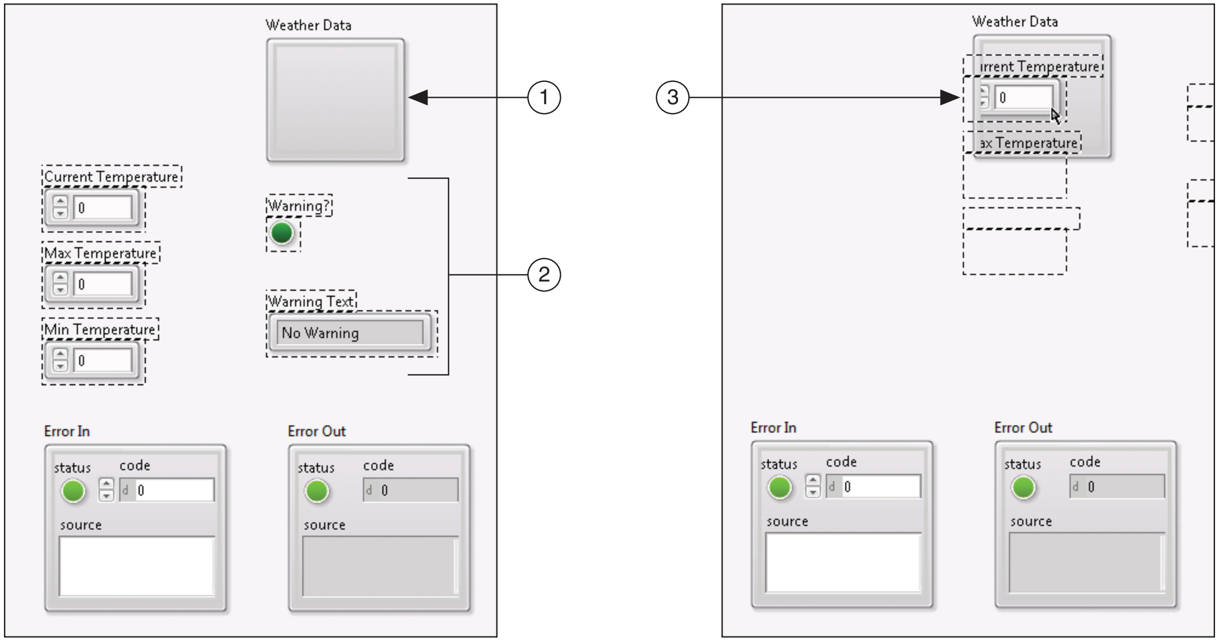
Implementation

1. Open `Weather Warnings.lvproj` in the `<Exercises>\LabVIEW Core 1\Weather Warnings` directory.

2. Open **Temperature Warnings** VI from the **Project Explorer** window.

3. Place existing controls and indicators in a cluster named `Weather Data` as shown in Figure 5-11.

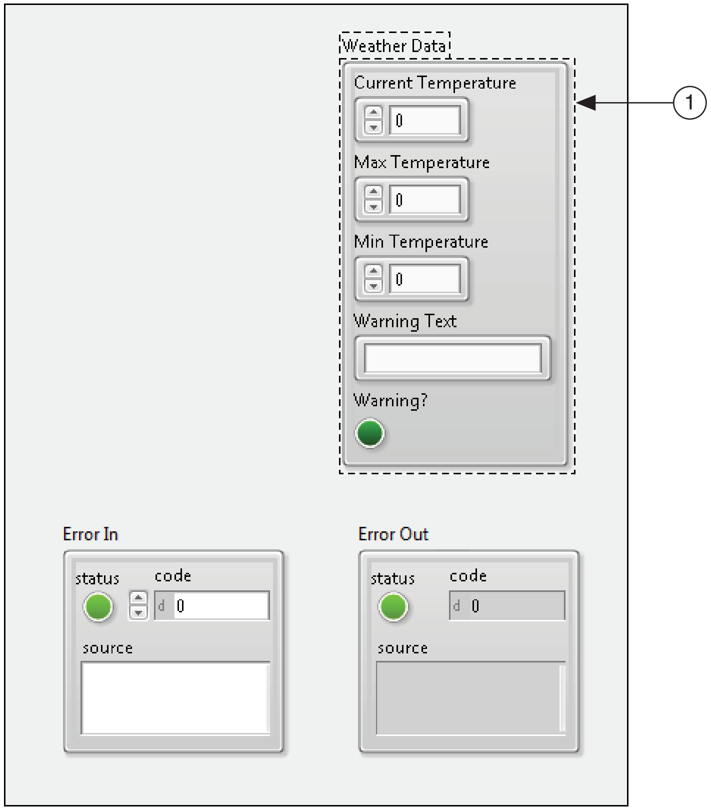
Figure 5-11. Create Cluster



1 **Cluster**—Use the Cluster control from the **Silver** 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 Figure 5-12.

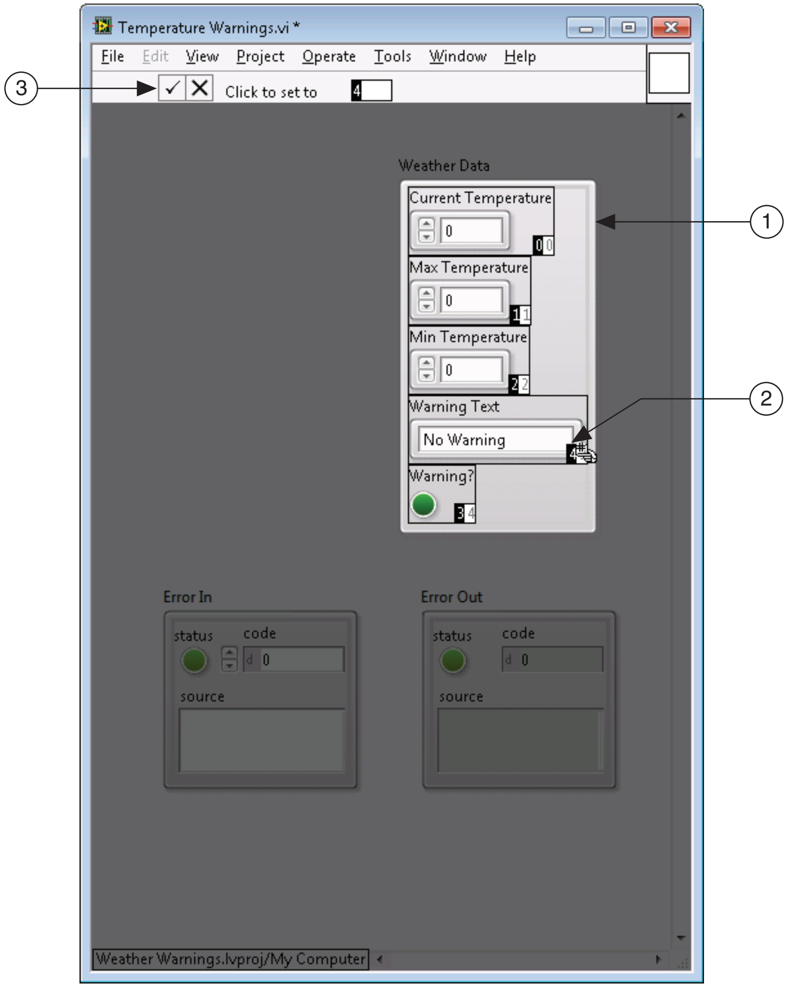
Figure 5-12. Resize Cluster Control



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 Figure 5-13.

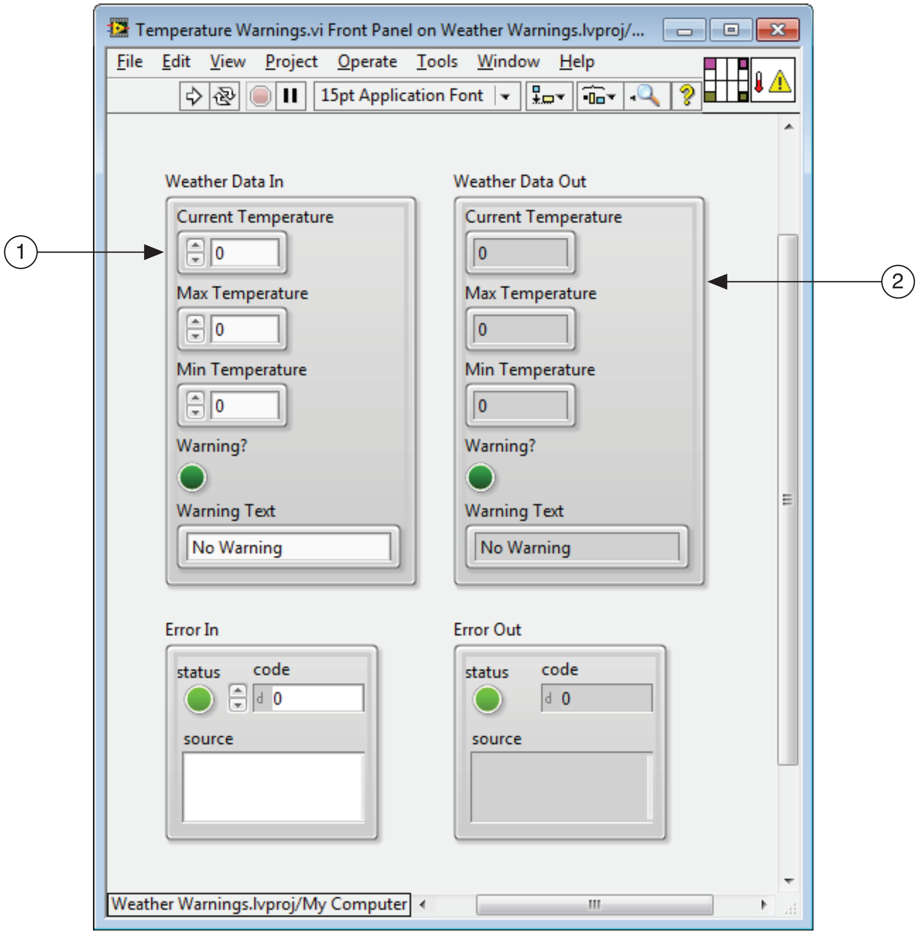
Figure 5-13. Reorder Cluster



1 Right-click the edge of the cluster and select **Reorder Controls in Cluster**.
2 Click the controls to toggle the order of the items in the cluster.
3 Click the **Confirm** button to save the changes.

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

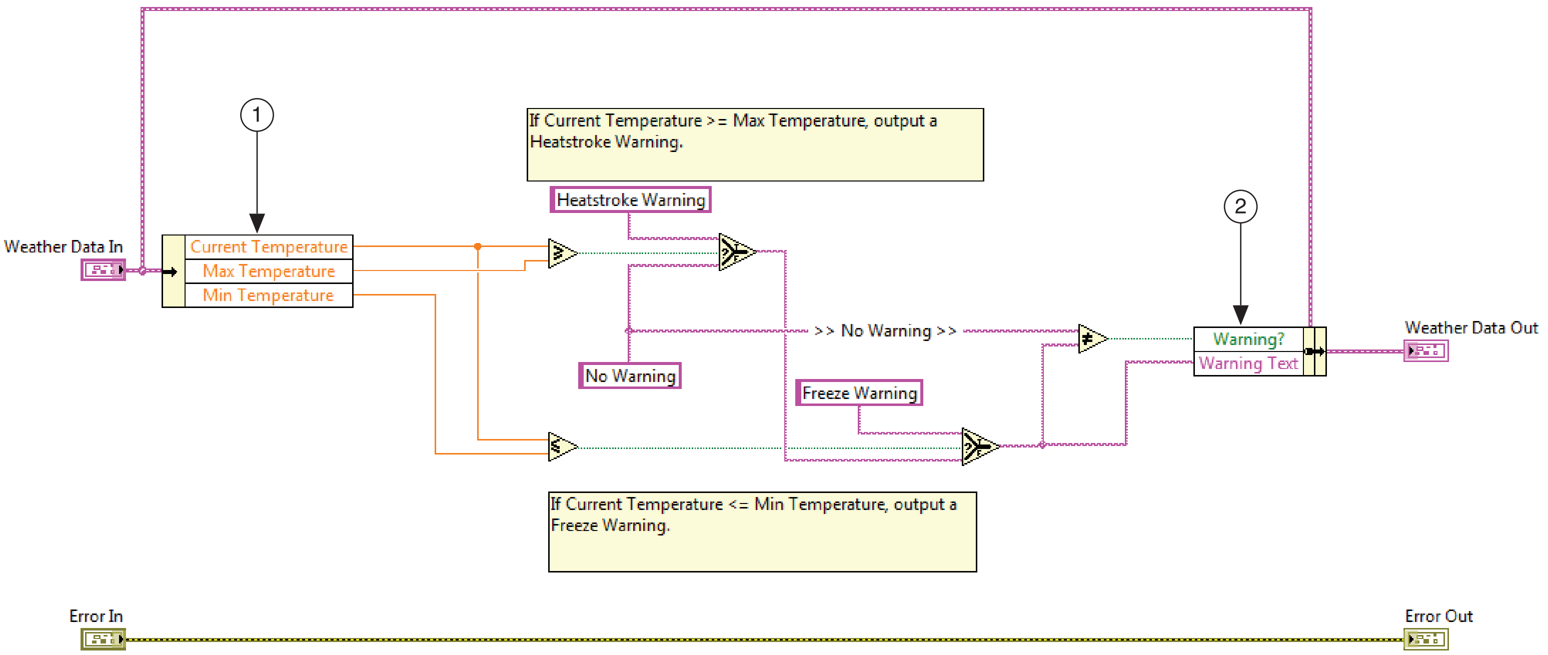
Figure 5-14. Temperature Warnings—Weather Data In and Weather Data Out Clusters



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

7. Modify the block diagram as shown in Figure 5-15 to extract data from the input cluster.

Figure 5-15. Temperature Warnings with Clusters Block Diagram



1 **Unbundle By Name**—Wire the **Weather Data In** control 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 wire the internal controls and indicators independently of each other.

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 use the Operating tool to select **Warning?** and **Warning Text** elements. Connect the broken wires to the Unbundle By Name inputs as shown.

Note If the order of the elements in the Unbundle By Name and the Bundle By Name functions is different than what you want, you can use the Operating tool to change the order.

8. Save and close the Temperature Warnings VI.

Test

1. Enter values in the **Current Temperature**, **Max Temperature** and **Min Temperature** controls in the **Weather Data In** cluster.

2. Run the VI and verify that the **Weather Data** indicator displays correct values.

3. Save and close the VI

End of Exercise 5-2

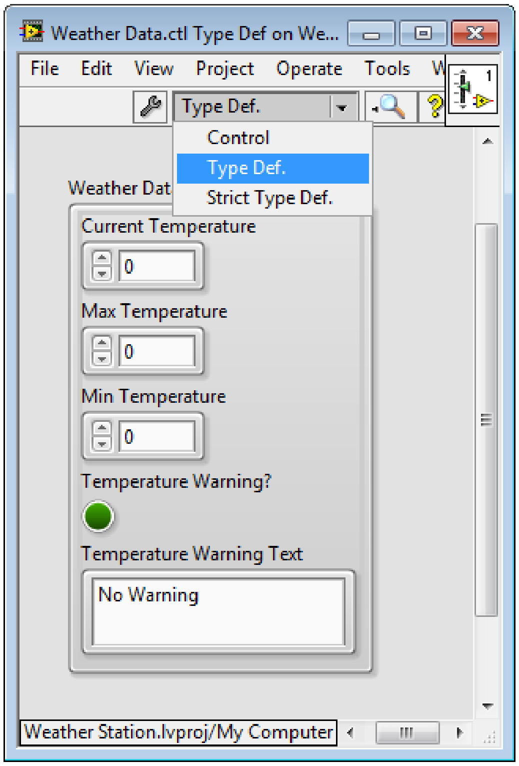
F. Type Definitions

Objective: Identify and determine when to use a type definition, strict type definition, or control.

Control Options

Use custom controls and indicators to extend the available set of front panel objects and to make them available on other front panels.

Three types of custom controls—Control, Type Definition, Strict Type Definition.



Current Temperature 0

Max Temperature 0

Min Temperature 0

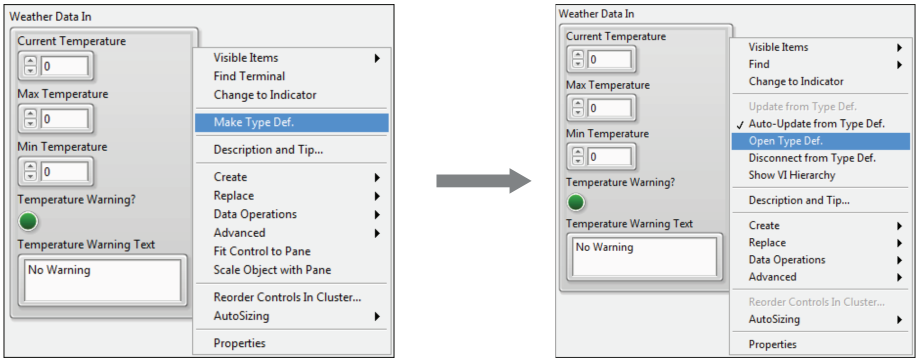
Temperature Warning?

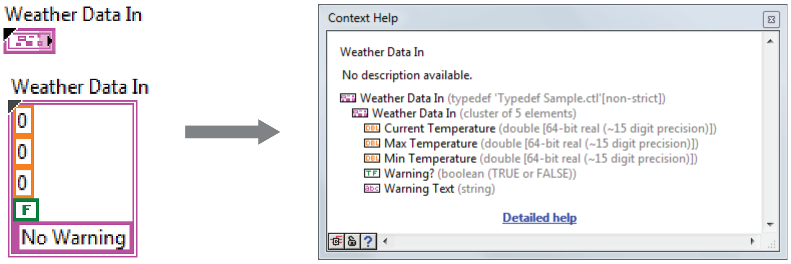
Temperature Warning Text: No Warning

Demonstration: Difference between Control, Type Def and Strict Type Def

- Custom controls are templates and used as starting points for other similar controls. Changes made to one control does not reflect in other controls.
- Type definitions and strict type definitions link to all the instances of a custom control or indicator to a saved custom control or indicator file. You can make changes to all instances of the custom control or indicator by editing only the saved custom control or indicator file
- Strict type defs apply cosmetic changes too, whereas type defs do not.

Demonstration: Creating and Identifying Type Definitions





Weather Data In

No description available.

Weather Data In (typedef 'Typedef Sample.ctl'[non-strict])
Weather Data In (cluster of 5 elements)
Current Temperature (double [64-bit real (~15 digit precision)])
Max Temperature (double [64-bit real (~15 digit precision)])
Min Temperature (double [64-bit real (~15 digit precision)])
Warning? (boolean (TRUE or FALSE))
Warning Text (string)

Detailed help

Exercise 5-3 Temperature Warnings VI—Type Definition

Goal

To improve the scalability of your application by using type definitions made from custom cluster controls, indicators, and constants of a particular data type.

Scenario

As a LabVIEW developer, you can encounter situations where you need to define your own custom data types in the form of clusters and enums. A challenge associated with using custom data types is that you may need to change them later in development. In addition, you may need to change them after they have already been used in VIs. For example, you create copies of a custom data type and use them as controls, indicators, or constants in one or more VIs. Then you realize that the custom data type needs to change. You need to add, remove, or change items in the cluster data type or the enum.

As a developer you must ask yourself the following questions:

- What should happen to the copies of the custom data types used in VIs that are already saved?
- Should the copies remain unchanged or should they update themselves to reflect changes to the original?

Usually, you want all the copies of the custom data type to update if you update the original custom data type. To achieve this you need copies of the custom data types to be tied to a 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 Temperature Warnings VI that you revised in Exercise 5-2 in such a way that the changes to the **Weather Data** custom data type propagate through the application.

When complete, the Weather Station application monitors temperature and wind information. This exercise modifies the Temperature Warnings VI. In the *Challenge* exercise, you modify the Windspeed Warnings VI.

1. Open `Weather Warnings.lvproj` in the `<Exercises>\LabVIEW Core 1\Weather Warnings` directory.

2. Open **Temperature Warnings** VI from the **Project Explorer** window.

3. Experiment with changing an existing cluster.

   □ Place a **File Path Control (Silver)** in the **Weather Data In** cluster control.

   □ Notice that the Temperature 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.

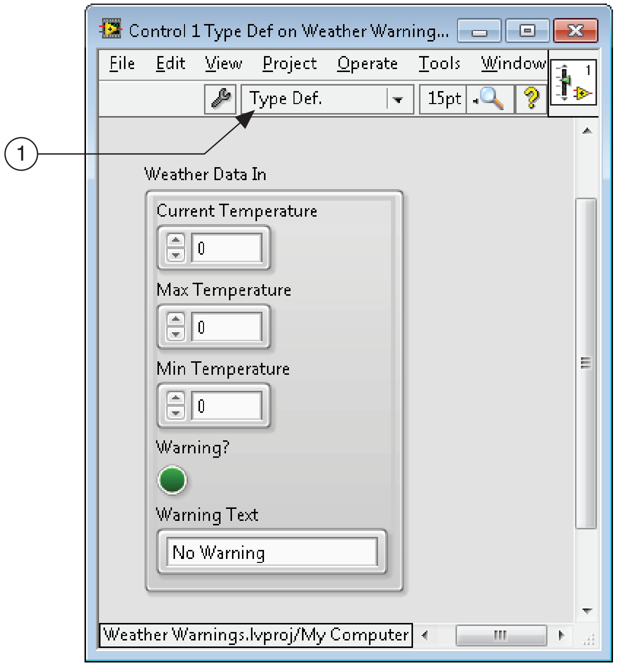
☐ Right-click the border of the **Weather Data In** control and select **Make Type Def**.

☐ On the block diagram, the **Weather Data In** terminal now has a black triangle on the corner indicating that it is connected to a type definition.

☐ Right-click the border of the **Weather Data In** control and select **Open Type Def** to display the **Custom Control Editor** window as shown in Figure 5-16.

The window looks like the front panel of a VI but it does not have a block diagram.

Figure 5-16. Custom Control Editor Window


Control 1 Type Def on Weather Warning...
File Edit View Project Operate Tools Window
Type Def.
15pt
1
Weather Data In
Current Temperature
0
Max Temperature
0
Min Temperature
0
Warning?
Warning Text
No Warning
Weather Warnings.lvproj/My Computer

1 The control type is Type Def, which maintains the link between this file and the custom control copies used in VIs.

☐ Save the custom control as `Weather Data.ctl` in the `<Exercises>\LabVIEW Core 1\Weather Warnings` directory and close the control editor window.

☐ On the block diagram of the Temperature Warnings VI, notice the coercion dot on the **Weather Data Out** indicator terminal. This indicates that the indicator is not tied to the type definition.

5. Tie the **Weather Data Out** indicator to the type definition.

   □ Right-click the border of the **Weather Data Out** indicator on the front panel and select **Replace»Select a Control** from the shortcut menu.

   □ Browse to and select the `Weather Data.ctl` file you just created.

   > **Note** You can no longer add or remove elements to or from the cluster control and indicator on the front panel. You must open the type definition and add or remove the element from the control editor window.

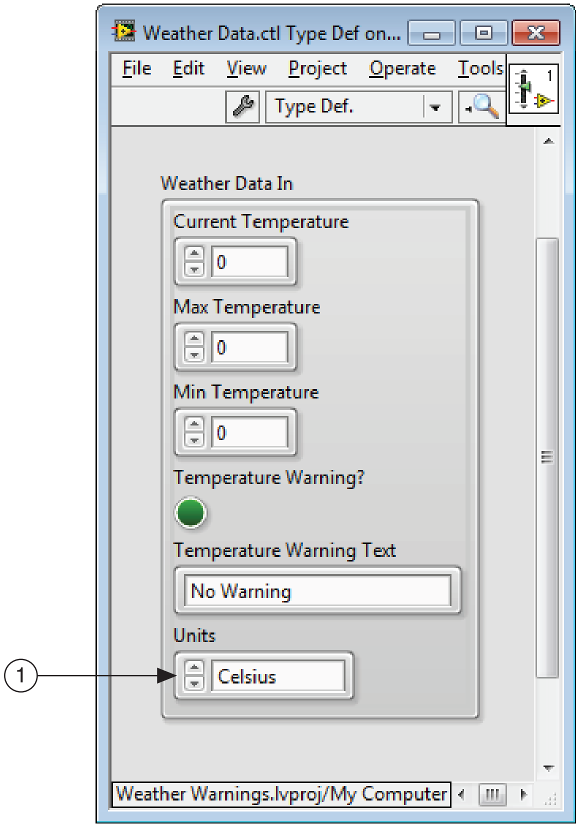
   □ Save the Temperature Warnings VI.

6. Edit the Weather Data type definition to include unit 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 Figure 5-17.

Figure 5-17. Weather Data Type Definition with Temperature Units



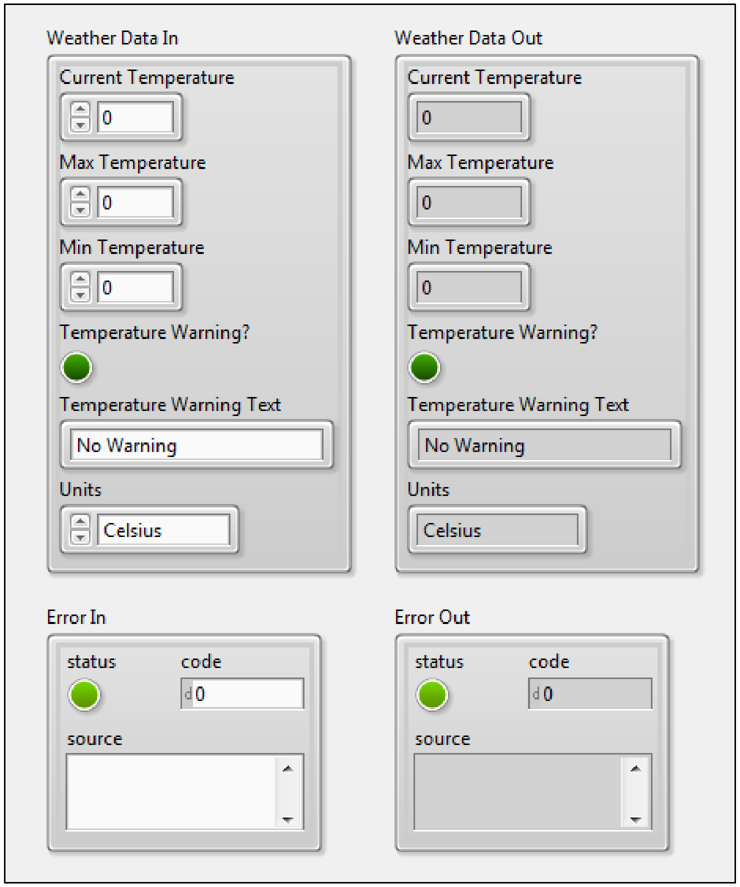
Weather Data In
Current Temperature 0
Max Temperature 0
Min Temperature 0
Temperature Warning? ●
Temperature Warning Text No Warning
Units Celsius
Weather Warnings.lvproj/My Computer

1 **Enum (Silver)**—Place an enum in the cluster and rename it `Units`. Right-click the enum and select **Edit items**. Create an item for `Celsius` and `Fahrenheit`.

   □ Save the Weather Data type definition and close the control editor window.

   □ Notice that the **Weather Data In** control and **Weather Data Out** indicator on the Temperature Warnings VI have been updated with the changes you made to the Weather Data type definition. Arrange the front panel of the VI as shown in Figure 5-18.

Figure 5-18. Temperature Warnings VI with Type Def Controls and Indicators



7. Run and Save the Temperature Warnings VI.

Challenge

In this challenge exercise, you modify the Windspeed Warnings VI to augment the Weather Station application.

1. Add the Windspeed Warnings VI to the Weather Warnings project.

   □ In the Project Explorer window, right click **My Computer** and select **Add»File** from the shortcut menu.

   □ Navigate to `<Exercises>\LabVIEW Core 1\Weather Warnings\Support VIs` and select `Windspeed Warnings.vi`.

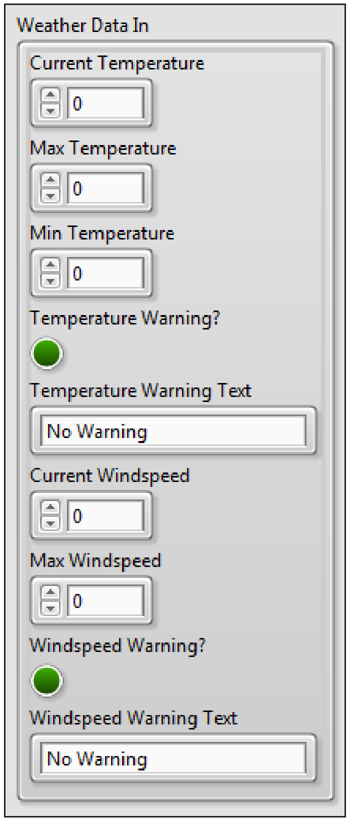
2. Open the Windspeed Warnings VI.

3. Copy the **Weather Data In** cluster from the Temperature Warnings VI to the Windspeed Warnings VI.

4. Right-click the **Weather Data In** cluster and select **Open Type Def** from the shortcut menu.

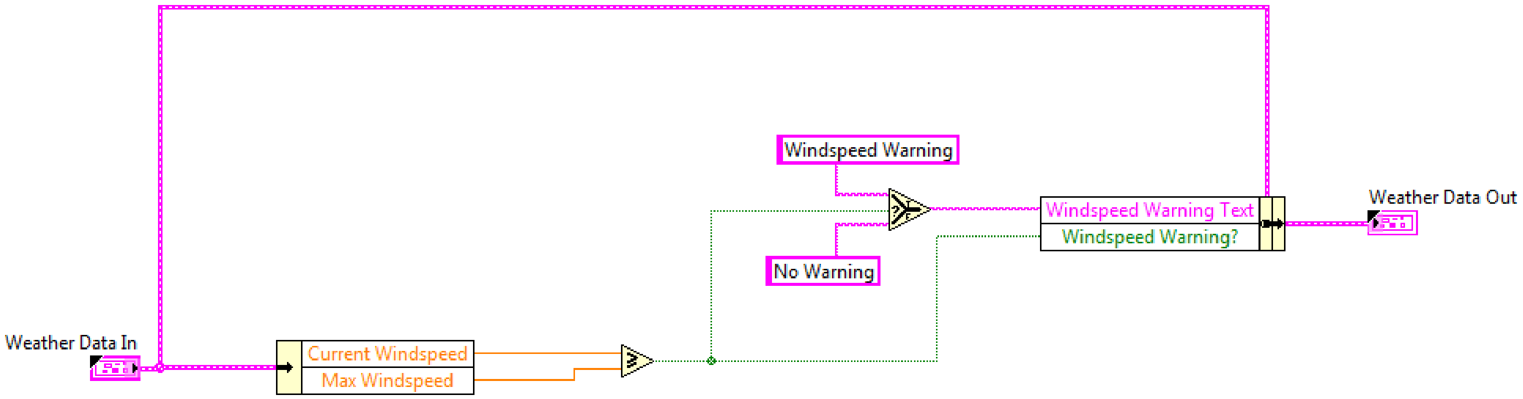
5. Modify the Weather Data type definition with windspeed controls as shown in Figure 5-19.

Figure 5-19. Windspeed Warnings VI Type Definition Controls and Indicators



6. Modify the block diagram of the Windspeed Warnings VI to use the new Weather Data type definition instead of individual controls and indicators, as shown in Figure 5-20.

Figure 5-20. Windspeed Warnings VI Using Type Definitions



7. Open Temperature Warnings VI and notice that the **Weather Data In** control and **Weather Data Out** indicator is updated to include the Windspeed data.

8. Save and close the VI and the project.

End of Exercise 5-3

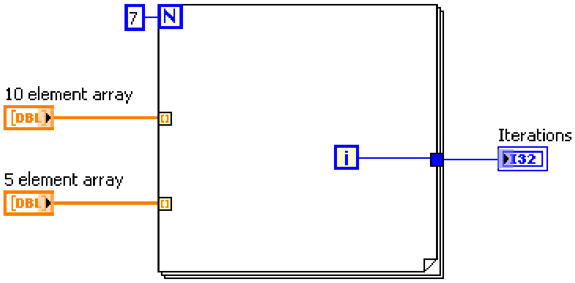
Additional Resources

Learn More About	LabVIEW Help Topic
Arrays	*Adding Elements to Arrays* *Changing Array Default Values* *Creating Array Controls and Indicators* *Default Sizes and Values of Arrays* *Determining the Size of Arrays*
Clusters	*Creating Cluster Controls and Indicators* *Modifying Cluster Element Order* *Moving Arrays and Clusters* *Setting Cluster Default Values* *Tabbing through Elements of an Array or Cluster*
Type Definitions	*Creating Type Definitions and Strict Type Definitions*

Activity 5-2: Lesson Review

1. You can create an array of arrays.
   a. True
   b. False

2. You have two input arrays wired to a For Loop. Auto-indexing is enabled on both tunnels. One array has 10 elements, the second array has five elements. A value of 7 is wired to the Count terminal, as shown in the following figure. What is the value of the **Iterations** indicator after running this VI?



3. Which of the following custom control settings defines the data type of all instances of a control but allows for different colors and font styles?
   a. Control
   b. Type Definition
   c. Strict Type Definition
   d. Cluster control

4. You have input data representing a circle: X Position (I16), Y Position (I16), and Radius (I16). In the future, you might need to modify your data to include the color of the circle (U32).

   What data structure should you use to represent the circle in your application?
   a. Three separate controls for the two positions and the radius.
   b. A cluster containing all of the data.
   c. A custom control containing a cluster.
   d. A type definition containing a cluster.
   e. An array with three elements.

Activity 5-2: Lesson Review - Answers

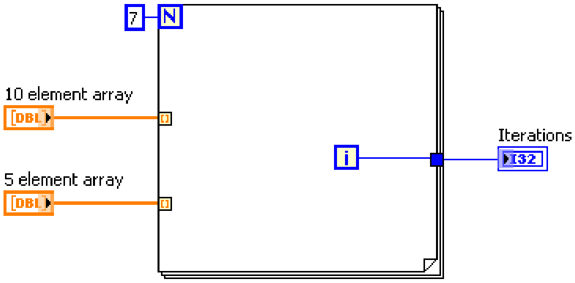
1. You can create an array of arrays.

   a. True

   b. False

   You cannot drag an array data type into an array shell. However, you can create two-dimensional arrays.

2. You have two input arrays wired to a For Loop. Auto-indexing is enabled on both tunnels. One array has 10 elements, the second array has five elements. A value of 7 is wired to the Count terminal, as shown in the following figure. What is the value of the **Iterations** indicator after running this VI?



   Value of Iterations = 4

   LabVIEW does not exceed the array size. This helps to protect against programming error. LabVIEW mathematical functions work the same way—if you wire a 10 element array to the **x** input of the Add function, and a 5 element array to the **y** input of the Add function, the output is a 5 element array.

   Although the for loop runs 5 times, the iterations are zero based, therefore the value of the Iterations indicators is 4.

3. Which of the following custom control settings defines the data type of all instances of a control but allows for different colors and font styles?

   a. Control

   b. Type Definition

   c. Strict Type Definition

   d. Cluster control

4. You have input data representing a circle: X Position (I16), Y Position (I16), and Radius (I16). In the future, you might need to modify your data to include the color of the circle (U32).

   What data structure should you use to represent the circle in your application?

   a. Three separate controls for the two positions and the radius.

   b. A cluster containing all of the data.

   c. A custom control containing a cluster.

   d. A type definition containing a cluster.

   e. An array with three elements.

6 Using Decision-Making Structures

In this lesson you will learn to create different decision-making structures and be able to identify applications where using these structures can be beneficial.

Topics

- Case Structures
- Event-Driven Programming

Exercises

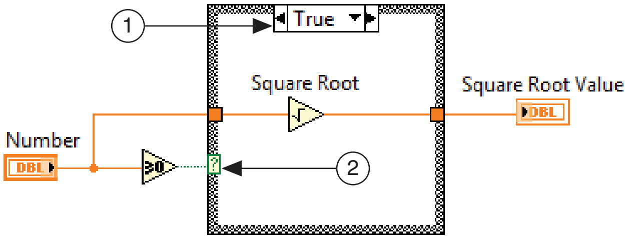
Exercise 6-1	Temperature Warnings With Error Handling
Exercise 6-2	Converting a Polling Design to an Event Structure Design

A. Case Structures

Objective: Recognize and use the basic features and functionality of Case Structures.

Activity 6-1: Case Structures Review

Figure 6-1. Case Structures Review



1 Case Selector Label
2 Selector Terminal

1. What is the purpose of the Case Structure?
   a. Execute one of its subdiagrams based on an input value
   b. Repeat a section of code until a condition occurs
   c. Execute a subdiagram a set number of times

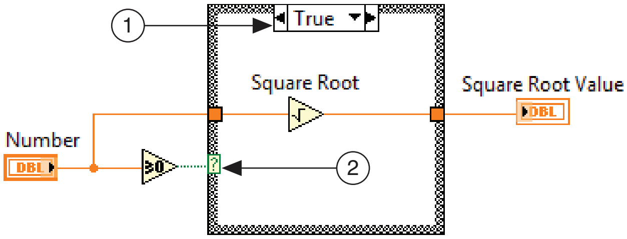
2. How many of its cases does a Case Structure execute at a time?
   a. All of them
   b. One

3. What is the purpose of the Case Selector Label?
   a. Lets you wire an input value to determine which case executes
   b. Show the name of the current state and enable you to navigate through different cases

4. What is the purpose of the Selector Terminal?
   a. Lets you wire an input value to determine which case executes
   b. Show the name of the current state and enable you to navigate through different cases

Activity 6-1: Case Structures Review - Answers

Figure 6-2. Case Structures Review: Answers

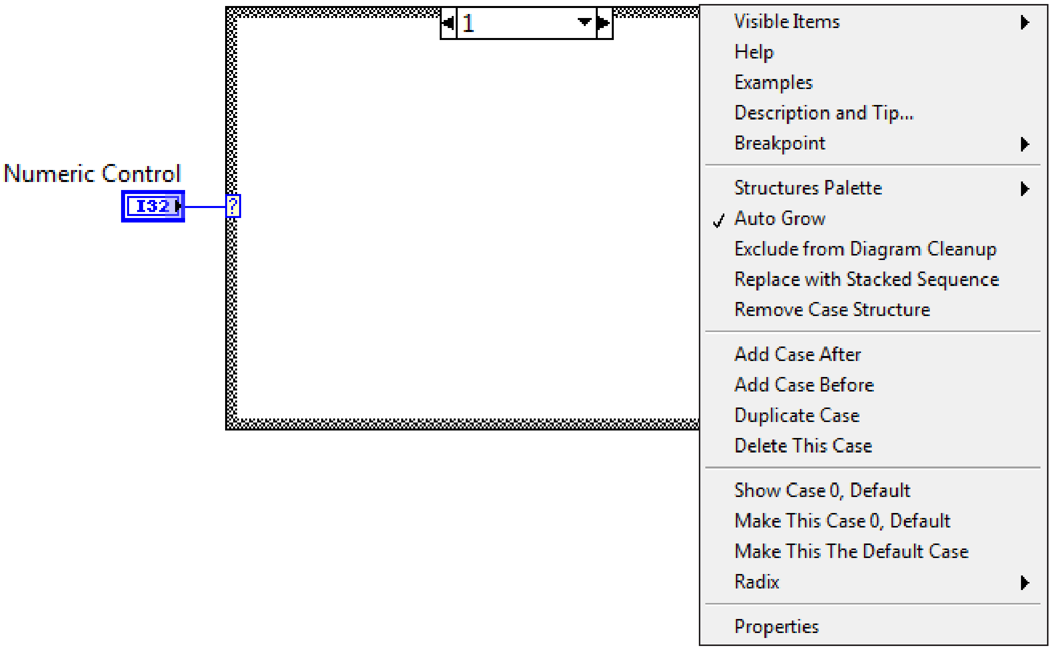


1	Case Selector Label	2	Selector Terminal

1. What is the purpose of the Case Structure?
   - **a. Execute one of its subdiagrams based on an input value**
   - b. Repeat a section of code until a condition occurs
   - c. Execute a subdiagram a set number of times
2. How many of its cases does a Case Structure execute at a time?
   - a. All of them
   - **b. One**
3. What is the purpose of the Case Selector Label?
   - a. Lets you wire an input value to determine which case executes
   - **b. Show the name of the current state and enable you to navigate through different cases**
4. What is the purpose of the Selector Terminal?
   - **a. Lets you wire an input value to determine which case executes**
   - b. Show the name of the current state and enable you to navigate through different cases

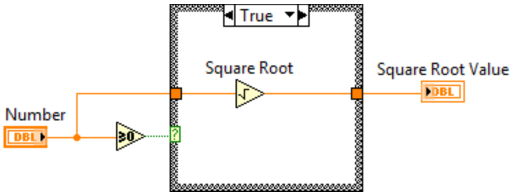
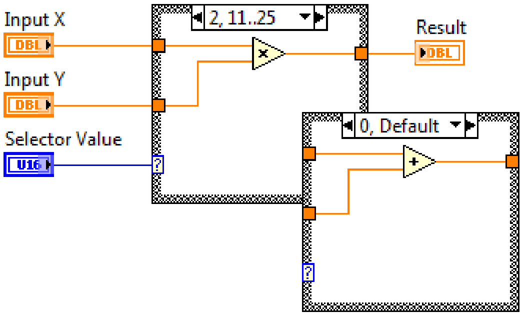
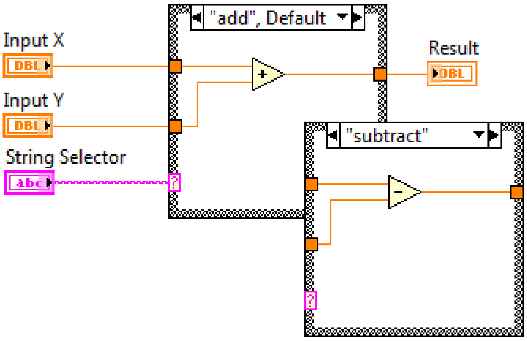
Demonstration: Case Structures

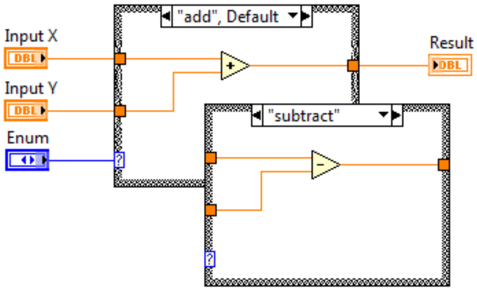
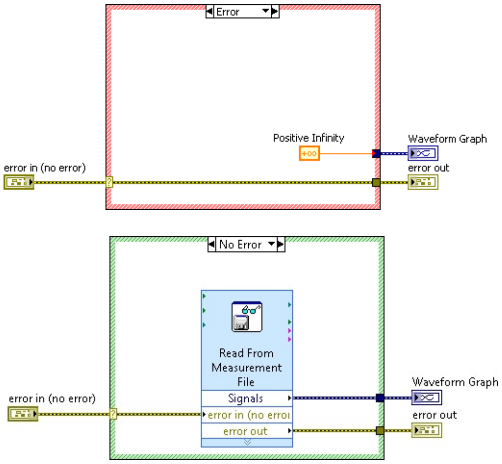
Right-click the Case structure to display the shortcut menu. The shortcut menu gives you options for configuring a Case structure.



Selector Terminal Data Types

You can wire a variety of data types to the Selector Terminal. The Case structure configuration changes based on the type of data that you connect.

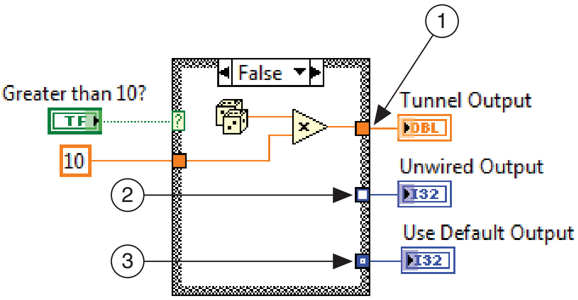
Data Type	Example
Boolean A newly-created Case structure defaults to a Boolean input. The Case Structure includes a True case and a False Case.	 True Square Root Square Root Value DBL Number DBL ≥0
Integer Case Structure has any number of cases. Specify a Default case. The numeric representation of the integer input will determine the range of possible values for the Case Selector Label. You can specify ranges of values for the Case Selector Label. Use Radix option in shortcut menu to specify whether the Case Selector Label displays values in decimal, hexadecimal, octal, or binary.	 Input X DBL Input Y DBL Selector Value U16 2, 11..25 Result DBL 0, Default
String Case Structure has any number of cases. Specify a Default case. By default, string values are case sensitive. Shortcut menu includes option for **Case Insensitive Match** for the string text.	 Input X DBL Input Y DBL String Selector abc "add", Default Result DBL "subtract"

Data Type	Example
Enum Possible to ensure that the Case Structure includes a case for every item in the enum. Right-click the border of the Case Structure and select **Add Case for Every Value** to create a case for every item.	 "add", Default Input X DBL Input Y DBL Enum "subtract" Result DBL
Error Cluster Case Structure includes an Error Case and a No Error Case. Wire an error cluster to the terminal to execute code if there is no error and skip code if there is an error.	 Error Positive Infinity +∞ Waveform Graph error out error in (no error) No Error Read From Measurement File Signals error in (no error) error out Waveform Graph error out error in (no error)

Input and Output Tunnels

As with other types of structures, you can create multiple input and output tunnels.

- Input tunnels are available to all cases if needed.
- Output tunnels require that you define a value for each case.
- Be cautious using the Use Default If Unwired option.
  - Adds a level of complexity to the code.
  - Can complicate debugging your code.
  - The default value may not be the value that you expect.

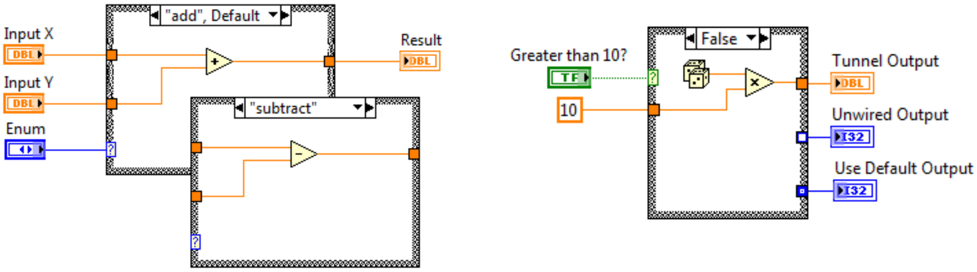


1 A tunnel that has been wired for all cases.
2 A tunnel that has not been wired.
3 A tunnel that is marked as Use Default If Unwired. Right-click on the tunnel and select **Use Default if Unwired**.

Data Type Wired to Tunnel	Default Value
Numeric	0
Boolean	FALSE
String	empty ("")

Demonstration: Selector Terminal Types and Tunnels

- Create Case structures using different data type selectors.
- Create different types of output tunnels.



Exercise 6-1 Temperature Warnings With Error Handling

Goal

Modify a VI to use a Case structure to make a software decision.

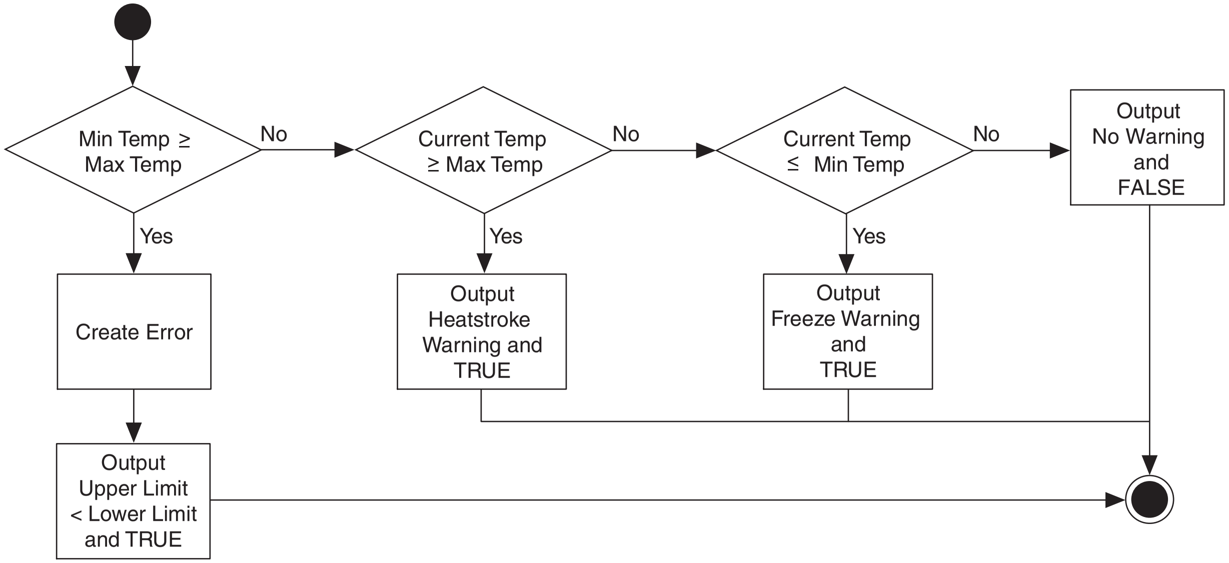
Scenario

You created a VI where a user inputs a temperature, a maximum temperature, and a minimum temperature. A warning string generates depending on the relationship of the given inputs. However, a situation could occur that causes the VI to work incorrectly. For example, the user could enter a maximum temperature that is less than the minimum temperature. Modify the VI to generate a different string to alert the user to the error: **`Upper Limit < Lower Limit`**. Set the **Warning?** indicator to TRUE to indicate the error.

Design

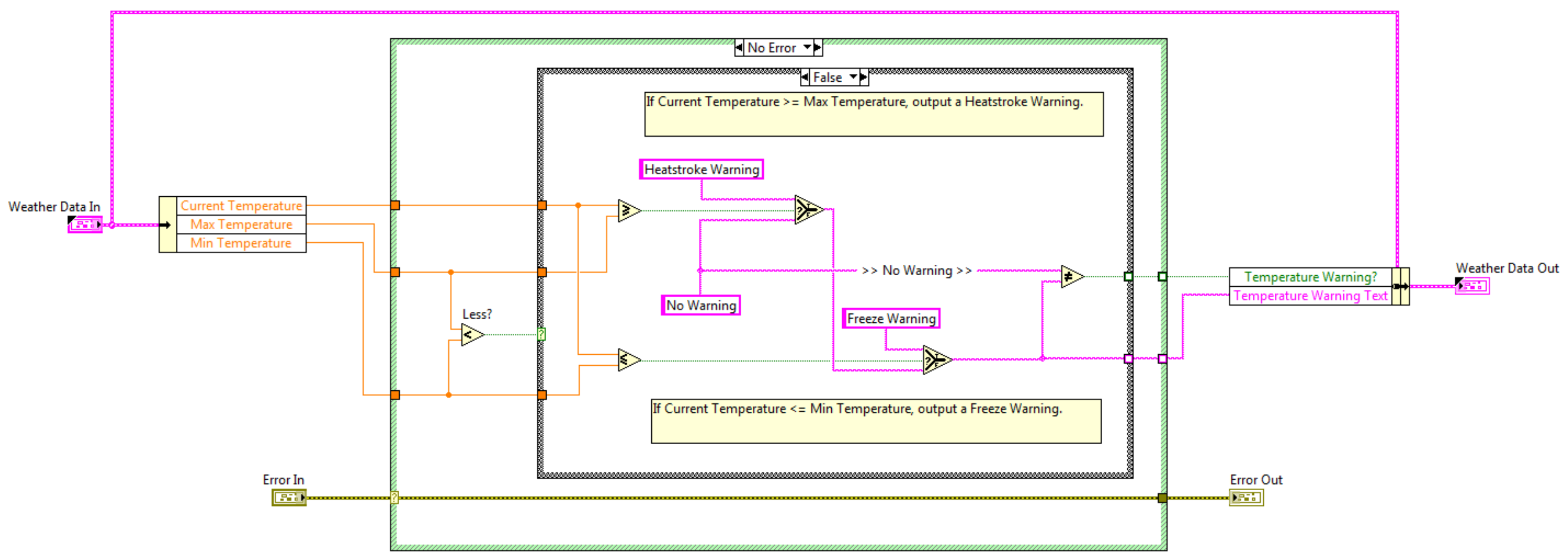
Modify the flowchart created for the original Temperature Warnings VI as shown in Figure 6-3.

Figure 6-3. Modified Temperature Warnings Flowchart


Min Temp ≥ Max Temp
No
Current Temp ≥ Max Temp
No
Current Temp ≤ Min Temp
No
Output No Warning and FALSE
Yes
Yes
Yes
Create Error
Output Heatstroke Warning and TRUE
Output Freeze Warning and TRUE
Output Upper Limit < Lower Limit and TRUE

You must add a Case structure to the Temperature Warnings VI to execute the code if the maximum temperature is less than or equal to the minimum temperature. Otherwise, the VI does not execute the code. Instead, the VI generates a new string and the **Warning?** indicator is set to TRUE.

Figure 6-4. Original Temperature Warnings VI Block Diagram



Implementation

1. Open `Weather Warnings.lvproj` in the `<Exercises>\LabVIEW Core 1\Weather Warnings` directory.

2. Open **Temperature Warnings.vi** from the **Project Explorer** window.

3. Open the block diagram and create space to add the Case structure.

   □ Select the **Weather Data In type-defined cluster terminal, the Unbundle by Name function, and the Error In terminal**.

   Tip To select more than one item press the <Shift> key while you select the items.

   □ While the objects are still selected, use the left arrow key on the keyboard to move the controls to the left.

   Tip Press and hold the <Shift> key to move the objects in five pixel increments.

   Tip Press the <Ctrl> key and use the Positioning tool to drag out a region of the size you want to insert.

   □ Select the **Weather Data Out type-define cluster terminal, the Bundle by Name function, and the Error Out terminal**.

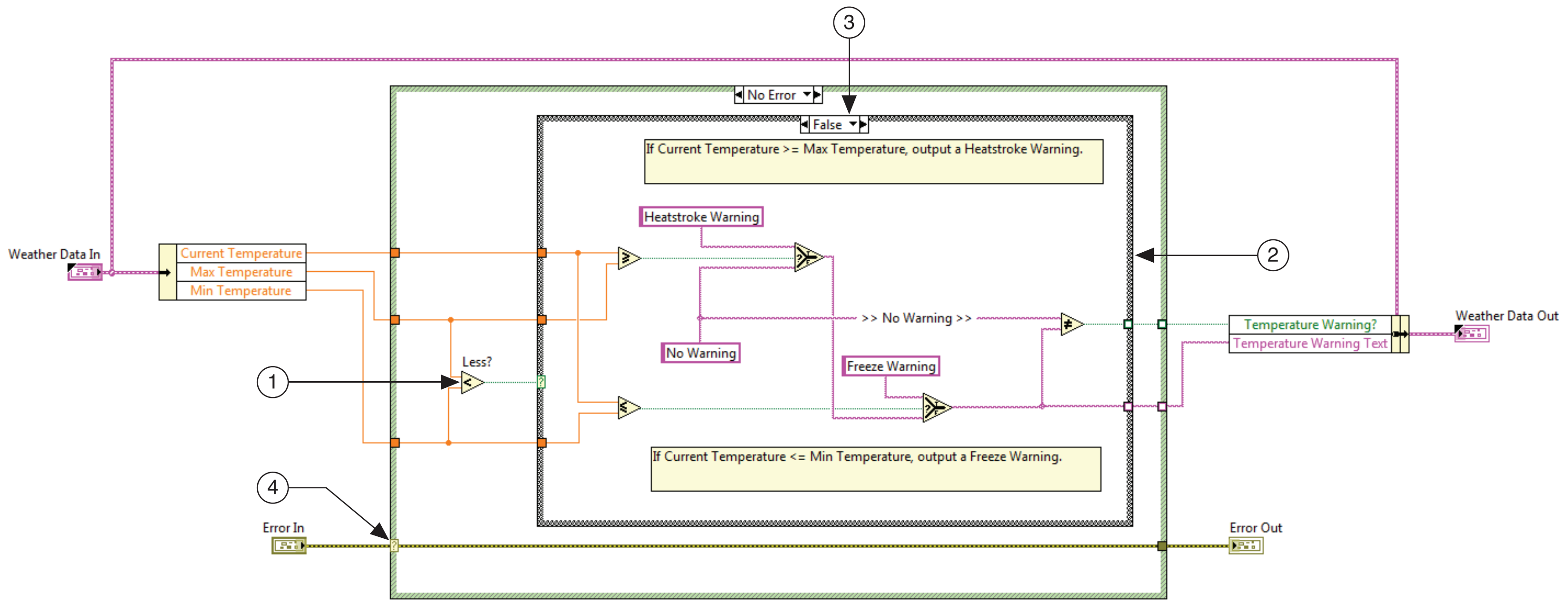
☐ While the terminals are still selected, use the right arrow key on the keyboard to move the indicators to the right.

☐ Select the wire connecting the **Weather Data In** terminal and the **Bundle by Name** function.

☐ While the wire is still selected, use the up arrow key on the keyboard to move the wire upward.

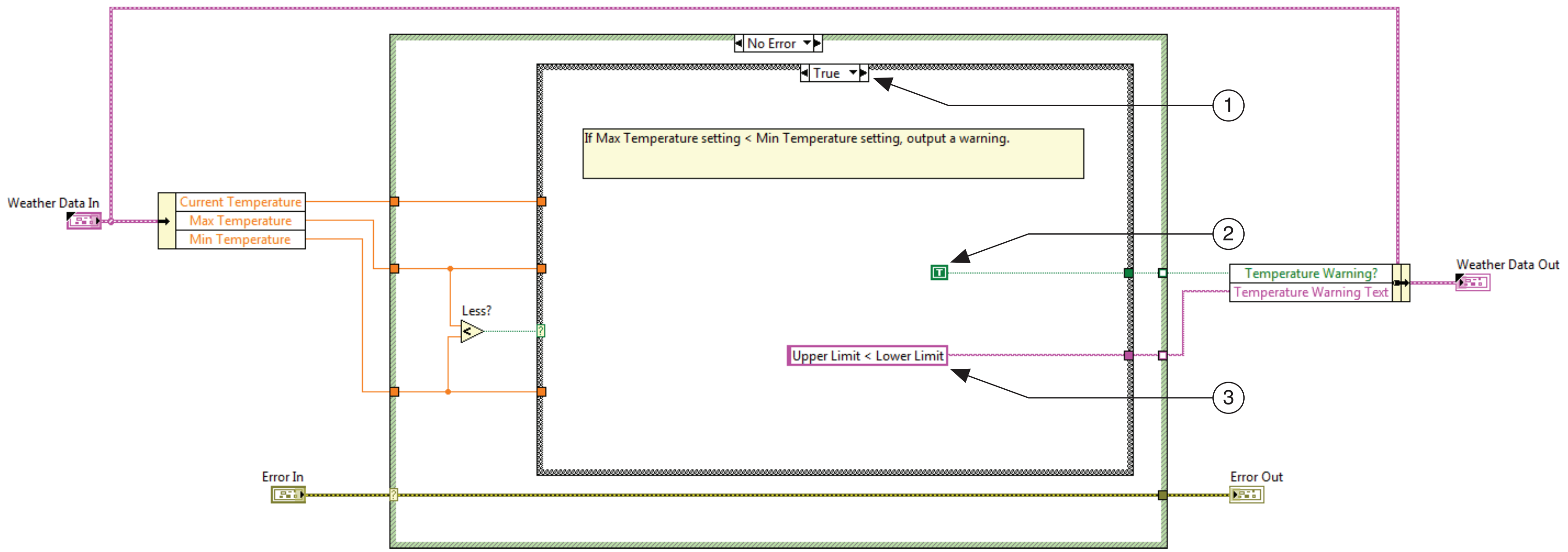
4. Modify the block diagram similar to that shown in Figure 6-5, Figure 6-6, and Figure 6-7. This VI is part of the temperature weather station project.

Figure 6-5. Temperature Warnings VI Block Diagram—No Error, False Case



1 **Less?**—Compares the Max Temperature and Min Temperature. Make sure the Less? function is outside the Case structure.
2 **Case Structure**—Do not include the **Weather Data In**, **Error In**, **Weather Data Out**, or **Error Out** terminals in the Case structure because these controls and indicators are used by both cases.
3 Set True and False cases—With the True case visible, right-click the border of the Case structure and select **Make this Case False**.
4 **Case Structure**—Wire the **Error In** terminal to the selector terminal to create No Error and Error cases. By default, the Case structure has True and False cases. These cases change to Error and No Error cases only after you wire Error In to the selector terminal.

Figure 6-6. Temperature Warnings VI—No Error, True Case



1 True case—If the Max Temperature is set lower than the Min Temperature, the True case executes. Click the case selector label to choose the True case.
2 **True Constant**—When the True case executes, the **Temperature Warning?** LED illuminates in the **Weather Data Out** cluster.
3 **String Constant**—If the Max Temperature is set lower than the Min Temperature, the warning `Upper Limit < Lower Limit` displays on the front panel. Enter the text in the String Constant.

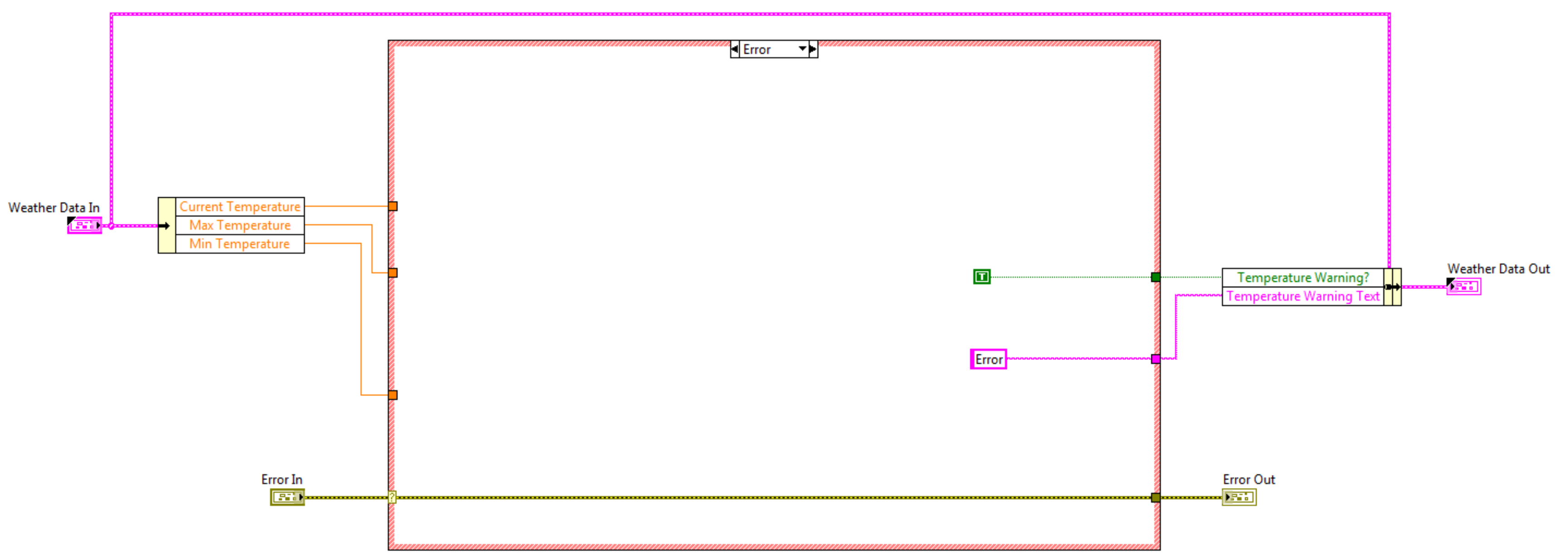
5. Predict the values for Temperature Warning Text and Temperature Warning? given each set of inputs.

Table 6-1. Predict Values for Temperature Warnings VI

Current Temperature	Max Temperature	Min Temperature	Temperature Warning Text	Temperature Warning?
30	30	10		
25	30	10		
10	30	10		
25	20	30		

6. Create the Error case in the outer Case structure so this VI can be used as a subVI.

Figure 6-7. Temperature Warnings VI—Error Case



7. Save the VI.

Test

1. Switch to the front panel of the VI.

2. Test the VI by entering values from Table 6-2 in the **Current Temperature**, **Max Temperature**, and **Min Temperature** controls and running the VI for each set of data.

Table 6-2 shows the expected Temperature Warning Text and Temperature Warning? Boolean value for each set of data.

Table 6-2. Testing Values for Temperature Warnings VI

Current Temperature	Max Temperature	Min Temperature	Temperature Warning Text	Temperature Warning?
30	30	10	`Heatstroke Warning`	True
25	30	10	`No Warning`	False
10	30	10	`Freeze Warning`	True
25	20	30	`Upper Limit < Lower Limit`	True

□ Do these values match the values that you predicted?

□ What happens if you set the value for all three inputs to 10?

□ How could you address this problem?

3. Test the Error case. To use this VI as a subVI, the VI must be able to handle an error coming into the VI. Test the Error case to make sure that this VI can output the error information it receives.

   □ On the front panel, use the Operating tool to click the **status** Boolean indicator inside the **Error In** cluster so that the indicator turns red and enter `7` in the **code** control.

   □ Run the VI. The error information you entered passes through the Error case in the VI and is output in the **Error Out** cluster.

   □ Display the block diagram, select the No Error case, highlight execution, and then run the VI again to see the error pass through the Error case.

   □ On the front panel, right-click the border of the **Error Out** cluster and select **Explain Error** to display information about the error that was returned.

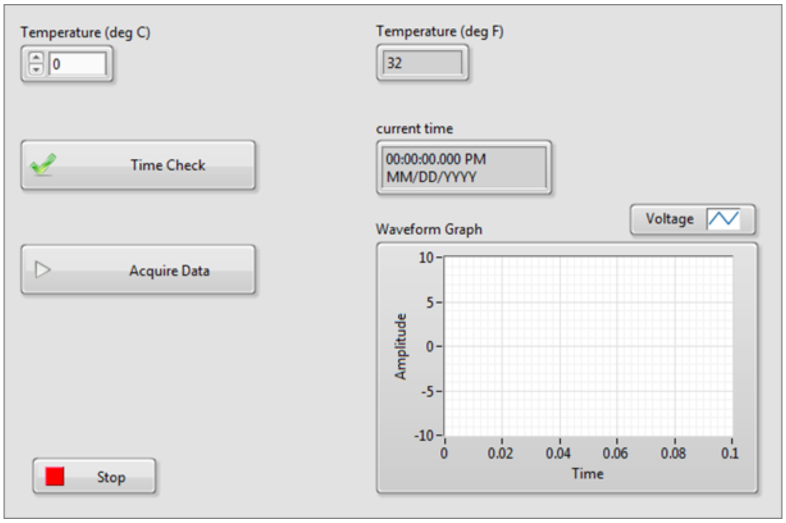
4. Save and close the VI.

End of Exercise 6-1

B. Event-Driven Programming

Objective: Recognize basic features and functionality of event structures.

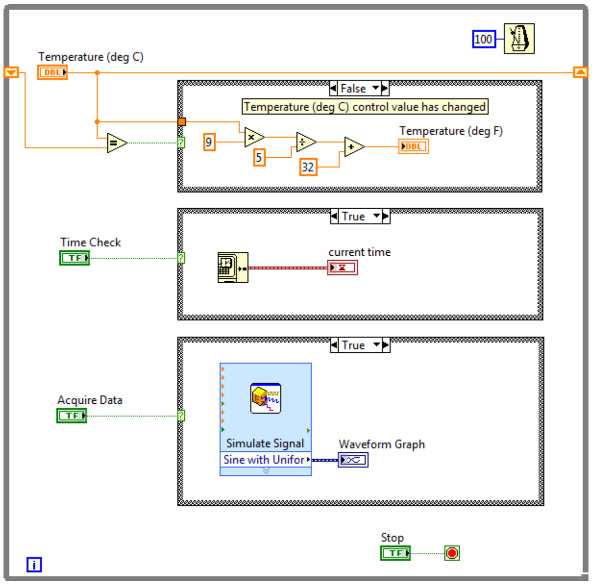
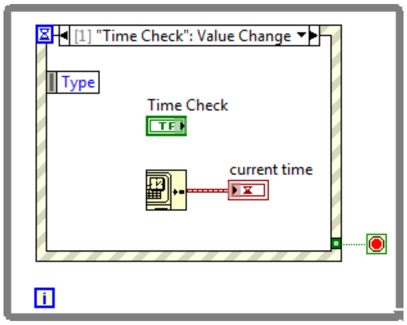
Demonstration: Event-Driven Scenario



Event-driven programming Method of programming where the program waits on an event to occur before executing the code written to handle that event.

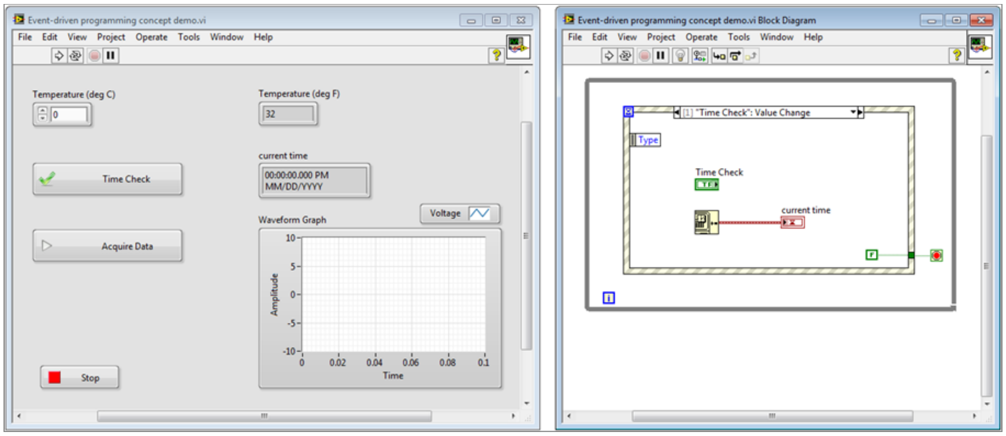
Event An asynchronous notification that something has occurred. Events can originate from the user interface, external I/O, or other parts of the program. In this course, you will only learn about user interface events, which include mouse clicks, key presses, and value changes of a control.

Polling Versus Events

Polling	Event
 100 Temperature (deg C) DBL False Temperature (deg C) control value has changed Temperature (deg F) 9 5 32 True Time Check current time Acquire Data Simulate Signal Sine with Unifor Waveform Graph Stop	 [1] "Time Check": Value Change Type Time Check current time

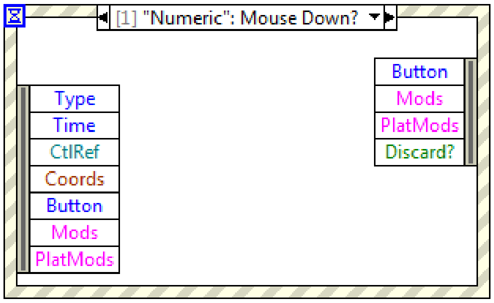
Multimedia: Event-Driven Programming

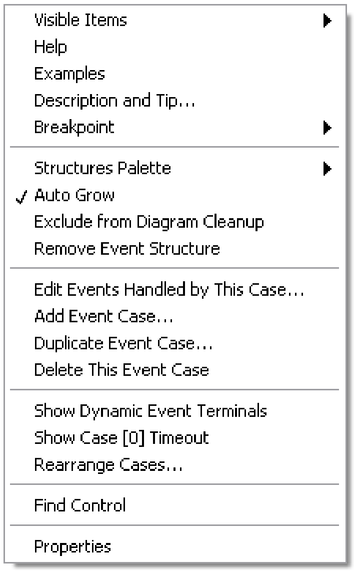
Complete the multimedia module, *Event-Driven Programming*, available in the `<Exercises>\LabVIEW Core 1\Multimedia\` folder to learn about programming with events.


Event-driven programming concept demo.vi
File Edit View Project Operate Tools Window Help
Temperature (deg C)
0
Temperature (deg F)
32
Time Check
current time
00:00:00.000 PM
MM/DD/YYYY
Acquire Data
Waveform Graph
Voltage
Amplitude
Time
Stop
Event-driven programming concept demo.vi Block Diagram
[1] "Time Check": Value Change
Type
Time Check
current time

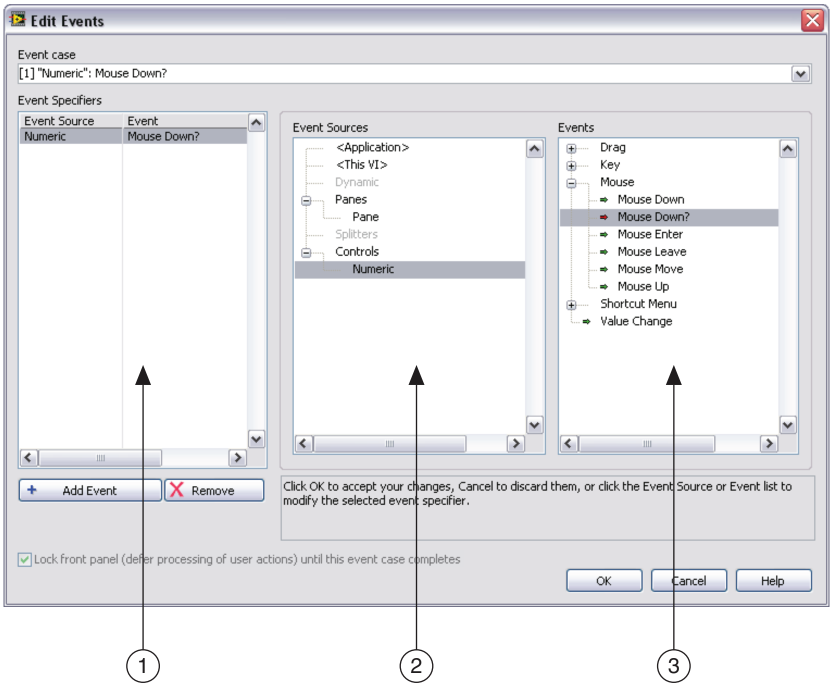
Configuring the Event Structure

You can select which events the Event structure implements by right-clicking the border and selecting **Edit Events Handled by This Case** from the shortcut menu





Edit Events Dialog Box



Event case: [1] "Numeric": Mouse Down?

Event Specifiers

Event Source	Event
Numeric	Mouse Down?

Click OK to accept your changes, Cancel to discard them, or click the Event Source or Event list to modify the selected event specifier.

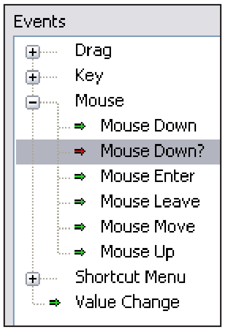
☑ Lock front panel (defer processing of user actions) until this event case completes

1	Configured events	2	Event sources	3	Events

Notify and Filter Events

LabVIEW categorizes user interface events into two different types of events:

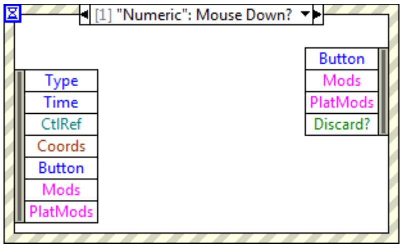
- Notify—(Green arrow) Notify events inform you that a user action occurred. LabVIEW has already performed the default action associated with that event.
- Filter—(Red arrow) Filter events allow you to validate or change the event data before LabVIEW performs the default action associated with that event. You also can discard the event entirely to prevent the change from affecting the VI.

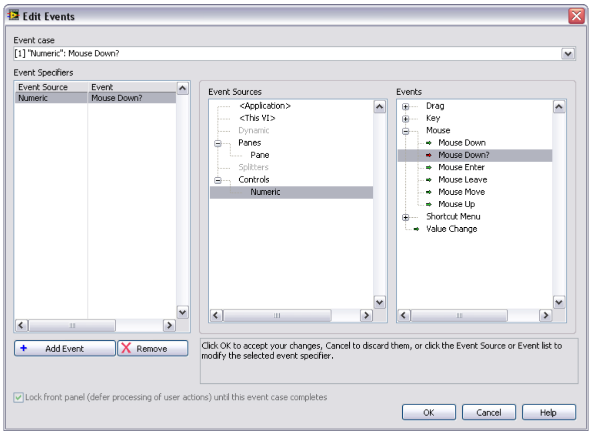


Note A single case in the Event structure cannot handle both notify and filter events. A case can handle multiple notify events but can handle multiple filter events only if the event data items are identical for all events.

Demonstration: Configure and Use Events

Configure and use an Event structure to create a VI that responds to user interface events using event-driven programming.





Event case: [1] "Numeric": Mouse Down?

Event Specifiers

Event Source	Event
Numeric	Mouse Down?

Click OK to accept your changes, Cancel to discard them, or click the Event Source or Event list to modify the selected event specifier.

☑ Lock front panel (defer processing of user actions) until this event case completes

Exercise 6-2 Converting a Polling Design to an Event Structure Design

Goal

To convert a polling-based application to an event-based application and compare the different in performance.

Description

First you observe the behavior of a polling VI.

Next, you modify the polling VI to create a more efficient, event-driven VI and observe the changes in behavior.

Finally, you add different types of events to the VI.

Table 6-3 lists the events you will implement in the UI Event Handler VI you create.

Table 6-3. User Interface Events

Event	Event Description
"Stop": Value Change	Stops the While Loop.
"Time Check": Value Change	Displays a time stamp when you click the **Time Check** button.
"Pane": Mouse Down	Displays the coordinates of the front panel point you click.
Panel Close?	Handles the event in which the user tries to close the running VI by clicking the window close button.
"Stop": Mouse Enter	Produces a beep when the mouse cursor moves over the **Stop** button.

Observing the Polling VI Behavior

1. Open and run Polling.vi.

   □ Open the `Events.lvproj` file in the `<Exercises>\LabVIEW Core 1\Events` directory and open the Polling VI from the project.

2. Examine the performance of a polling VI using the Windows Task Manager.

   □ Press the <Ctrl-Alt-Delete> keys and select **Start Task Manager** from the menu.

   □ Click the **Performance** tab in the **Windows Task Manager** window.

   □ Run the VI.

- □ Notice how high the CPU usage is.
- □ Stop the VI and notice how the CPU usage drops.

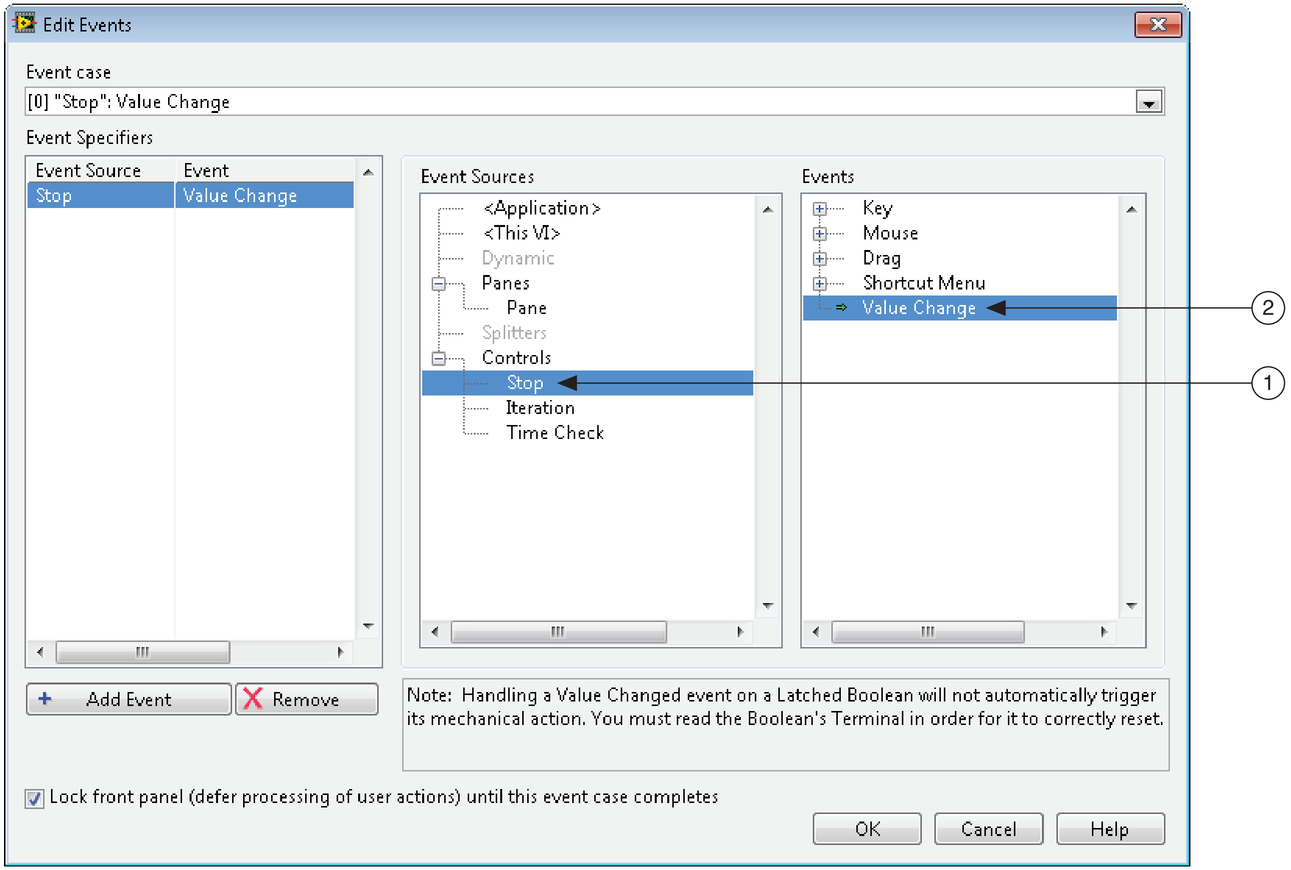
3. Open the block diagram, turn on execution highlighting, and run the VI again.
4. Notice how often the **Time Check** terminal sends data to the Case structure and how often the While Loop iterates.
5. Stop the VI and turn off execution highlighting.

Modifying the Polling VI to Use Events Instead of Polling

1. Save Polling VI as `UI Event Handler.vi` so you can modify it.
   - □ Select **Open additional copy** and add the copy to the project.
2. Close Polling.vi.
3. Open the block diagram of UI Event Handler.vi and move the **Stop** terminal and the **Time Check** terminal outside the While Loop. You move these terminals into the appropriate event cases later in this exercise.
4. Delete the Case structure and clean up any broken wires.
5. Place an Event structure inside the While Loop between the iteration terminal and the conditional terminal.
6. Right-click the Event structure and select **Edit Events Handled by This Case** from the shortcut menu.

7. Configure the event as shown in Figure 6-8.

Figure 6-8. Configuring the "Stop": Value Change Event



Edit Events

Event case

[0] "Stop": Value Change

Event Specifiers

Event Source	Event
Stop	Value Change

Note: Handling a Value Changed event on a Latched Boolean will not automatically trigger its mechanical action. You must read the Boolean's Terminal in order for it to correctly reset.

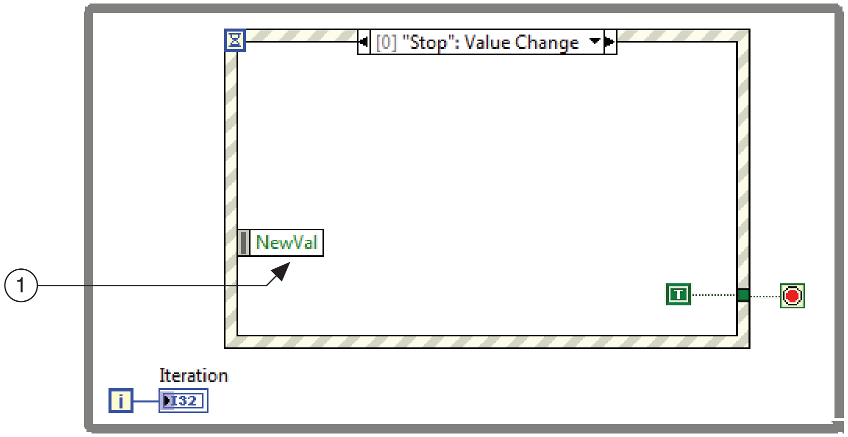
☑ Lock front panel (defer processing of user actions) until this event case completes

1 Click **Stop** in the **Event Sources** panel.
2 Click **Value Change** in the **Events** panel.

8. Click **OK** to close the dialog box.

9. Place a True constant inside the new **"Stop": Value Change** event and wire it to the conditional terminal of the While Loop as shown in Figure 6-9.

Figure 6-9. Event Structure with "Stop": Value Change Event



1 NewVal event data—Resize the event data items list so that only one item displays. Click the item and select **NewVal**.

Observing the Behavior of the Event-Driven VI

1. Run the VI.

2. Notice that the **Iteration** indicator does not increment.

3. Switch to the block diagram and enable execution highlighting.

4. Notice that the While Loop is executing the first iteration. The Event structure is waiting for an event.

5. Disable execution highlighting and switch back to the front panel.

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

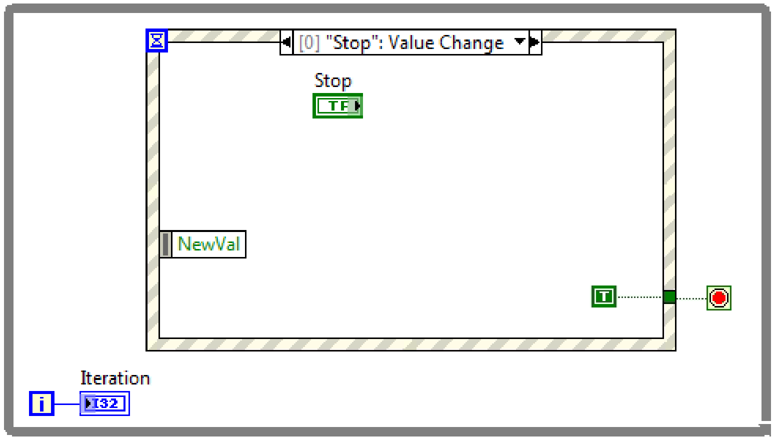
7. Notice that the VI stops running even though the **Stop** button is disconnected.

8. Notice that the **Stop** button stays depressed even though the mechanical action is set to **Latch When Released**. The reason the button stays depressed is because the VI stopped running after you clicked the button.

9. Reset the **Stop** button by clicking it again.

10. Drag the terminal of the **Stop** button into the "Stop": Value Change event as shown in Figure 6-10.

Figure 6-10. "Stop": Value Change Event with Stop Button Terminal



11. Run the VI and click the **Stop** button again.

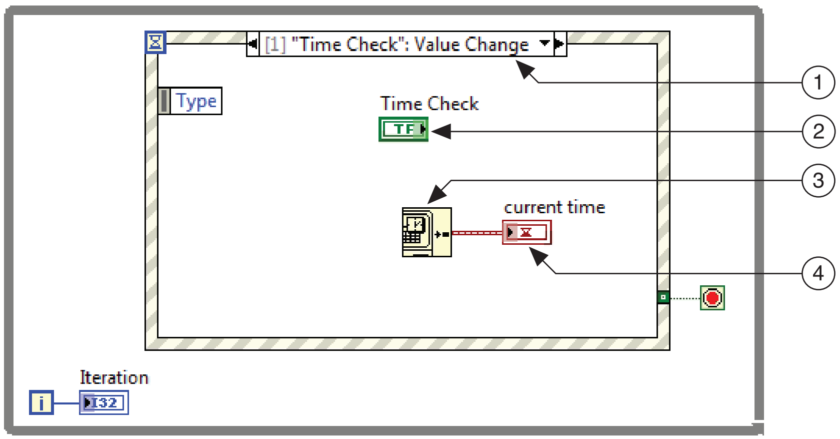
12. Notice this time the VI stops and the button resets.

Programming the "Time Check": Value Change Event

1. Add a new event case and create a “Time Check”: Value Change event as shown in Figure 6-11.

   □ Right-click the event structure and select **Add Event Case**.

Figure 6-11. Event Structure with "Time Check": Value Change Event



1 In the **Edit Events** window, select **Time Check** in the **Event Sources** panel and **Value Change** in the **Events** panel.
2 Move the Time Check terminal from outside the While Loop into the "Time Check": Value Change event case.
3 Get Date/Time In Seconds—Creates a time stamp in memory.
4 Indicator—Displays the **current time** output of the Get Date/Time In Seconds function.

2. Run the VI.

3. Click the **Time Check** button to see the current time display in the **current time** indicator.

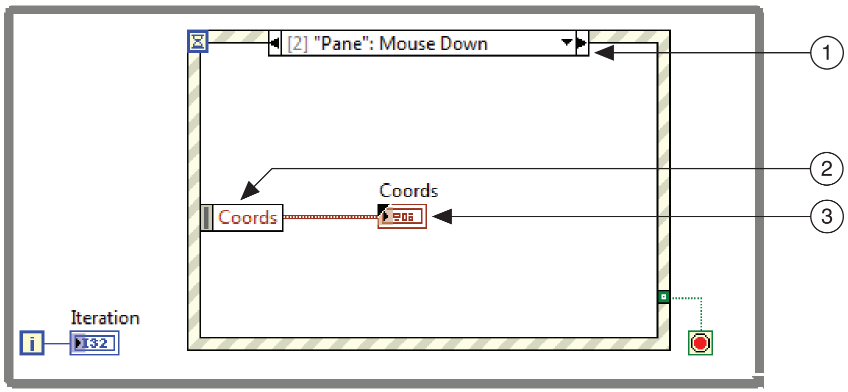
4. Display the Task Manager window and notice that CPU usage has decreased when you use events instead of polling.

5. Stop the VI.

Adding More Notify Events to the VI

1. Add a new event case and create a Mouse Down event as shown in Figure 6-12.

Figure 6-12. Event Structure with "Pane": Mouse Down Event



1 In the **Edit Events** window, select **Panes»Pane** in the **Event Sources** panel and **Mouse»Mouse Down** in the **Events** panel.
2 Coords event data—Click the event data node and select **Coords»All Elements**.
3 Coords indicator—Right-click the output of the **Coords** event data item and select **Create»Indicator** from the shortcut menu.

2. Run the VI.

3. Click on different parts of the front panel.

   □ Notice that the **Coords** indicator displays the coordinates for each point you click.

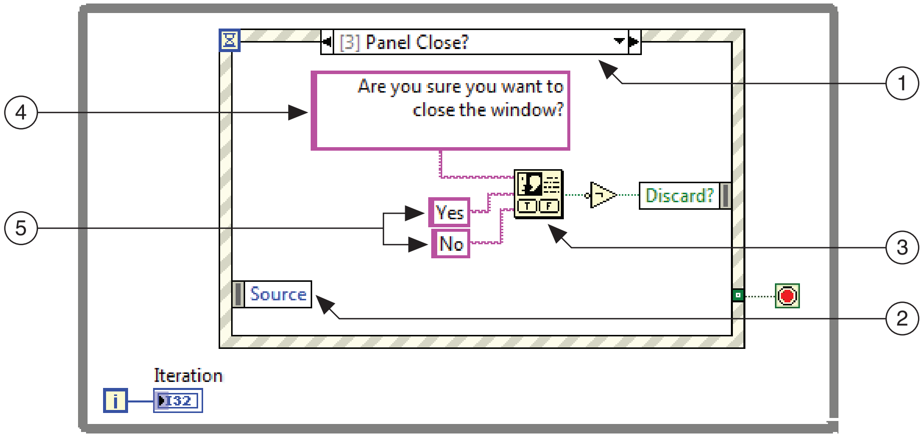
   □ Notice that the other events continue to behave as before.

4. Stop the VI.

Adding Filter Events to the VI

1. Add a new event case and create a Panel Close? event as shown in Figure 6-13.

Figure 6-13. Event Structure with Panel Close? Event



1 After you add the event, in the **Edit Events** window, select **<This VI>** in the **Event Sources** panel and **Panel Close?** in the **Events** panel.
2 Event data node—Click the Event Data Node and select **Source** from the menu.
3 Two Button Dialog function and Not function—Wire the **T button?** output to the Not function and wire the Not function to the Discard? event filter node.
4 String constant—Wire **Are you sure you want to close the window?** to the **message** input.
5 Yes and No string constants—Wire **Yes** to the **T button name ("OK")** input and wire **No** to the **F button name ("Cancel")** input.

2. Save and run the VI.

3. Click the "X" at the top-right of the window of the front panel.

4. Notice that clicking the **No** button cancels the event and returns to the VI.

5. Clicking the **Yes** button stops and closes the VI.

6. Stop the VI if necessary.

Challenge

1. If you have a sound card, add an event that produces a sound when the cursor is over the Stop button.

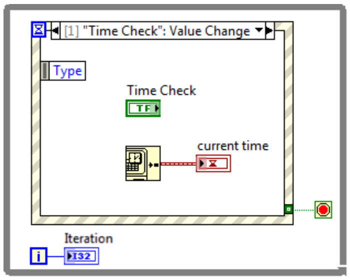
Tip Use Quick Drop to find the Beep VI.

End of Exercise 6-2

Caveats and Recommendations

The following list describes some of the caveats and recommendations to consider when incorporating events into LabVIEW applications.

- Place only one Event structure in a loop.
- Use a Value Change event to detect value changes.
- Keep event handling code short and quick.
- Place Boolean control terminals inside an event case for latched operations to work properly.
- Avoid using an Event structure outside of a loop.
- Avoid configuring two Event structures for the same event.



Think about the VIs that you will need to develop at your job.

Will you use event-based programming to implement any of your VIs? Why or why not?

Additional Resources

Learn More About	LabVIEW Help Topic or ni.com
Event-driven programming	*Locking Front Panels* *Choosing How the Event Structure Monitors For Events* *Viewing Enqueued Events at Run Time* *Event-Driven Programming* *Events in LabVIEW*
Deciding whether to use events	*Caveats and Recommendations when Using Events in LabVIEW*
Part of the Event structure	*Event Structure* *Configuring Events Handled by the Event Structure*
Determining which notifier to use	*Determining Which Type of User Interface Events to Use*

Activity 6-2: Lesson Review

1. Which of the following can NOT be used as the case selector input to a Case structure?

   a. Error cluster

   b. Array

   c. Enum

   d. String

2. How many events can an Event structure handle each time it executes?

   a. As many as have occurred since the last time the event structure executed

   b. One per configured event case

   c. One

3. Which statements about event-driven programming versus polling are true?

   a. Events execute on demand.

   b. Event-driven programming is less CPU-intensive.

   c. Event structures handle all events in the order the occur.

   d. Polling may fail to detect a change.

Activity 6-2: Lesson Review - Answers

1. Which of the following can NOT be used as the case selector input to a Case structure?
   a. Error cluster
   b. Array
   c. Enum
   d. String

2. How many events can an Event structure handle each time it executes?
   a. As many as have occurred since the last time the event structure executed
   b. One per configured event case
   c. One

3. Which statements about event-driven programming versus polling are true?
   a. Events execute on demand.
   b. Event-driven programming is less CPU-intensive.
   c. Event structures handle all events in the order the occur.
   d. Polling may fail to detect a change.

7 Modularity

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

Topics

+ Understanding Modularity
+ Icon
+ Connector Pane
+ Documentation
+ Using SubVIs

Exercises

Exercise 7-1 Temperature Warnings VI—As SubVI

A. Understanding Modularity

Objective: Recognize the benefit of using modular code and identify sections of code that could be reused.

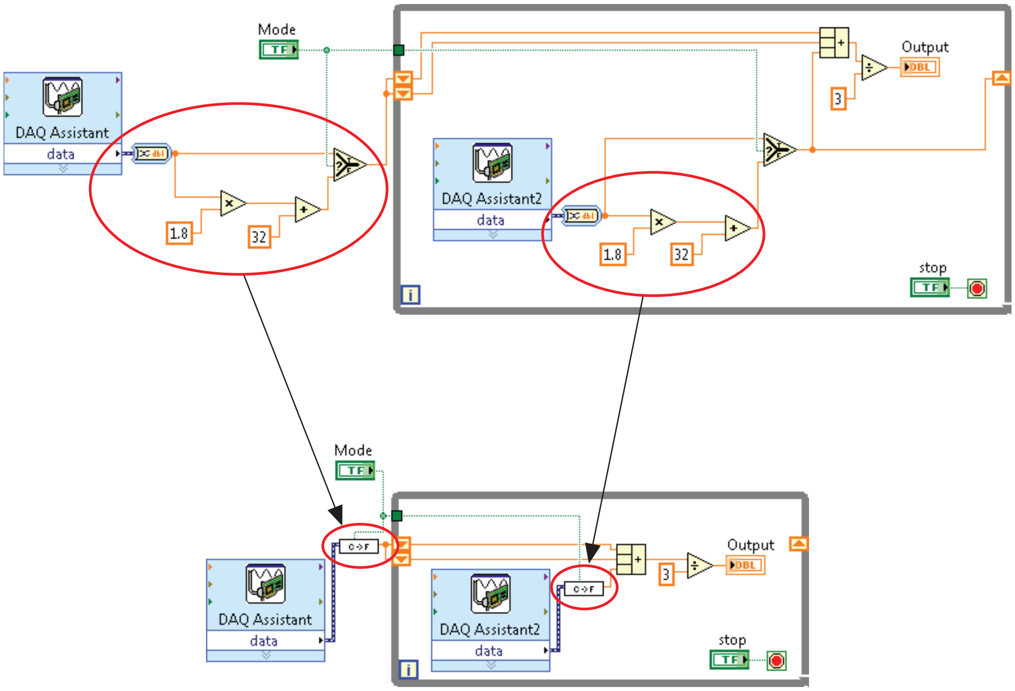
Modularity and SubVIs

Modularity The degree to which a program is composed of discrete modules such that a change tonne module has minimal impact on other modules. Modules in LabVIEW are called subVIs.

SubVI A VI used within another VI.The degree to which a program is composed of discrete modules such that a change tonne module has minimal impact on other modules. Modules in LabVIEW are called subVIs.

SubVIs—Reusing Code

SubVIs are similar to a subroutine in text-based programming languages. Use subVIs when you have code that performs identical operations on different parts of your block diagram or in another VI.

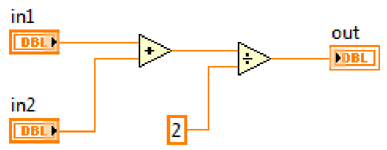
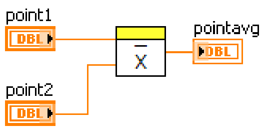


SubVIs

The following pseudo-code and block diagrams demonstrate the analogy between subVIs and subroutines.

Function Code	Calling Program Code

function average (in1, in2, out)
{
out = (in1 + in2)/2.0;
}
	main
	{
	average (point1, point2, pointavg)
	}

SubVI Block Diagram	Calling VI Block Diagram
	

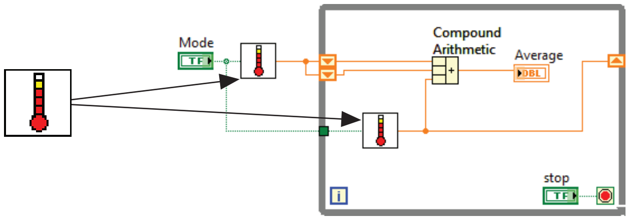
B. Icon

Objective: Recognize characteristics of a good icon and use the LabVIEW Icon Editor to create a custom icon.

Icon A VI icon is a graphical representation of a VI. It can contain text, images, or a combination of both. If you use a VI as a subVI, the icon identifies the subVI on the block diagram of the VI.

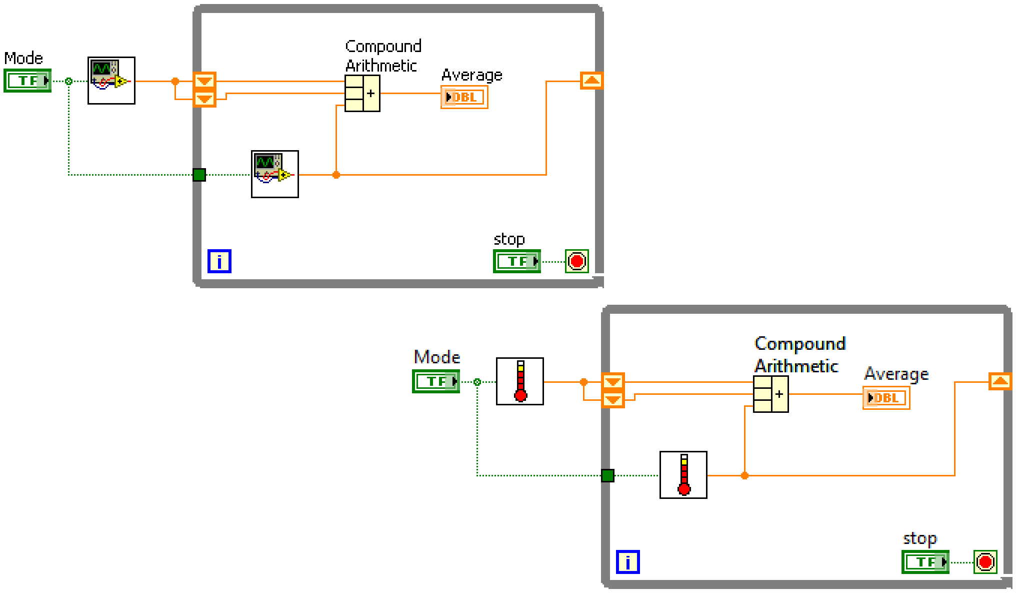
Purpose of Icon

This icon displayed in the upper right corner of the front panel is the same as the icon that appears when you place the VI on the block diagram.



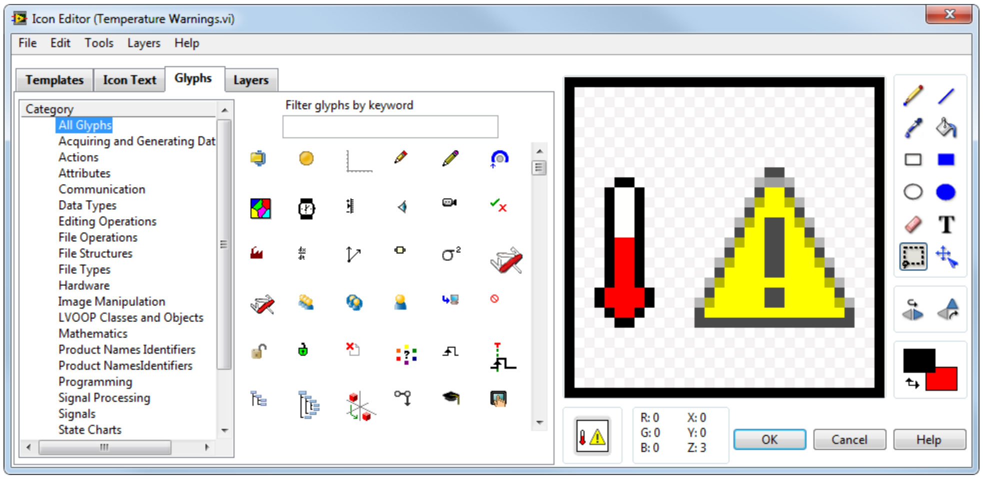
Characteristics of a Good Icon

A good icon conveys the functionality of the VI by using relevant graphics and text. You can also uses banners to identify related VIs.



Creating Icons—Icon Editor

Use the Icon Editor dialog box to edit a VI icon. You can customize your icon with text, graphics from the glyphs library, and banners. Additionally, you can create and save custom templates.



Demonstration: Creating an Icon

Using the LabVIEW Icon Editor, the instructor will demonstrate how to use the templates, icon text, glyphs, and layers to create an icon.

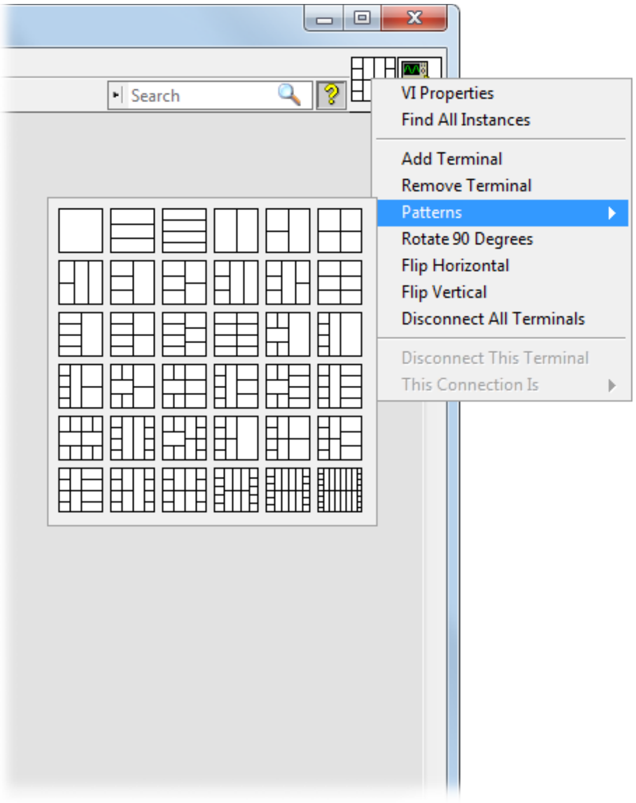
C. Connector Pane

Objective: Select and configure a connector pane for a subVI.

Connector Pane A set of terminals that correspond to the controls and indicators of that VI, similar to the parameter list of a function call in text-based programming languages. The connector pane defines the inputs and outputs you can wire to the VI so you can use it as a subVI.

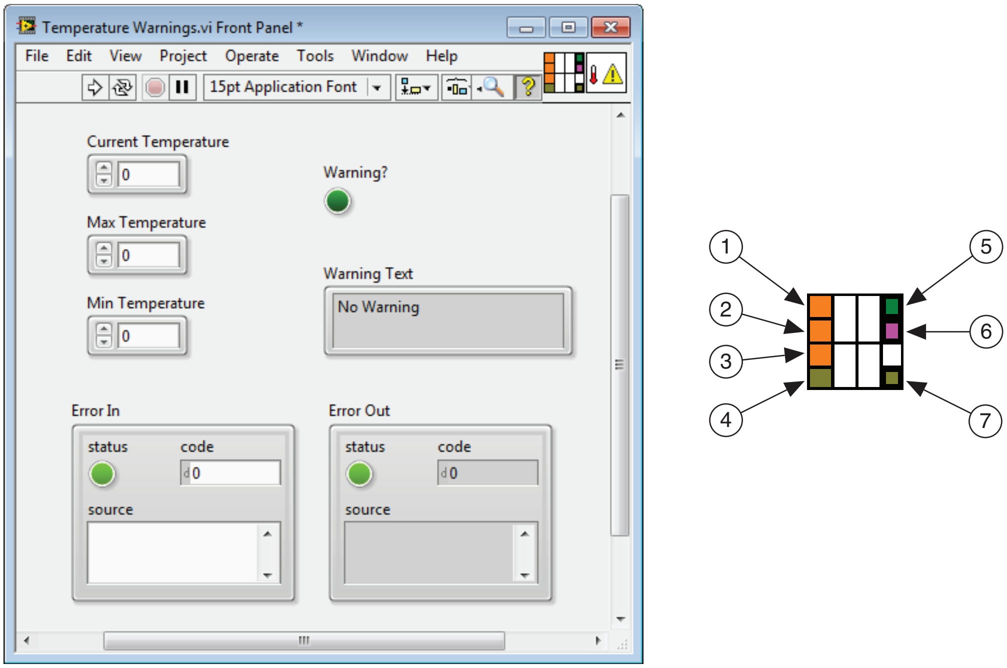
Patterns

A connector pane receives data at its input terminals and passes the data to the block diagram code through the front panel controls and receives the results at its output terminals from the front panel indicators. The connector pane is displayed next to the icon on the front panel window. You can select from many different patterns depending on how many inputs and outputs are required.



Assigning Terminals

Click the connector pane terminal and then click the front panel control or indicator to assign the terminal.


Temperature Warnings.vi Front Panel *
File Edit View Project Operate Tools Window Help
15pt Application Font
1
2
3
4
5
6
7

Current Temperature 0

Max Temperature 0

Min Temperature 0

Warning?

Warning Text: No Warning

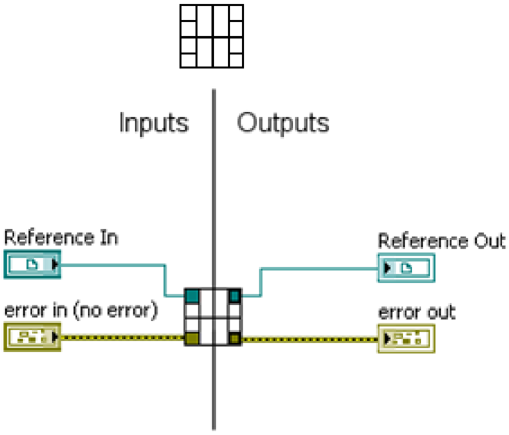
Error In: status, code 0, source

Error Out: status, code 0, source

1	Current Temperature	4	Error In	6	Warning Text
2	Max Temperature	5	Warning?	7	Error Out
3	Min Temperature				

Standards

The standard connector pane, shown in the following figure is 4 x 2 x 2 x 4.



Use the following guidelines when assigning connector pane terminals:

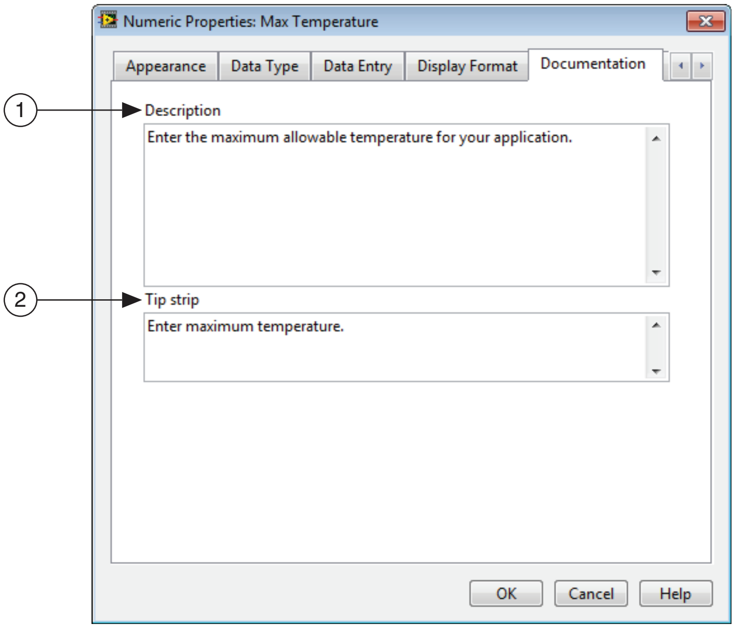
- References on the top
- Error clusters on the bottom
- Inputs on the left
- Outputs on the right
- Unused terminals can be assigned later if you modify the VI

D. Documentation

Objective: Explain how to document code in LabVIEW using descriptions and tip strips, and describe four methods for documenting code on the block diagram.

Creating Descriptions and Tip Strips

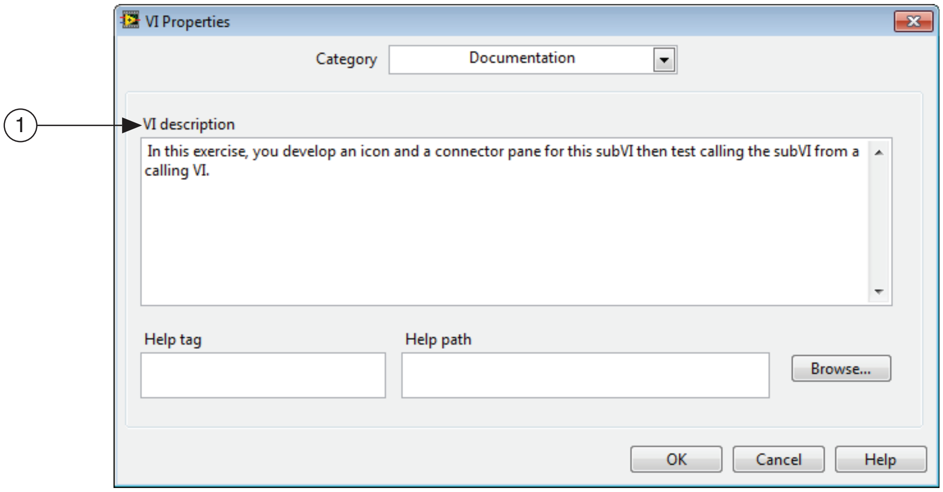
Create descriptions and tip strips for front panel objects in the properties dialog box for the object. Create descriptions for VIs on the Documentation page of the in the VI Properties dialog box.



Description: Enter the maximum allowable temperature for your application.

Tip strip: Enter maximum temperature.

1 Descriptions appear in the Context Help window.
2 Tip strips display when the cursor is over an object while the VI runs



Category: Documentation

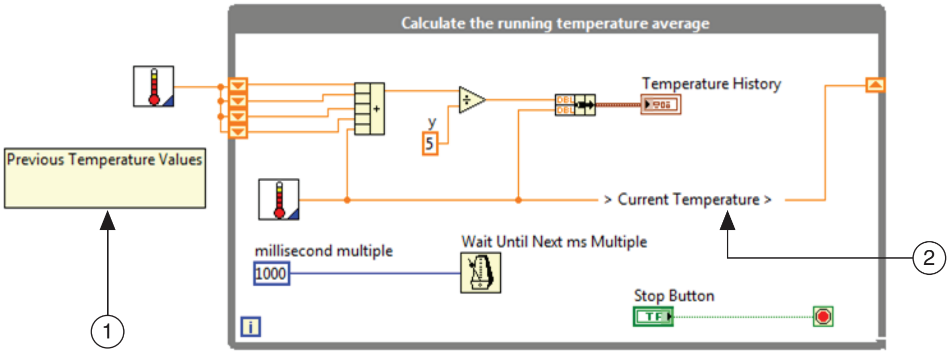
VI description: In this exercise, you develop an icon and a connector pane for this subVI then test calling the subVI from a calling VI.

Help tag

Help path

1 Descriptions appear in the Context Help window.

Documenting Block Diagram Code


Calculate the running temperature average
Previous Temperature Values
Temperature History
y
5
> Current Temperature >
millisecond multiple
1000
Wait Until Next ms Multiple
Stop Button
1
2

1 Free label
2 Owned label

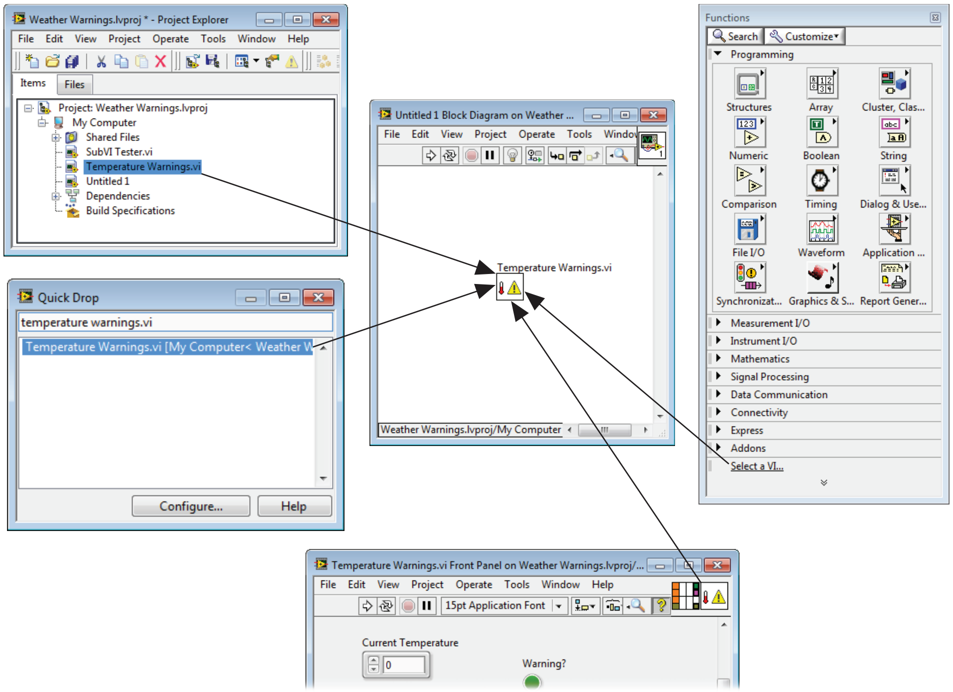
E. Using SubVIs

Objective: Demonstrate how to place subVIs on the block diagram, explain terminal settings and error handling, and create subVIs from a section of existing code.

Placing SubVIs on the Block Diagram

Place a SubVI on the block diagram in one of the following ways:

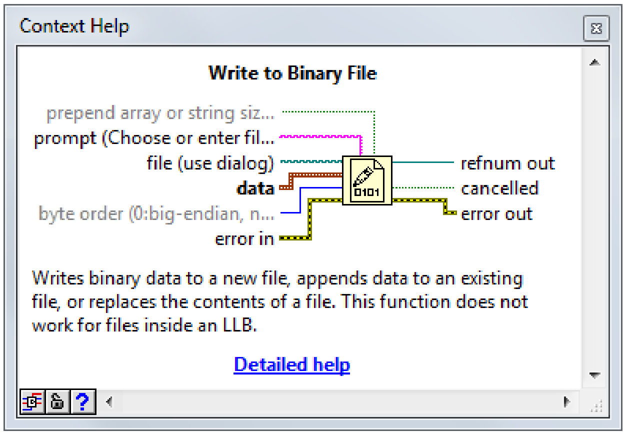
- Click Select a VI on the Functions palette.
- Drag the VI from the Project Explorer Window.
- Drag the Icon from an open VI.
- Use Quick Drop to search for the VI by name.



Terminal Settings

You can designate which inputs and outputs are required, recommended, and optional to prevent users from forgetting to wire subVI terminals. LabVIEW sets inputs and outputs of VIs you create to Recommended by default. Set a terminal to required only if the VI must have the input to run properly.

In the Context Help window, terminal labels appear differently depending on their setting.



Context Help

Write to Binary File

prepend array or string siz...
prompt (Choose or enter fil...
file (use dialog)
data
byte order (0:big-endian, n...
error in
refnum out
cancelled
error out

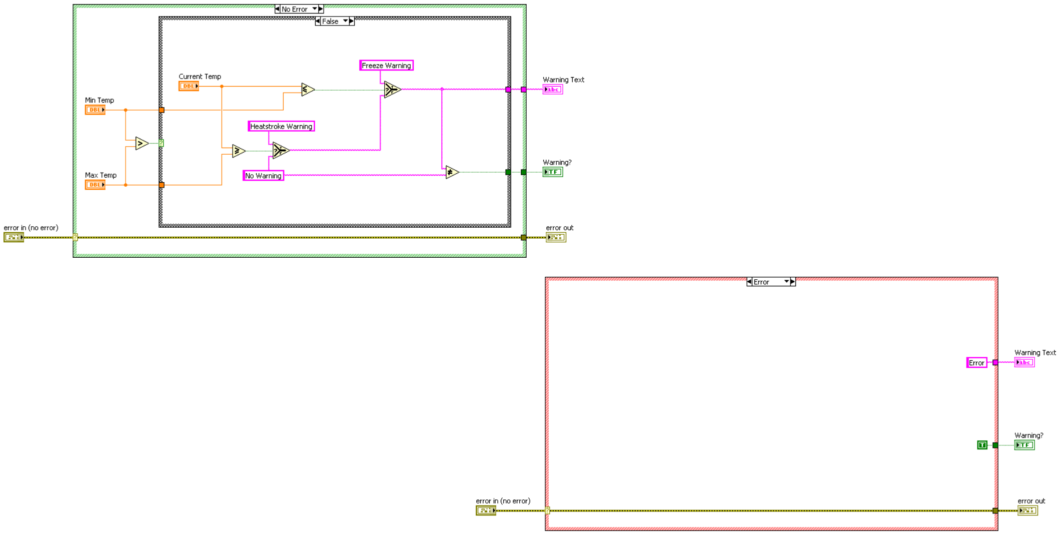
Writes binary data to a new file, appends data to an existing file, or replaces the contents of a file. This function does not work for files inside an LLB.

Detailed help

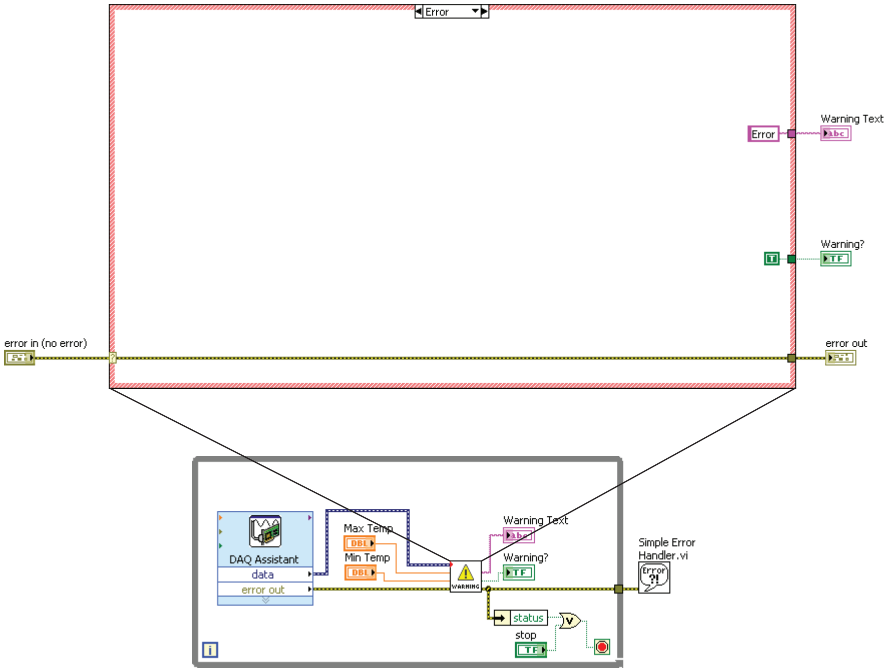
Appearance	Meaning	Description
Bold	Required	The block diagram containing the subVI will be broken if you do not wire the required inputs.
Plain	Recommended	The block diagram containing the subVI can execute if you do not wire the recommended or optional terminals. If you do not wire the terminals, the VI does not generate any warnings.
Dimmed	Optional	

Handling Errors

Use a Case structure to handle errors passed into the subVI.



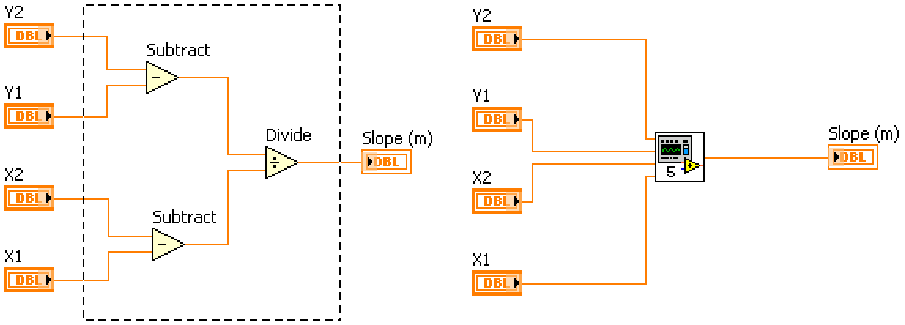
Avoid using LabVIEW error handler VIs inside subVIs.



Convert a Section of a VI to a SubVI

Simplify your block diagram by converting sections of the block diagram into subVIs.

1. Select the section of the block diagram to convert.
2. Select **Edit»Create SubVI**.
3. Double-click the icon to open the subVI and edit the icon and connector pane.



Exercise 7-1 Temperature Warnings VI—As SubVI

Goal

Create the icon and connector pane for a VI so that you can use the VI as a subVI.

Scenario

You have created a VI that determines a warning string based on the inputs given. Create an icon and a connector pane so that you can use this VI as a subVI.

Design

The Temperature Warnings VI contains the following inputs and outputs:

Table 7-1. Temperature Warnings VI Inputs and Outputs

Inputs	Outputs
Weather Data In	Weather Data Out
Error In	Error Out

Use the standard connector pane terminal pattern to assure room for future expansion.

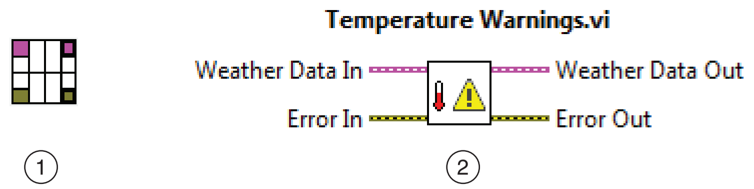
Implementation

1. Open `Weather Warnings.lvproj` in the `<Exercises>\LabVIEW Core 1\Weather Warnings` directory.

2. Open **Temperature Warnings VI** from the **Project Explorer** window.

3. Connect the inputs and outputs to the connector pane as shown in Figure 7-1.

Figure 7-1. Connector Pane Connections for Temperature Warnings VI



1 Connector Pane—Located in the upper right corner of the VI window, the connector pane displays potential terminals for the VI. The connector pane shown here displays the standard pattern of terminals. You can right-click the connector pane and select **Patterns** to choose different terminal designs.
2 Connections—The **Context Help** window displays the connections for the VI.

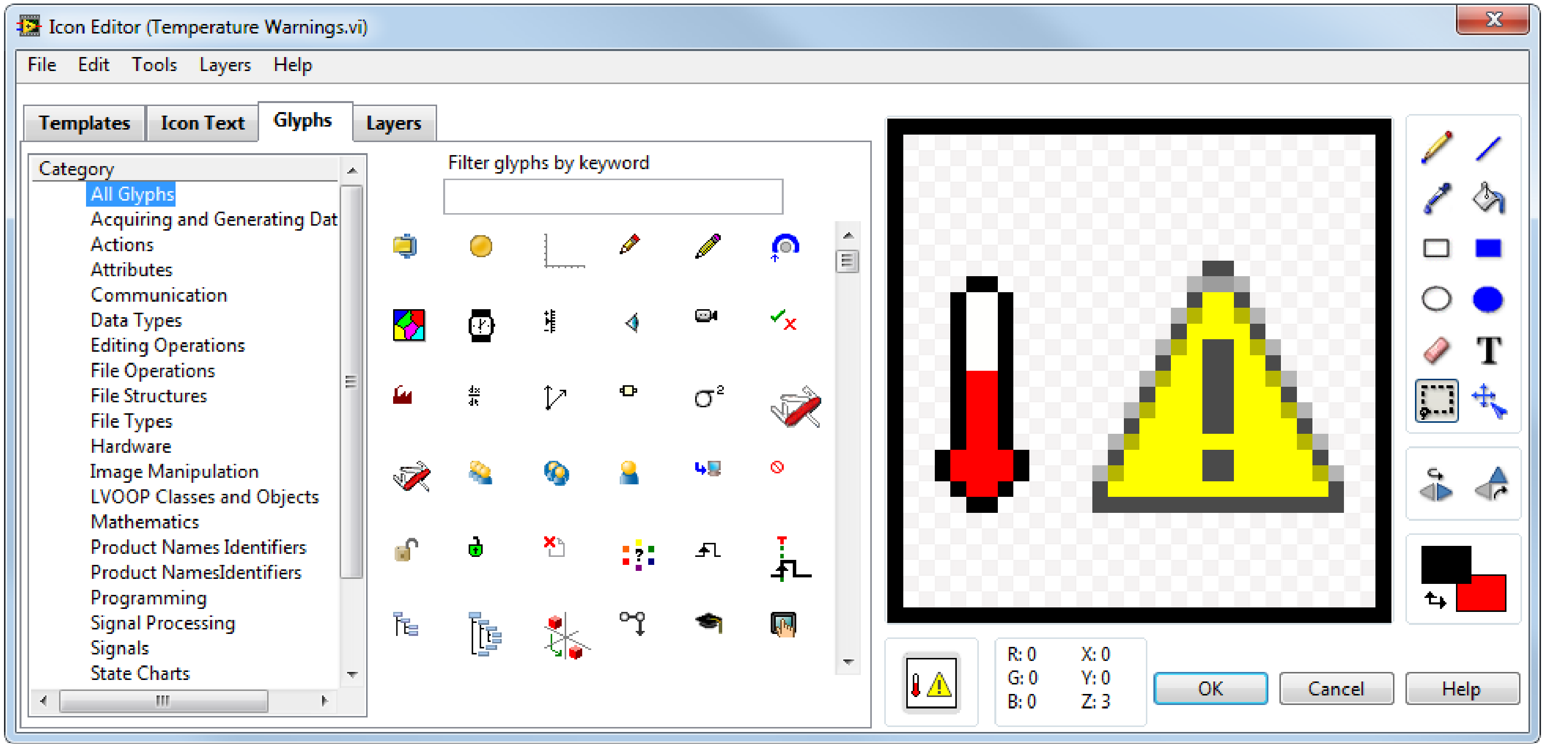
- □ Using the Wiring tool, click the upper-left terminal of the connector pane.
- □ Click the corresponding front panel control, **Weather Data In**.
- □ Notice that the connector pane terminal fills in with a color to match the data type of the control connected to it.
- □ Click the bottom-left terminal of the connector pane.
- □ Click the corresponding front panel control, **Error In**.
- □ Continue wiring the connector pane until all controls and indicators are wired, and the **Context Help** window matches that shown in Figure 7-1.

4. Create an icon.

- □ Right-click the icon and select **Edit Icon**.

□ Use the tools in the **Icon Editor** dialog box to create an icon. Make the icon as simple or as complex as you want, however, it should be representative of the function of the VI. Figure 7-2 shows a simple example of an icon for this VI.

Figure 7-2. Sample Warning Icon



Tip Double-click the Selection tool to select the existing graphic. Press the <Delete> key to delete the graphic. Then, double-click the rectangle tool to automatically create a border for the icon.

Tip Double-click the Text tool to modify fonts. You can select **Small Fonts** to choose fonts smaller than 9 points in size.

Tip Select the **Glyphs** tab and filter the glyphs by the keyword `temperature`, then drag a thermometer glyph onto your icon. And then filter by the keyword `warning` and drag a warning glyph onto your icon.

5. Click **OK** when you are finished to close the **Icon Editor** dialog box.

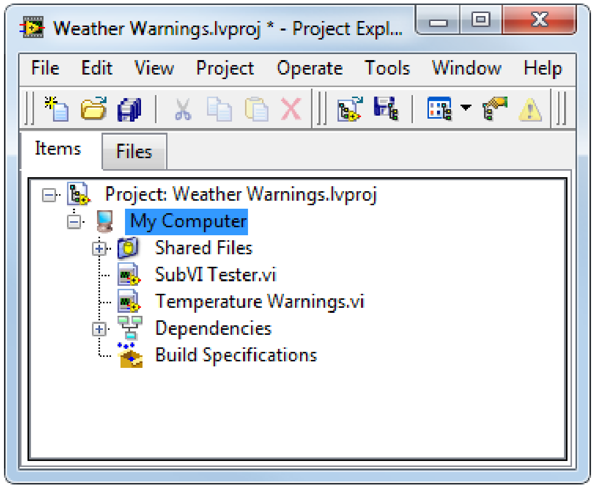
6. Save and close the VI.

Test

Test the Temperature Warnings VI as a SubVI.

1. Add files to the Weather Warnings LabVIEW project as shown in Figure 7-3.

Figure 7-3. Weather Warnings Project


Weather Warnings.lvproj * - Project Expl...
File Edit View Project Operate Tools Window Help
Items
Files

Project: Weather Warnings.lvproj
My Computer
Shared Files
SubVI Tester.vi
Temperature Warnings.vi
Dependencies
Build Specifications

□ Add an auto-populating folder to the Weather Warnings LabVIEW project. LabVIEW continuously monitors auto-populating folders and updates the folder in the **Project Explorer** window according to changes made in the project and on disk.

- Right-click **My Computer** in the Weather Warnings project and select **Add»Folder (Auto-populating)** from the shortcut menu.
- Navigate to `<Exercises>\LabVIEW Core 1\Shared Files` and click the **Select Folder** button.

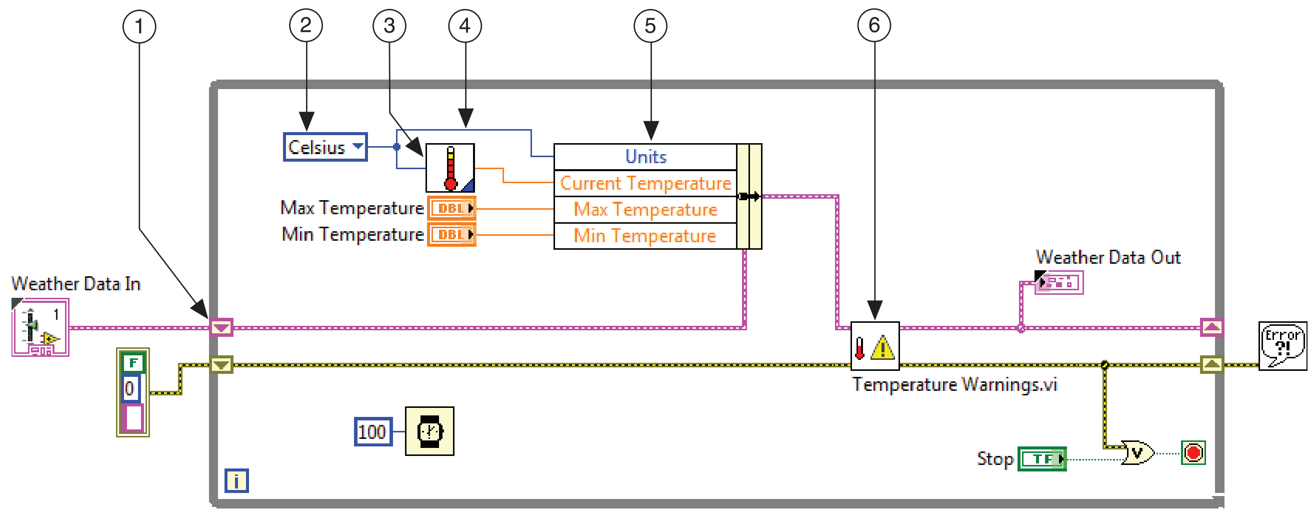
  The **Shared Files** folder contains shared files that you use in this and future exercises.

□ Add `SubVI Tester.vi` to the project.

- Right-click **My Computer** and select **Add»File** from the shortcut menu.
- Navigate to `<Exercises>\LabVIEW Core 1\Weather Warnings\Test VIs\SubVI Tester.vi` and click **Add File**.

2. Open the SubVI Tester VI and complete the block diagram as shown in Figure 7-4.

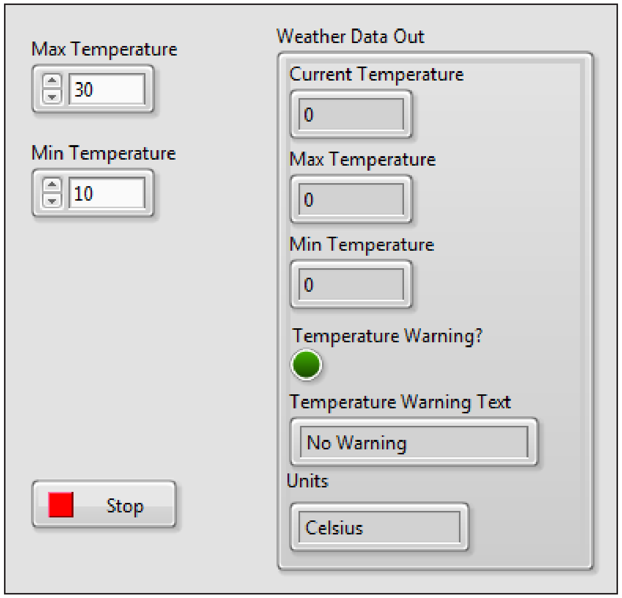
Figure 7-4. Test SubVI Block Diagram



1 Initialize shift register—Right-click the left shift register and select **Create»Constant** to initialize the shift register. Right-click the cluster and select **View Cluster as Icon**.
2 **Enum Constant**—Right-click the **Units (0:Celsius)** input of the Thermometer (Demo) VI and select **Create»Constant**. Creating the enum constant from the Thermometer (Demo) VI automatically populates the enum with the appropriate choices. Use the Operating tool to select **Celsius**.
3 **Thermometer (Demo)**—Locate this VI in the **Shared Files** folder in the **Project Explorer** window, drag it to the block diagram and wire it as shown. This VI generates sample temperature values.
4 Wire the **Units (0:Celsius)** constant to the **Units** element of the Bundle By Name function.
5 **Bundle By Name**—Expand the node to display four elements. Use the Operating tool to select **Units**.
6 **Temperature Warnings**—Because of the modifications you made to Temperature Warnings VI, you can use it as a subVI. Wire the Temperature Warnings VI using the connections you just created.

3. Arrange the front panel as shown in Figure 7-5.

Figure 7-5. SubVI Tester Front Panel



Max Temperature: 30

Min Temperature: 10

Weather Data Out

Current Temperature: 0

Max Temperature: 0

Min Temperature: 0

Temperature Warning? ●

Temperature Warning Text: No Warning

Units: Celsius

4. On the front panel of the SubVI Tester VI, enter test values for the **Max Temperature** and **Min Temperature** controls.

5. Run the VI.

   The Thermometer (Demo) VI generates sample temperatures, which the SubVI Tester VI displays on the **Current Temperature** indicator.

6. Notice how **Temperature Warning Text** indicator changes as the temperature rises and falls.

7. After you have finished testing, save and close the VI.

End of Exercise 7-1

Additional Resources

Learn More About	LabVIEW Help Topic or ni.com
SubVIs	*Creating SubVIs* *Customizing VIs* *Placing SubVIs on Block Diagrams*
Icons	*Using Icons* *Creating Icons*
Documentation	*Creating Object Descriptions and Tip Strips* *Creating Documentation* *Creating and Editing VI Descriptions*

Activity 7-1: Lesson Review

1. On a subVI, which terminal setting causes a broken VI if the terminal is not wired?
   a. Required
   b. Recommended
   c. Optional

2. You must create a custom icon to use a VI as a subVI.
   a. True
   b. False

Activity 7-1: Lesson Review - Answers

1. On a subVI, which terminal setting causes a broken VI if the terminal is not wired?
   - **a. Required**
   - b. Recommended
   - c. Optional

2. You must create a custom icon to use a VI as a subVI.
   - a. True
   - **b. False**

     You do not need to create a custom icon to use a VI as a subVI, but it is highly recommended to increase the readability of your code.

8 Acquiring Measurements from Hardware

In this lesson you will learn about the differences between NI DAQ systems and instrument control and how LabVIEW connects to hardware to get real-world measurements.

Topics

Exercises

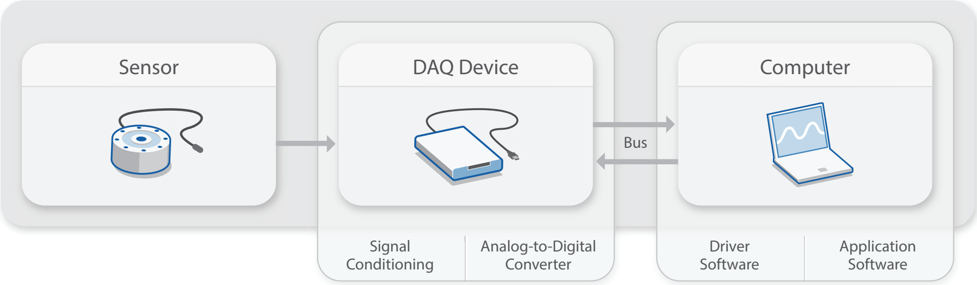
A. Measurement Fundamentals with NI DAQ Hardware

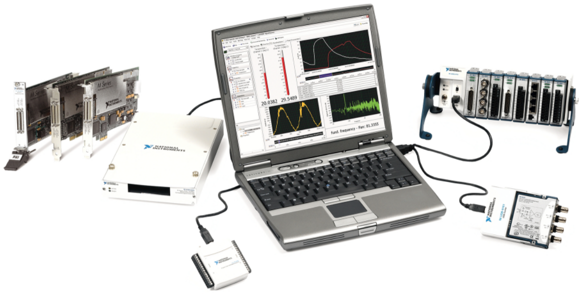
Objective: Recognize the components of a DAQ system and practice connecting to hardware.

DAQ—Measuring Physical Phenomenon with a Computer

By using NI DAQ systems and LabVIEW, you can customize all aspects of data collection.

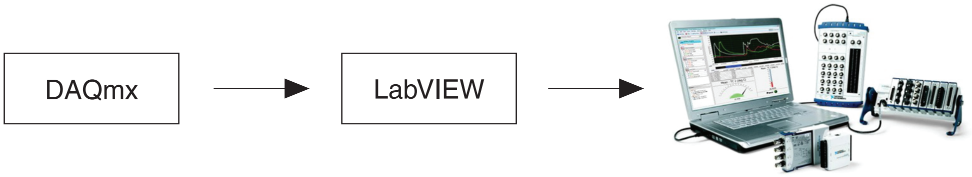
Data acquisition The process of measuring an electrical or physical phenomenon with a computer. A DAQ system consist of a sensor, NI DAQ device, and a computer running LabVIEW.





Multimedia: Measurement Fundamentals with NI DAQ

Complete the multimedia module, *Measurement Fundamentals with NI DAQ*, available in the `<Exercises>\LabVIEW Core 1\Multimedia\` folder to learn how to acquire different types of measurements using LabVIEW and an NI DAQ device.



DAQmx → LabVIEW →

The following table lists common sensors and the phenomenon they measure.

Sensor	Phenomenon
Thermocouple, RTD, thermistor	Temperature
Photo sensor	Light
Microphone	Sound
Strain gage, piezoelectric transducer	Force and Pressure
Potentiometer, LVDT, optical encoder	Position and Displacement
Accelerometer	Acceleration
pH probe	pH

You can group real-world signals as analog, digital or counters.

Signal Type	Description	Examples
Analog	Signals that vary continuously	Temperature, current
Digital	Electrical signals that transfer binary data, such as on/off or true/false.	LEDs, switches
Counter	Digital timing signal that includes characteristics of a digital signal that can be counted or measured, such as rising edges, frequency of edges occurring, and pulse-width.	Event counting, period measurement, position measurement

Exercise 8-1 Using NI MAX to Examine a DAQ Device

Goal

Use MAX to examine, configure, and test a device.

Implementation

Complete the following steps to examine the configuration for the DAQ device in the computer using MAX. Use the test routines in MAX to confirm operation of the device. If you do not have a DAQ device, you can simulate a device using the instructions in step 3.

Note Portions of this exercise can only be completed with the use of a real device and a BNC-2120, shown in Figure 8-1. Some of these steps have alternative instructions for simulated devices.

1. Launch MAX by selecting **Start»Programs»NI MAX** or by double-clicking the NI MAX icon on your desktop. MAX searches the computer for installed National Instruments hardware and displays the information.

2. If you have a DAQ device installed, skip step 3 and go to the *Examining the DAQ Device Settings* section.

3. Create an NI-DAQmx simulated device to allow you to complete the exercises without hardware.

   □ Right-click **Devices and Interfaces** and select **Create New** from the shortcut menu.

   □ In the **Create New** dialog box, select **Simulated NI-DAQmx Device or Modular Instrument**.

   □ Click the **Finish** button.

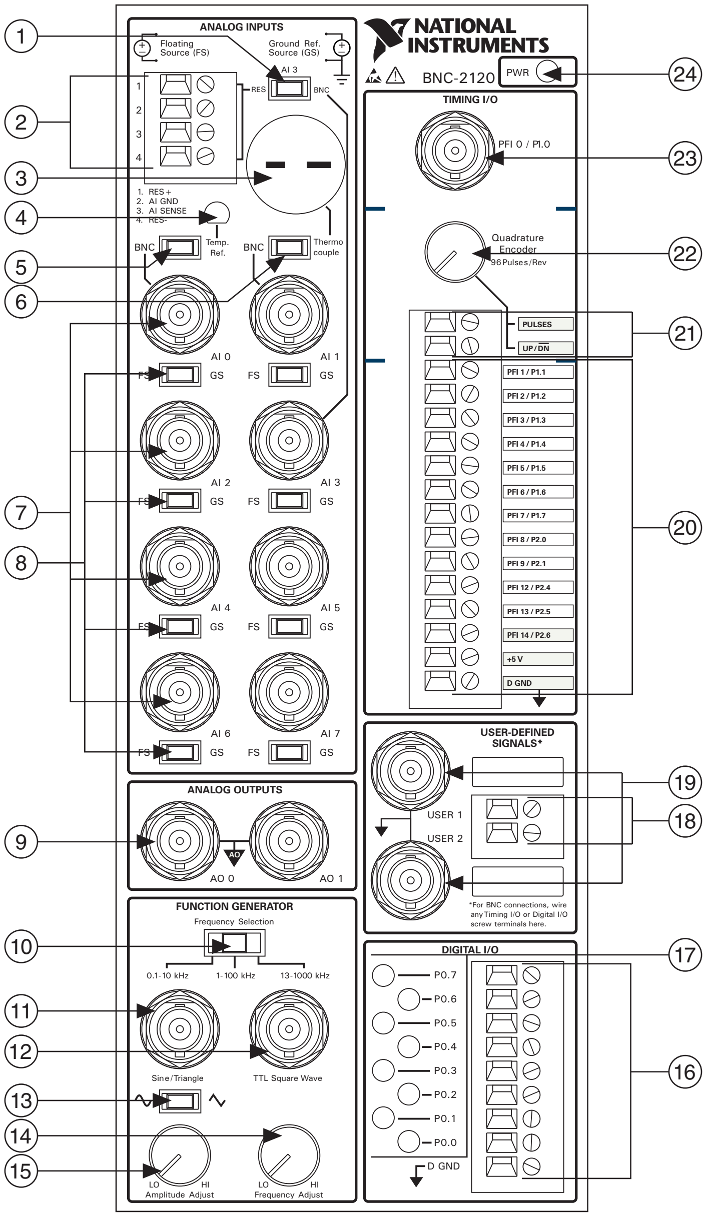
   □ In the **Create Simulated NI-DAQmx Device** dialog box, select **M Series DAQ» NI PCI 6225**.

   □ Click the **OK** button.

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.

Figure 8-1. BNC-2120



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

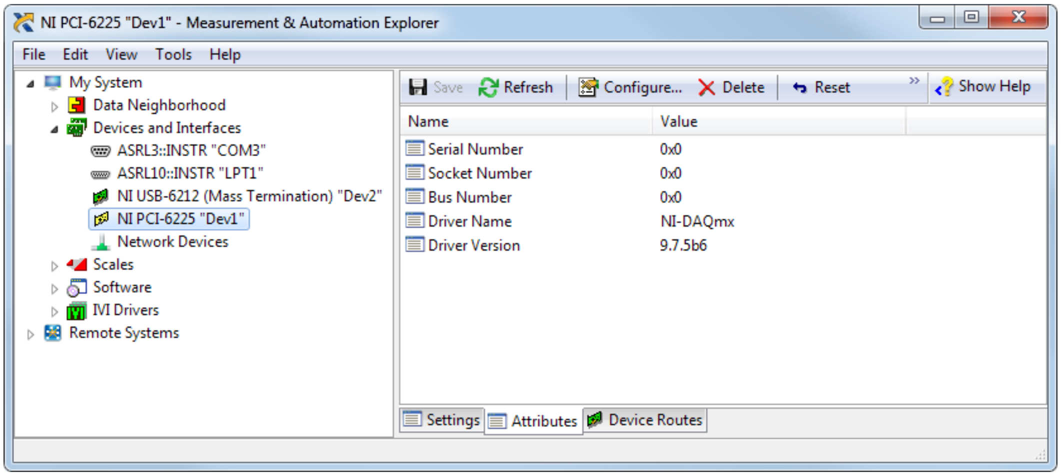
1. Expand the **Devices and Interfaces** section.

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. Figure 8-2 shows the simulated PCI-6225 device.

   Make sure the device you use is named `Dev 1`. To rename a device, right-click the device and select **Rename** from the shortcut menu.

Figure 8-2. MAX with Device and Interfaces Expanded



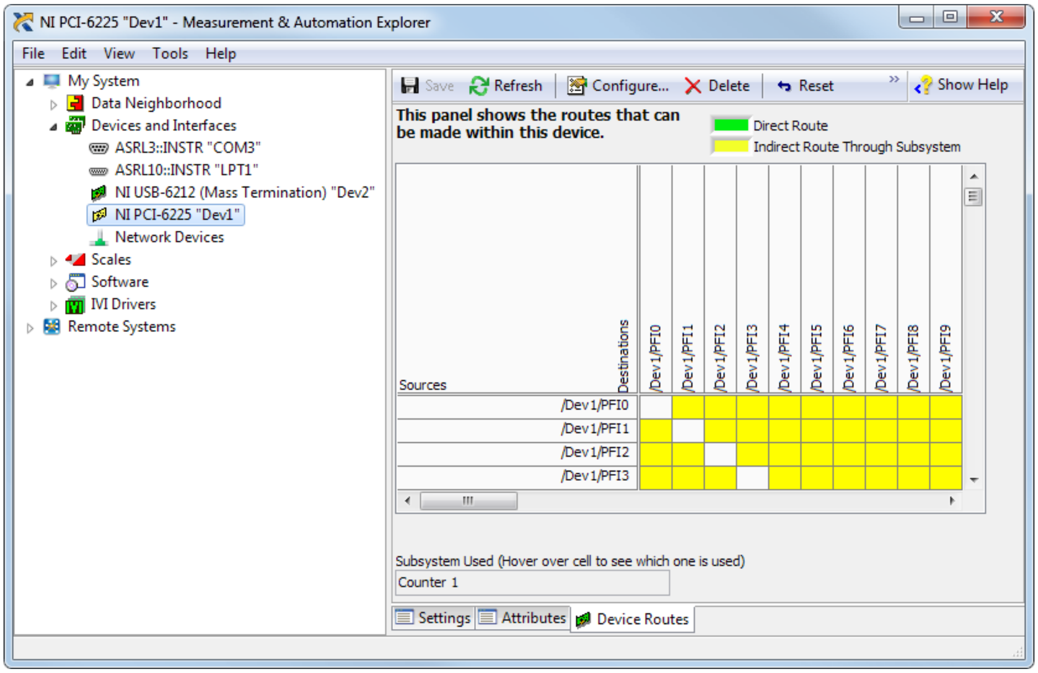
Name	Value
Serial Number	0x0
Socket Number	0x0
Bus Number	0x0
Driver Name	NI-DAQmx
Driver Version	9.7.5b6

Tip The **Show Help/Hide Help** button in the top right corner of MAX is available for certain items. Click the **Show Help/Hide Help** button to hide online help or show the DAQ device information.

3. Select the **Device Routes** tab at the bottom of MAX to see detailed information about the internal signals that can be routed to other destinations on the device, as shown in Figure 8-3. This is a powerful resource that gives you a visual representation of the signals

that are available to provide timing and synchronization with components that are on the device and other external devices.

Figure 8-3. Device Routes



This panel shows the routes that can be made within this device.

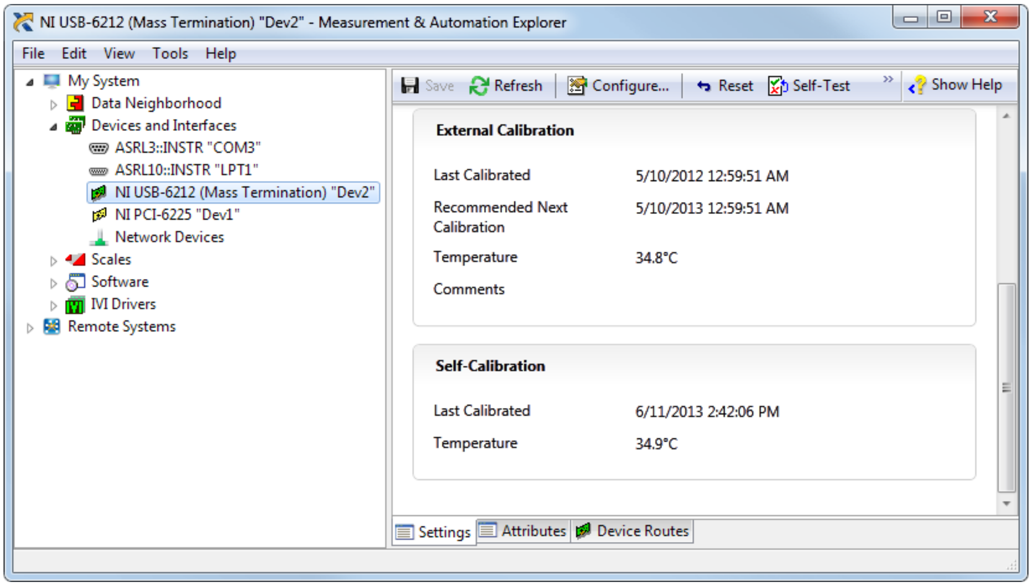
🟩 Direct Route
🟨 Indirect Route Through Subsystem

Sources \ Destinations	/Dev1/PFI0	/Dev1/PFI1	/Dev1/PFI2	/Dev1/PFI3	/Dev1/PFI4	/Dev1/PFI5	/Dev1/PFI6	/Dev1/PFI7	/Dev1/PFI8	/Dev1/PFI9
/Dev1/PFI0		🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
/Dev1/PFI1	🟨		🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
/Dev1/PFI2	🟨	🟨		🟨	🟨	🟨	🟨	🟨	🟨	🟨
/Dev1/PFI3	🟨	🟨	🟨		🟨	🟨	🟨	🟨	🟨	🟨

Subsystem Used (Hover over cell to see which one is used)
Counter 1

4. Select the **Settings** tab, as shown in Figure 8-4, to see information about the last time the device was calibrated both internally and externally. Not all devices contain calibration information.

Figure 8-4. Calibration



External Calibration

Last Calibrated	5/10/2012 12:59:51 AM
Recommended Next Calibration	5/10/2013 12:59:51 AM
Temperature	34.8°C
Comments	

Self-Calibration

Last Calibrated	6/11/2013 2:42:06 PM
Temperature	34.9°C

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.

Testing the DAQ Device Components

1. Click the **Self-Test** button in MAX to test the device. The device should pass the test because it is already configured.

2. Click the **Test Panels** button to test the individual functions of the DAQ device, such as analog input and output. The **Test Panels** dialog box appears.

   □ Use the **Analog Input** tab to test the various analog input channels on the DAQ device. Click the **Analog Input** tab. Click the **Start** button to acquire data from analog input channel 0 and click the **Stop** button when you finish.

   – If you are using the BNC-2120, make sure the switch over the **AI 0** connector is in the **Temp. Ref.** position to connect the temperature sensor to ai0. Place your finger on the sensor to see the voltage rise.

   – If you are using a simulated device, a sine wave is shown on all input channels.

   □ Use the **Analog Output** tab to set up a single voltage or sine wave on one of the DAQ device analog output channels. Click the **Analog Output** tab.

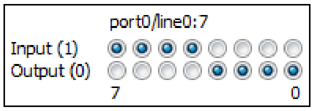
   – Select **Sinewave Generation** in the **Mode** drop-down menu and click the **Start** button. MAX generates a continuous sine wave on analog output channel 0.

   – If you have hardware installed, you can read the sine wave that channel 0 outputs. On the BNC-2120, wire **Analog Out Ch0** to **Analog In Ch1**. Click the **Analog Input** tab in the **Test Panels** dialog box and select **Dev1/ai1** from the **Channel Name** drop-down menu. Click the **Start** button to acquire data from analog input channel 1. MAX displays the sine wave from analog output channel 0.

   □ Use the **Digital I/O** tab to test the digital lines on the DAQ device. Click the **Digital I/O** tab.

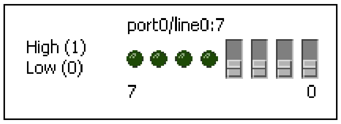
   – In the **Select Direction** section, set lines 0 through 3 as output as shown in Figure 8-5.

Figure 8-5. Digital I/O Line Direction



   – Click **Start** to begin the digital output test, then toggle the switches in the **Select State** section shown in Figure 8-6. If you have a BNC-2120, toggling the switches turns the LEDs on or off. Notice that the LEDs use negative logic.

Figure 8-6. Digital I/O Switches



   – Click **Stop** to stop the digital output test.

   □ Use the **Counter I/O** tab to determine if the DAQ device counter/timers are functioning properly. Click the **Counter I/O** tab.

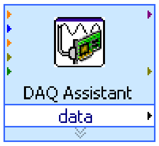
- If you have hardware installed, you can verify counter/timer operations by selecting **Edge Counting** from the **Mode** drop-down menu and clicking the **Start** button. The **Counter Value** indicator increments rapidly. Click **Stop** to stop the counter test.

□ Click the **Close** button to close the **Test Panel** dialog box and return to MAX.

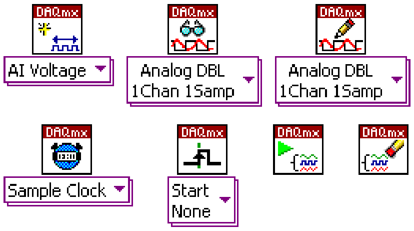
End of Exercise 8-1

Three Ways to Connect with a DAQ Device

NI MAX



DAQ Assistant

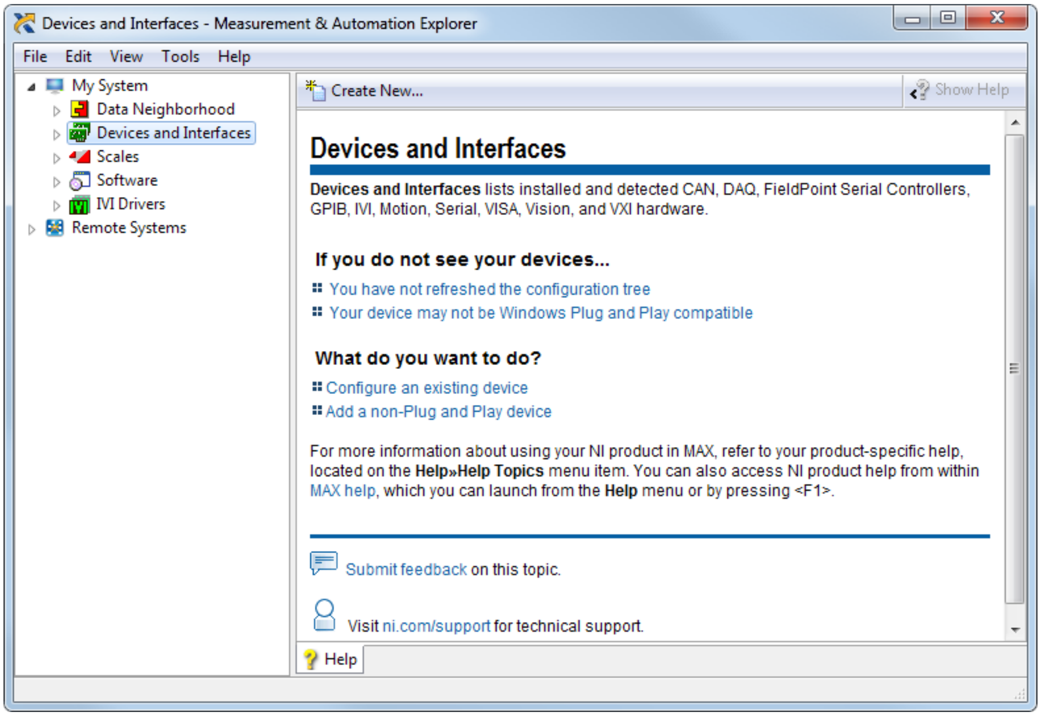


DAQmx API

Connecting to a DAQ Device Using MAX

MAX lets you quickly test the connection to your DAQ device and configure the hardware, if necessary. With it you can also create simulated DAQ devices to create and test your application even if you don't have hardware available.

Access MAX by double-clicking the NI MAX desktop icon or selecting **Tools»Measurement & Automation Explorer** in LabVIEW.



Devices and Interfaces

Devices and Interfaces lists installed and detected CAN, DAQ, FieldPoint Serial Controllers, GPIB, IVI, Motion, Serial, VISA, Vision, and VXI hardware.

If you do not see your devices...

- You have not refreshed the configuration tree
- Your device may not be Windows Plug and Play compatible

What do you want to do?

- Configure an existing device
- Add a non-Plug and Play device

For more information about using your NI product in MAX, refer to your product-specific help, located on the **Help»Help Topics** menu item. You can also access NI product help from within MAX help, which you can launch from the **Help** menu or by pressing <F1>.

Submit feedback on this topic.

Visit ni.com/support for technical support.

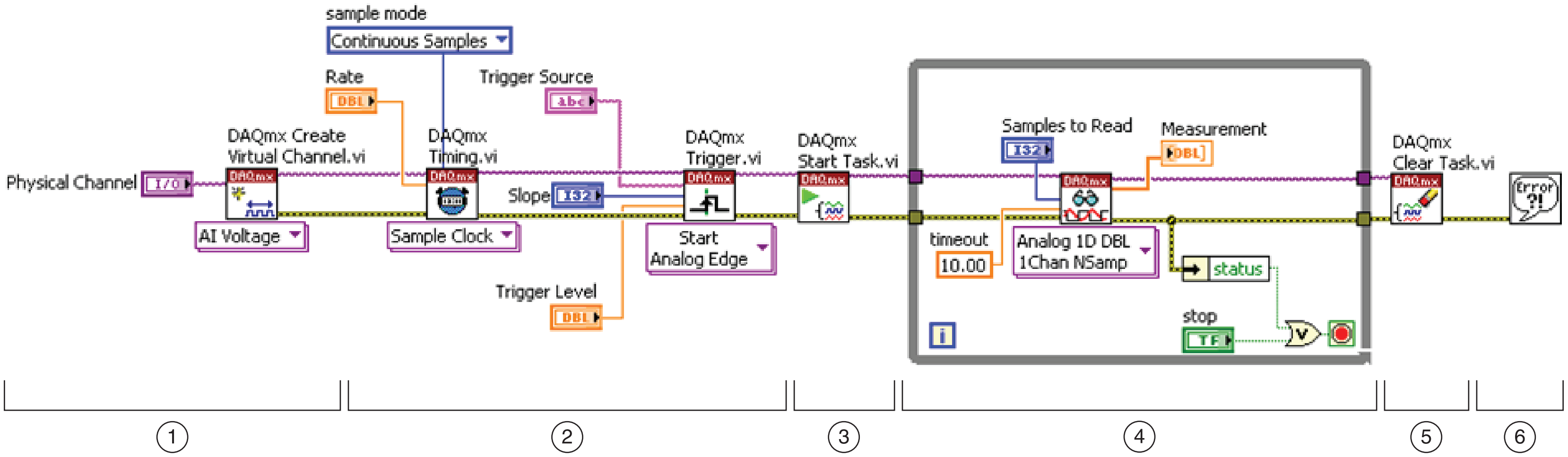
Connecting Using LabVIEW DAQmx VIs

The LabVIEW DAQmx VIs let you control your hardware with finer options then the DAQ Assistant Express VI. The DAQmx VIs also let you make the measurement or generation you want to perform globally accessible from any application.

A basic DAQmx application involves the following process.

- Create task
- Configure task
- Start task
- Acquire or generate data
- Clear task
- Check for errors

The following diagram shows how this process might be implemented.



1 Create task—The DAQmx Create Virtual Channel VI creates a channel and adds it to a task.
2 Configure task—The DAQmx Timing VI configures the starting and stopping of a task or the triggering of an application.
3 Start task—The DAQmx Start Task VI starts the task.
4 Acquire or generate data—The DAQmx Read VI or DAQmx Write VI acquires or generates data. You must select a compatible instance from the pull-down menu.
5 Clear task—The DAQmx Clear Task VI stops the task and releases resources.
6 Check for errors—Use the Simple Error Handler VI or other manual error handling techniques to check for and respond to errors in your data acquisition application.

Exercise 8-2 Programming with the DAQmx API

Goal

Explore a DAQmx example program that continuously acquires data, and modify it to wait on a digital trigger.

Scenario

Explore a DAQmx example program that continuously acquires a voltage signal on channel analog input 1 (AI1) of a DAQ device. Modify the VI to use a digital trigger. The VI begins measuring when the user sends a digital trigger to the device. In this exercise, the user sends a trigger by turning the **Quadrature Encoder** knob on the BNC-2120. The VI stops measuring when the user clicks the **Stop** button on the front panel of the VI.

Implementation—External Connections

1. If you are using the BNC-2120, connect the **Sine/Triangle** output on the function generator to channel **AI 1** with a BNC cable, and make sure the switch on the function generator is set to the sine wave. Also, ensure that there is a wire connecting the **UP/DN** screw terminal to the **PFI 1** screw terminal in the **Timing I/O** section.

Note The **UP/DN** terminal on the BNC-2120 outputs a high or a low signal indicating the rotation direction of the **Quadrature Encoder** knob. When you rotate the **Quadrature Encoder** knob clockwise, the **UP/DN** terminal outputs a high signal. When you rotate the **Quadrature Encoder** knob counterclockwise, the **UP/DN** terminal outputs a low signal. In this exercise, this signal triggers the VI to start acquiring data.

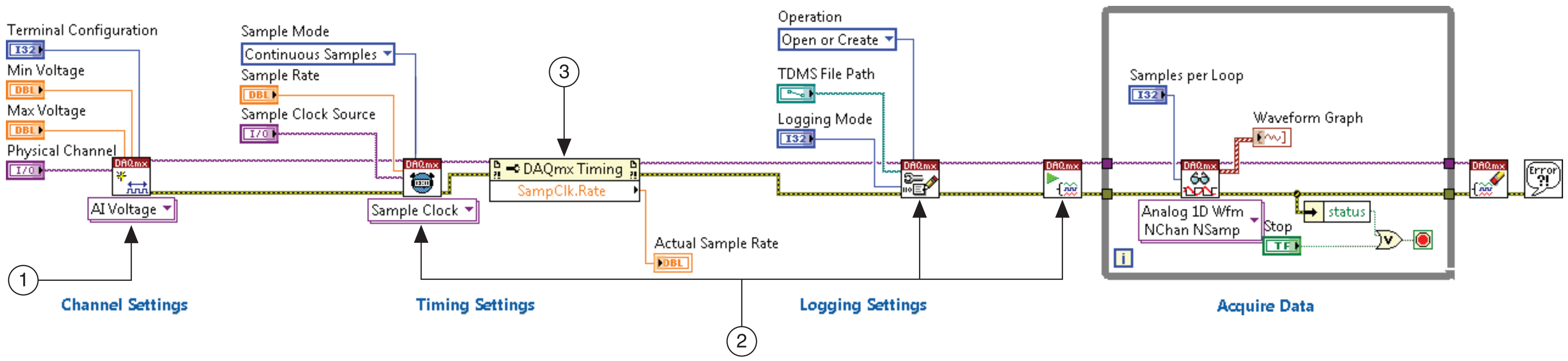
Open and Run a DAQmx Example

1. Open `Voltage - Continuous Input.vi` in the `<Exercises>\LabVIEW Core 1\Using DAQmx` directory. This VI demonstrates how to acquire a continuous amount of data from a DAQ device.

2. Select **File»Save As** to save the VI as `<Exercises>\LabVIEW Core 1\Triggered Analog Input\Trigger AI Acquisition.vi`. When prompted, select **Copy** - **Substitute copy for original**.

3. Open and explore the block diagram as shown in Figure 8-7.

Figure 8-7. Block Diagram of Original Voltage - Continuous Input VI



1 **DAQmx Create Virtual Channel VI**—Click the pull-down menu and notice it is set to **Analog Input»Voltage**.
2 Press <Ctrl-H> to open the **Context Help** window. Hover over each of the DAQmx functions to learn about each function.
3 **Property Node**—Gets or sets properties for a reference. You can learn more about Property Nodes in *LabVIEW Core 2* or refer to the *LabVIEW Help*.

4. Set the default values and settings on the front panel.

   □ Select **Dev1\ai1** from the **Physical Channel** control.

   □ Set **Max Voltage** to 1.

   □ Set **Min Voltage** to -1.

5. Run the VI. The VI should begin acquiring data continuously.

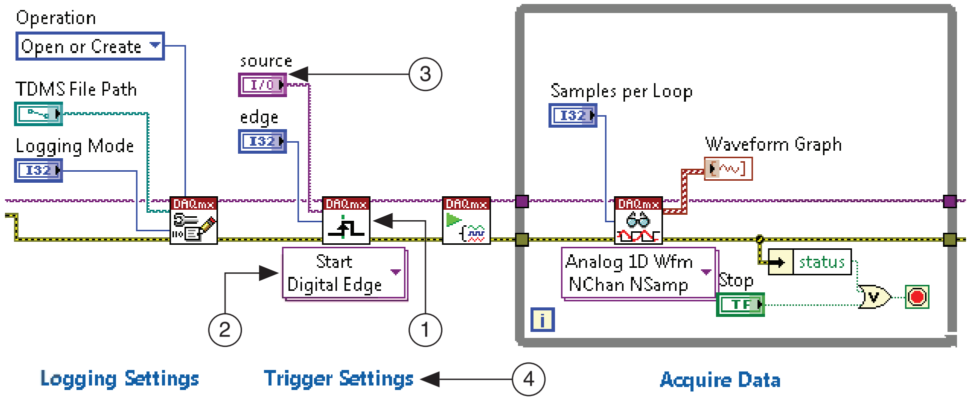
   □ Use the **Frequency Selection** switch and the **Frequency Adjust** knob on the BNC-2120 to change the frequency of the generated and acquired signal.

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

Add Triggering to the Example Program

1. Modify the block diagram as shown in Figure 8-8 to add trigger functionality. After you modify this VI, the VI waits for a trigger before acquiring data.

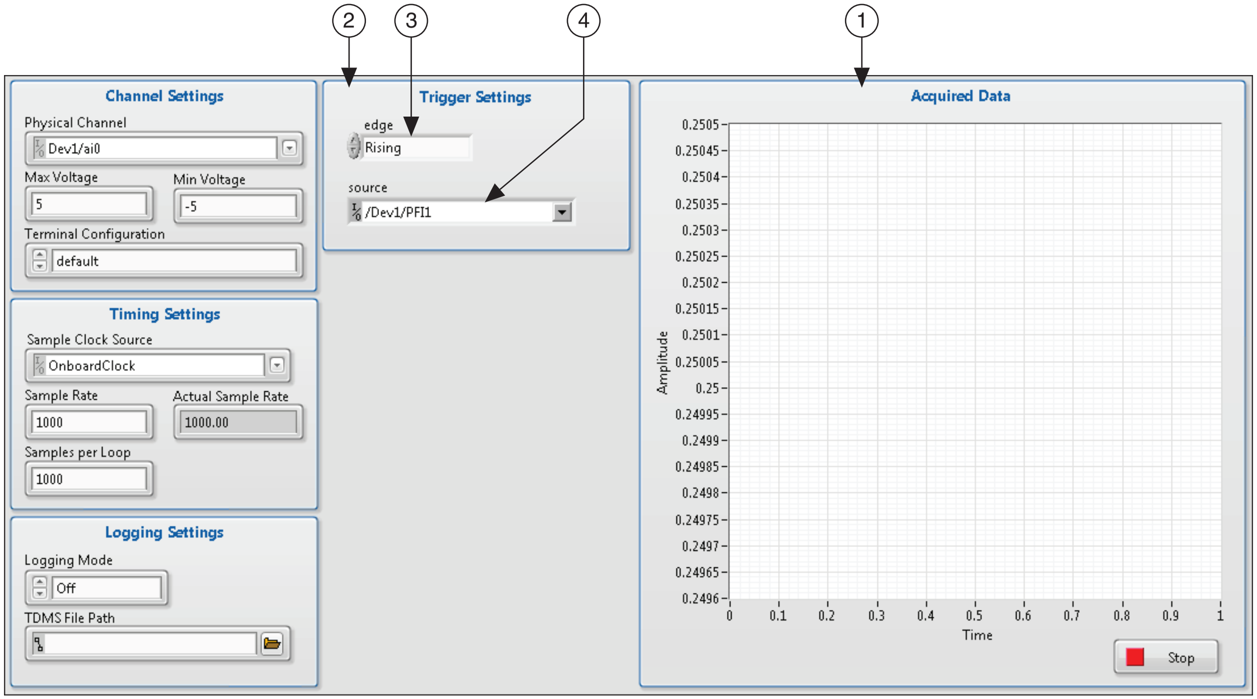
Figure 8-8. Trigger AI Acquisition VI Block Diagram



1 **DAQmx Trigger VI**—Place to the right of the DAQmx Configure Logging VI. Delete the **task out** and **error out** wires from the DAQmx Configure Logging VI and then wire them through the DAQmx Trigger VI to the DAQmx Start Task VI.
2 Configure the trigger—Click the DAQmx Trigger VI pull-down menu and select **Start»Digital Edge**.
3 Create controls—Right-click the **source** input and the **edge** input of the DAQmx Trigger VI and select **Create»Control**.
4 Free label—Create a label and enter `Trigger Settings`.

2. Modify the front panel and set the default settings as shown in Figure 8-9.

Figure 8-9. Trigger AI Acquisition VI Front Panel



Channel Settings
Physical Channel: Dev1/ai0
Max Voltage: 5
Min Voltage: -5
Terminal Configuration: default

Timing Settings
Sample Clock Source: OnboardClock
Sample Rate: 1000
Actual Sample Rate: 1000.00
Samples per Loop: 1000

Logging Settings
Logging Mode: Off
TDMS File Path

Trigger Settings
edge: Rising
source: /Dev1/PFI1

1 Move the **Acquired Data** group—Select all the items in the **Acquired Data** group and shift them to the right. <Shift>-click to select multiple items and press <Shift-arrow key> to move them.
2 Create Trigger Settings group—Copy a container from another group, label the copy `Trigger Settings`, and place the **edge** and **source** controls in it.
3 Select **Rising** in the **edge** control.
4 Select **Dev1/PFI1** in the **source** control.

3. Save the VI.

4. Run the VI. Turn the **Quadrature Encoder** knob on the BNC-2120 counterclockwise then clockwise to begin the acquisition.

5. Stop and close the VI.

End of Exercise 8-2

B. Automating Non-NI Instruments

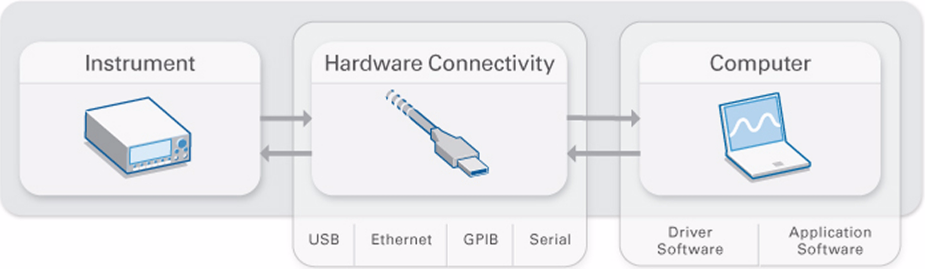
Objective: Recognize the components of an instrument control system and practice connecting to hardware.

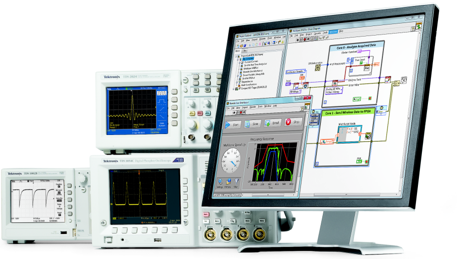
LabVIEW controls instruments through different buses.

LabVIEW can control stand-alone instruments, such as third-party oscilloscopes or analyzers, so you can automate instrument-based processes and consolidate multiple instrument tasks into one development environment.

Hardware connectivity The physical cable connecting the instrument. Also, the communication protocol, or bus.

A basic instrument control system consists of an instrument, hardware connectivity, and a computer running LabVIEW.

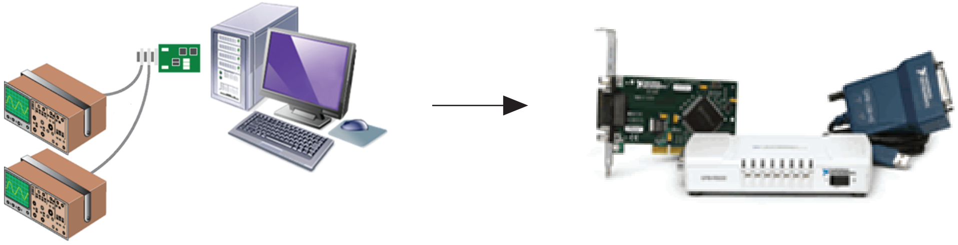




Multimedia: Automating Non-NI Instruments

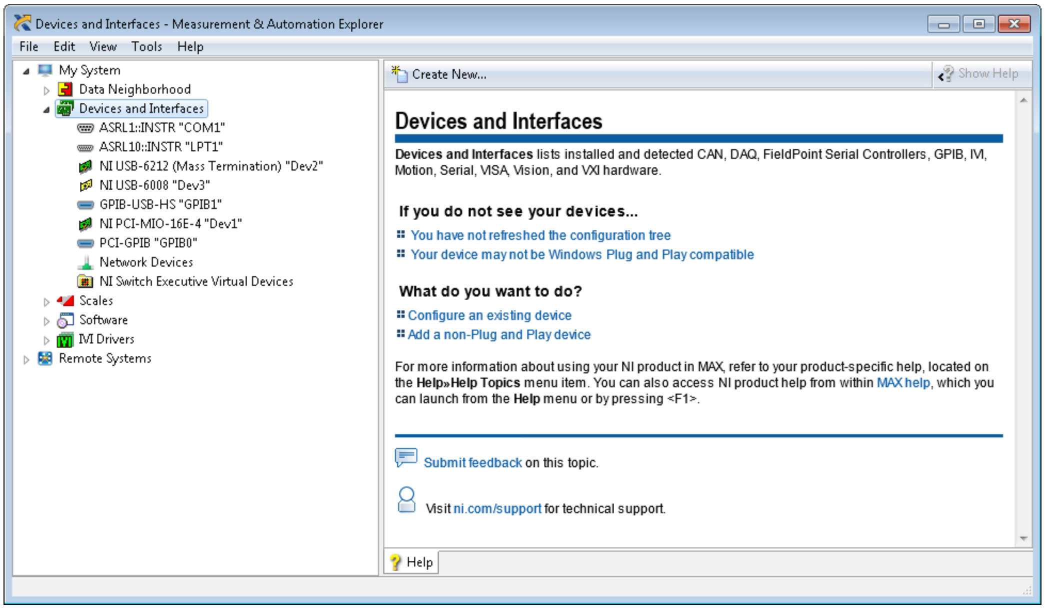
Common bus types include GPIB, serial, USB, and Ethernet. Different buses have different capabilities when it comes to latency and bandwidth. Instruments often have multiple ports for hardware connectivity.

Complete the multimedia module, *Automating Non-NI Instruments*, available in the `<Exercises>\LabVIEW Core 1\Multimedia\` folder to learn about the relationship between LabVIEW and instrument control and how to use VISA to communicate with third-party instruments



Configuring Instrument Control in NI MAX

Before creating your instrument control application, you should test communication with the instrument using NI MAX.



Devices and Interfaces

Devices and Interfaces lists installed and detected CAN, DAQ, FieldPoint Serial Controllers, GPIB, IVI, Motion, Serial, VISA, Vision, and VXI hardware.

If you do not see your devices...

- You have not refreshed the configuration tree
- Your device may not be Windows Plug and Play compatible

What do you want to do?

- Configure an existing device
- Add a non-Plug and Play device

For more information about using your NI product in MAX, refer to your product-specific help, located on the **Help»Help Topics** menu item. You can also access NI product help from within MAX help, which you can launch from the **Help** menu or by pressing <F1>.

Submit feedback on this topic.

Visit ni.com/support for technical support.

Exercise 8-3 Instrument Configuration with NI MAX

Goal

Learn to use MAX to examine and communicate with an instrument using GPIB interface settings.

Set Up

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

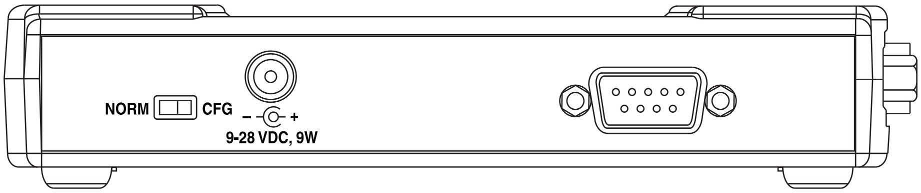
Complete the following steps to configure the NI Instrument Simulator to have a GPIB interface.

1. Configure the NI Instrument Simulator.

□ Power off the NI Instrument Simulator.

□ Set the configuration switch on the rear panel to CFG, as shown in Figure 8-10.

Figure 8-10. NI Instrument Simulator



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

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

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

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

□ Click **Next**.

□ Click **Next**.

□ On the **Select Interface** page, select **GPIB Interface** and click **Next**.

□ Select **Change GPIB Settings** and click **Next**.

□ Select **Single Instrument Mode** and click **Next**.

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

□ Set **GPIB Secondary Address** to `0(disabled)`.

□ Click **Next**.

□ Click **Update**.

□ Click **OK** when you get the message that the update was successful.

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

☐ Set the configuration switch on the rear panel to **NORM**.

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

☐ Verify that both the **PWR** and **RDY** LEDs are lit.

Implementation

1. Launch MAX by either double-clicking the icon on the desktop or by selecting **Tools» Measurement & Automation Explorer** in LabVIEW.

2. View the settings for the GPIB interface.

☐ Expand the **Devices and Interfaces** section to display the installed interfaces. If a GPIB interface is listed, the NI-488.2 software is correctly loaded on the computer.

☐ Select the GPIB interface.

☐ Examine but do not change the settings for the GPIB interface.

3. Communicate with the GPIB instrument.

☐ Make sure the GPIB interface is still selected in the **Devices and Interfaces** section.

☐ Click the **Scan for Instruments** button on the toolbar.

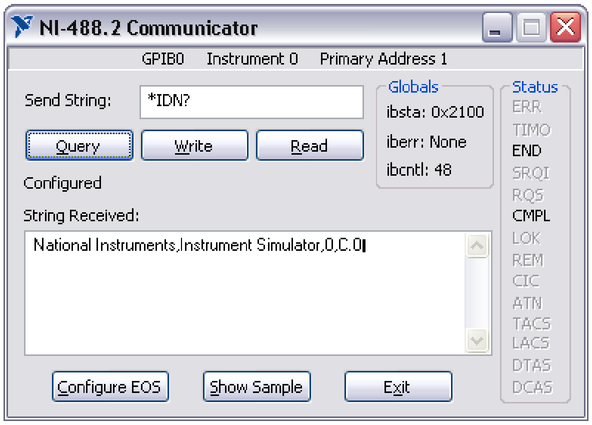
☐ Expand the GPIB interface selected in the **Devices and Interfaces** section. An instrument named `Instrument Simulator` appears.

☐ Click `Instrument Simulator` to display information about it in the right pane of MAX. Click the **Attributes** tab. Notice the NI Instrument Simulator has a GPIB primary address.

☐ Click the **Communicate with Instrument** button on the toolbar. An interactive window appears. You can use it to query, write to, and read from that instrument.

☐ Enter `*IDN?` in the **Send String** text box and click the **Query** button. The instrument returns its make and model number in the **String Received** indicator as shown in Figure 8-11. You can use this communicator window to debug instrument problems or to verify that specific commands work as described in the instrument documentation.

Figure 8-11. Communication with the GPIB instrument



NI-488.2 Communicator

GPIB0 Instrument 0 Primary Address 1

Send String: *IDN?

Configured

String Received:

National Instruments,Instrument Simulator,0,C.0

Globals
ibsta: 0x2100
iberr: None
ibcntl: 48

☐ 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.

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

    - □ While `Instrument Simulator` is selected in MAX, click the **VISA Properties** tab.
    - □ Enter `devsim` in the **VISA Alias on My System** field. You use this alias later in the course.
    - □ Click **Save**.

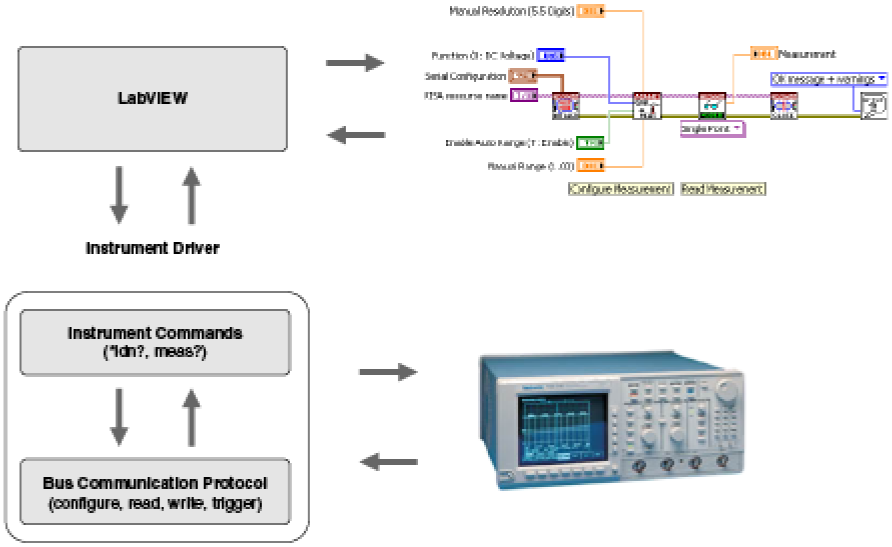
5. Select **File»Exit** to exit MAX.

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

End of Exercise 8-3

Simplify Instrument Control

Controlling your benchtop instrument using LabVIEW helps you save time by automating processes using the instrument such as testing and data logging. Instrument control in LabVIEW also lets you consolidate multiple instrument tasks into one development environment.

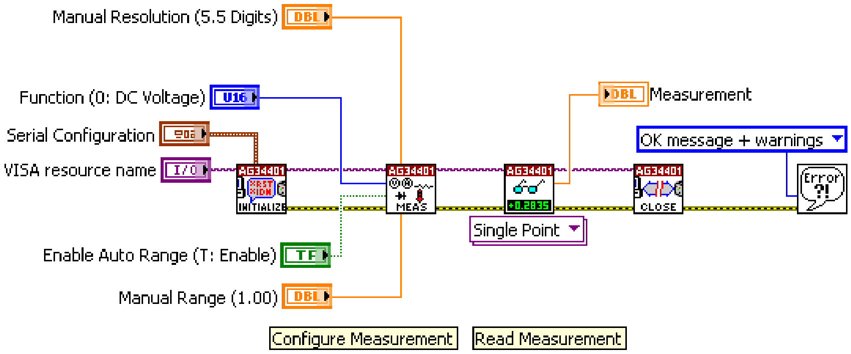


LabVIEW instrument drivers, available on the Instrument Driver Network at `ni.com/idnet`, provide the following additional benefits:

- Provide a high-level API, meaning that VIs perform multiple instructions, thereby simplifying the code.
- Do not require knowledge of different bus protocols, such as GPIB or serial, that you might use to connect with the instrument.
- Do not require learning low-level programming commands for each instrument.

Instrument Driver VIs for an Agilent Digital Multimeter

Every application that uses an instrument driver has a similar sequence of events: Initialize, Configure, Read Data, and Close.The block diagram initializes the Agilent 34401 digital multimeter (DMM), uses a configuration VI to choose the resolution and range, select the function, and enable or disable auto range, uses a data VI to read a single measurement, closes the instrument, and checks the error status.



The VIs in an instrument driver are organized into six categories. These categories are summarized in the following table.

Category	Description
Initialize	The Initialize VI establishes communication with the instrument and is the first instrument driver VI called.
Configure	Configure VIs are software routines that configure the instrument to perform specific operations. After calling these VIs, the instrument is ready to take measurements or stimulate a system.
Action/Status	Action/Status VIs command the instrument to carry out an action (for example, arming a trigger) or obtain the current status of the instrument or pending operations.
Data	The data VIs transfer data to or from the instrument.
Utility	Utility VIs perform a variety of auxiliary operations, such as reset and self-test.
Close	The close VI terminates the software connection to the instrument. This is the last instrument driver VI called.

Exercise 8-4 Exploring Instrument Drivers

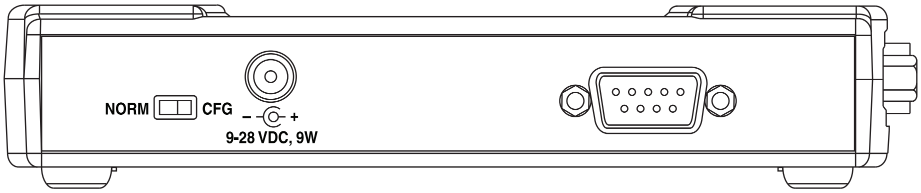
Goal

Install an instrument driver and explore the example programs that accompany it.

Implementation

Install the instrument driver for the NI Instrument Simulator. After installation, explore the VIs that the instrument driver provides and the example programs that are added to the NI Example Finder.

Figure 8-12. NI Instrument Simulator



Install Instrument Driver and Open Project—Download from Internet

If you have internet access and have, or want to create, a user profile on `ni.com`, complete the following steps. Otherwise, install the driver from the course CD following the instructions in the *Install Instrument Driver and Open Project—Extract from Disk* section.

1. Select **Help»Find Instrument Drivers**.
2. Click the **Login** button.
3. If you have an `ni.com` profile, log in with your ID.
4. If you do not have an `ni.com` profile, follow the onscreen instructions to create one at this time. Be sure to make a note of the user ID and password you create.
5. After you log in, click the **Scan for Instruments** button. If you have an instrument connected, clicking this button detects the instrument and finds the correct driver. In this case, it detects the Instrument Simulator.
6. Double-click **Instrument Simulator** in the list and click the **Search** button.
7. Select Version 2.0 of the instrument driver from the **Driver** list.
8. Click the **Install** button.
9. Click the **Start using this driver** button.
10. Click the **Open Project** button.
11. Expand the **Examples** folder in the **Project Explorer** window.
12. Close the NI Instrument Driver Finder window and go to the *Explore Instrument Driver* section to continue this exercise.

Install Instrument Driver and Open Project—Extract from Disk

If you do not have Internet access or do not want to create a user profile, complete the following steps to install the instrument driver.

1. If you have it open, close LabVIEW and then navigate to the `<Exercises>\LabVIEW Core 1\Instrument Driver` directory. This folder contains a zip file with the LabVIEW Plug and Play instrument drivers for the Instrument Simulator.
2. Right-click the zip file and follow the wizard to extract all files to the `<Program Files>\National Instruments\LabVIEW 2014\instr.lib` directory.
3. Open `National Instruments Instrument Simulator.lvproj` in the `<Program Files>\National Instruments\LabVIEW 2014\instr.lib\National Instruments Instrument Simulator` directory.
4. Expand the **Examples** folder in the **Project Explorer** window.
5. Go to the *Explore Instrument Driver* section to continue this exercise.

Explore Instrument Driver

1. Open **National Instruments Instrument Simulator Acquire Single Measurement(DMM).vi** from the **Examples** folder in the **Project Explorer** window of the National Instruments Instrument Simulator project.

   This VI reads a single measurement from the Instrument Simulator.
2. Verify that the **PWR** and **RDY** LEDs are lit on the Instrument Simulator.
3. Select **devsim** from the **VISA Resource Name** control on the front panel. You specified the VISA alias for this GPIB instrument as `devsim` in step 4 of Exercise 8-3, *Instrument Configuration with NI MAX*.
4. Run the VI.
5. Open the block diagram of the VI.
6. Open the **Functions** palette and navigate to the **Instrument I/O»Instrument Drivers»National Instruments Instrument Simulator** palette.
7. Explore the palette and subpalettes using the **Context Help** window to familiarize yourself with the functionality of functions on the palette.
8. Stop and close the VI. Do not save changes.
9. Open **National Instruments Instrument Simulator Acquire Waveform(Scope).vi** from the project. This VI reads a single waveform from the Instrument Simulator.
10. Select the same VISA resource name you selected in step 3.
11. Run the VI.
12. Select a different function from the **Waveform Function** control.
13. Run the VI again.
14. Explore the block diagram of the VI.

    □ To familiarize yourself with the VI's functionality, explore the block diagram using the **Context Help** window.

    □ Double-click the **Read Waveform VI** to open it.

- □ Open to the block diagram of the Read Waveform VI. Notice that it uses VISA functions to communicate with the instrument.

15. Close the VIs and project when you are finished. Do not save changes.

End of Exercise 8-4

Additional Resources

Learn More About	LabVIEW Help Topic or ni.com
Creating a DAQ system	`ni.com/DAQ` *General-Purpose DAQ* *Creating a Typical DAQ Application*
Signal types and other measurement fundamentals	*Taking Measurements* *Displaying a Signal*

Setting up and automating instrument control system	`ni.com/instrument-control`
	`ni.com/idnet`
	Using Instrument Drivers
	Creating a Typical VISA Application

NI MAX	*Configuring DAQ Devices, Instruments, and Other Devices* *LabVIEW Tools for DAQ Configuration (Windows)*

Activity 8-1: Lesson Review

1. What is NI MAX? (multiple answers)
   a. A tool to configure and test DAQ devices
   b. A tool to test instrument communication
   c. A configurable Express VI
   d. A window to view LabVIEW project files

2. Which of the following are benefits of instrument control? (multiple answers)
   a. Automate processes
   b. Improve productivity and repeatability
   c. One platform for multiple tasks
   d. Limited to only one type of instrument

3. VISA is a high-level API that calls low-level drivers.
   a. True
   b. False

Activity 8-1: Lesson Review - Answers

1. What is NI MAX? (multiple answers)
   - **a. A tool to configure and test DAQ devices**
   - **b. A tool to test instrument communication**
   - c. A configurable Express VI
   - d. A window to view LabVIEW project files

2. Which of the following are benefits of instrument control? (multiple answers)
   - **a. Automate processes**
   - **b. Improve productivity and repeatability**
   - **c. One platform for multiple tasks**
   - d. Limited to only one type of instrument

3. VISA is a high-level API that calls low-level drivers.
   - **a. True**
   - b. False

9 Accessing Files in LabVIEW

In this lesson you will learn to describe the basic concept of file I/O and apply the appropriate File I/O functions to a given scenario.

Topics

+ Accessing Files from LabVIEW
+ High-Level and Low-Level File I/O Functions
+ Comparing File Formats

Exercises

Exercise 9-1 Exploring High-Level File I/O

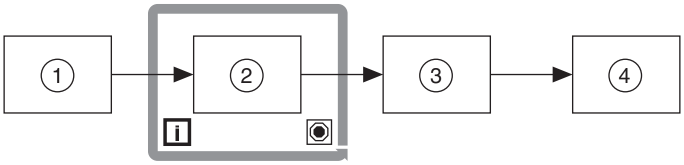
Exercise 9-2 Temperature Monitor VI—Logging Data

A. Accessing Files from LabVIEW

Objective: Identify the steps for writing and reading files from a LabVIEW application.

Typical File I/O Operations

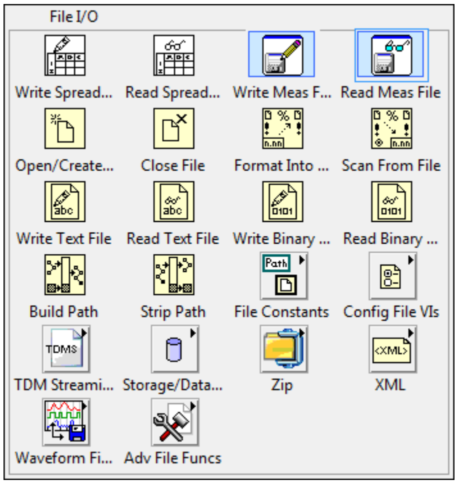
File I/O operations pass data to and from files.



1	Open File	3	Close File
2	Read/Write File	4	Check for Errors

File I/O Palette for File Operation Functions

The File I/O palette includes functions to create or open a file, read data from or write data to the file, and close the file. You can also use the functions to create directories; move, copy, or delete files; list directory contents; change file characteristics; or manipulate paths.



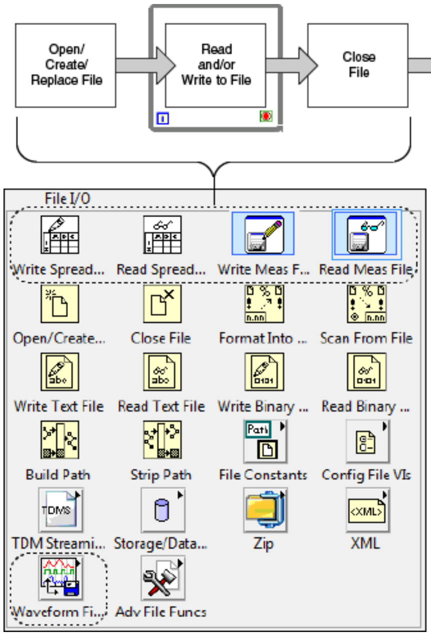
B. High-Level and Low-Level File I/O Functions

Objective: Identify when to use high-level and low-level File I/O functions.

High-Level File I/O Functions	Low-Level File I/O Functions
Perform three steps of the file I/O process. (open/write/close)	Perform one step of the file I/O process.
Simplify block diagram.	Provide finer control of file access.
Create unnecessary resource overhead when used in loops.	Save memory resources when used in loops.
Are good to use when writing to a file in a single operation.	Are good to use when streaming data to disk.

High-Level File I/O

High-level File I/O functions combine three steps (open, read/write, close) for common file I/O operations. High-level file I/O functions simplify the block diagram but might not provide the control or configuration you need for your application.



The following table compares the high-level file I/O functions.

Function	Type of Data	File Formats
Write To Spreadsheet File Read From Spreadsheet File	1D or 2D arrays	Text
Write To Measurement File Read From Measurement File	Signals (dynamic data types)	.lvm (text) .tdms (binary)
Write Waveforms to File Read Waveforms from File	Waveform data	Text

Exercise 9-1 Exploring High-Level File I/O

Goal

Observe how a high-level file I/O VI writes to a spreadsheet-readable file.

Scenario

The Spreadsheet Example VI does the following:

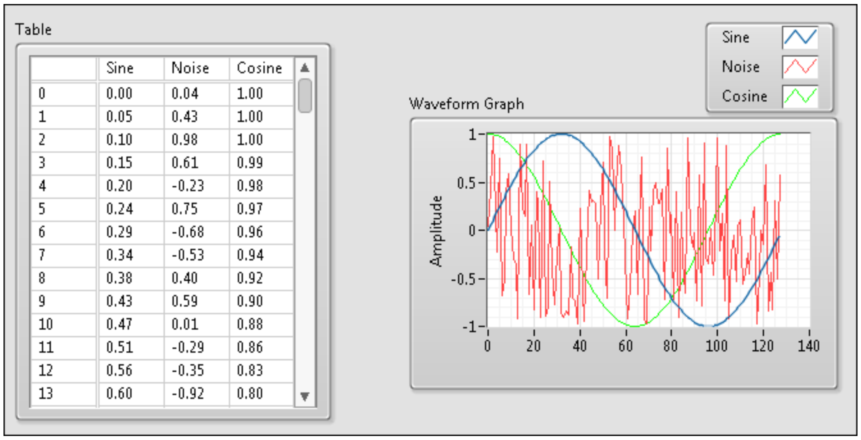
- Generates sine, noise, and cosine data for 128 points
- Stores this data in a 2D array that is 128 rows × 3 columns.
- Displays the 2D array in a **Table** indicator with three columns (Sine, Noise, and Cosine) for the first 14 rows of the array.
- Plots each column in a **Waveform Graph** indicator.
- Uses the Write To Spreadsheet File VI to save the numeric 2D array in a text file so a spreadsheet application can access the file.

Implementation

Complete the following steps to examine how the Spreadsheet Example VI performs the tasks described in the *Scenario* section.

1. Open `Spreadsheet Example.lvproj` in the `<Exercises>\LabVIEW Core 1\Spreadsheet Example` directory.
2. Open **Spreadsheet Example.vi** from the **Project Explorer** window.

Figure 9-1. Spreadsheet Example VI Front Panel

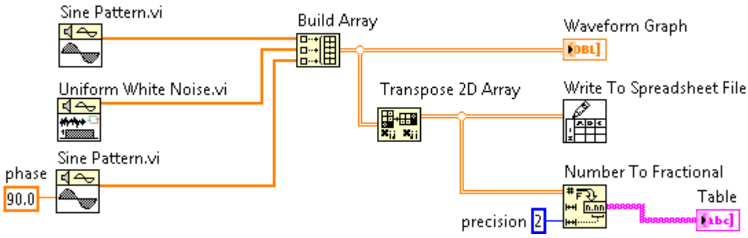


Table

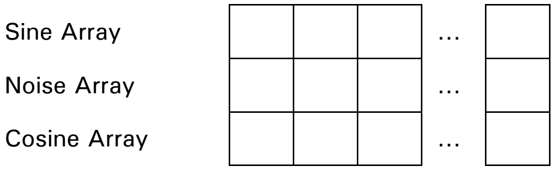
	Sine	Noise	Cosine
0	0.00	0.04	1.00
1	0.05	0.43	1.00
2	0.10	0.98	1.00
3	0.15	0.61	0.99
4	0.20	-0.23	0.98
5	0.24	0.75	0.97
6	0.29	-0.68	0.96
7	0.34	-0.53	0.94
8	0.38	0.40	0.92
9	0.43	0.59	0.90
10	0.47	0.01	0.88
11	0.51	-0.29	0.86
12	0.56	-0.35	0.83
13	0.60	-0.92	0.80

3. Run the VI.
4. Save the file, when prompted, as `wave.txt` in the `<Exercises>\LabVIEW Core 1\Spreadsheet Example` directory and click the **OK** button. You examine this file later.
5. Display and examine the block diagram for this VI.

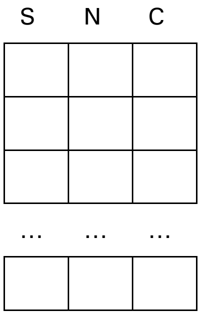
Figure 9-2. Spreadsheet Example VI Block Diagram



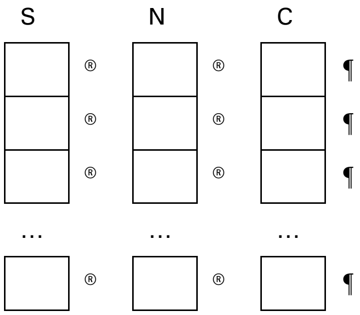
- Sine Pattern VI—Returns a numeric array of 128 elements containing a sine pattern. The constant `90.0`, in the second instance of the Sine Pattern VI, specifies the phase of the sine pattern which generates the cosine pattern.
- Uniform White Noise VI—Returns a numeric array of 128 elements containing a noise pattern.
- Build Array function—Builds the following 2D array from the sine array, noise array, and cosine array.



- Transpose 2D Array function—Rearranges the elements of the 2D array so element `[i,j]` becomes element `[j,i]`, as follows.



- Write To Spreadsheet File VI—Formats the 2D array into a spreadsheet string and writes the string to a file. The string has the following format, where an arrow (→) indicates a tab, and a paragraph symbol (¶) indicates an end of line character.



- Number To Fractional String function—Converts an array of numeric values to an array of strings that the table displays.

6. Close the VI. Do not save changes.

Note This example stores only three arrays in the file. To include more arrays, increase the number of inputs to the Build Array function.

7. Open the `wave.txt` file using a word processor, spreadsheet application, or text editor and view its contents.

   □ Open a word processor, spreadsheet application, or text editor, such as Notepad or WordPad.

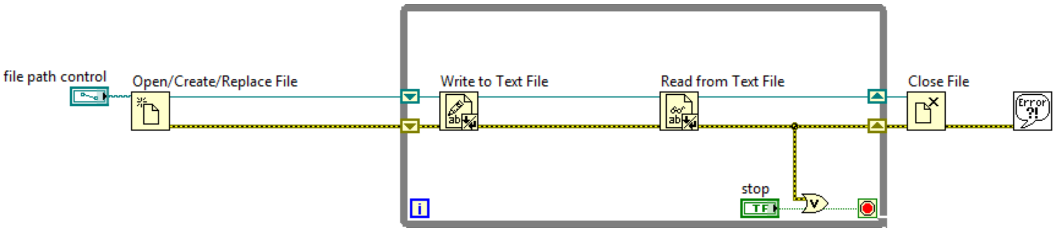
   □ Open `wave.txt`. The sine waveform data appear in the first column, the random (noise) waveform data appear in the second column, and the cosine waveform data appear in the third column.

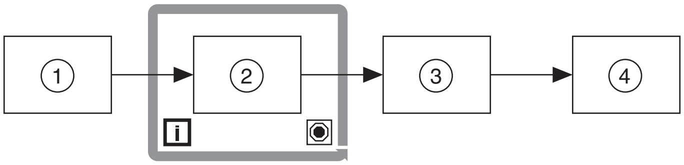
8. Exit the word processor or spreadsheet application and return to LabVIEW.

End of Exercise 9-1

Low-Level File I/O

Low-level File I/O functions provide individual functions for each step in file I/O operations. For example, one function opens an ASCII file, one function reads an ASCII file, and one function closes an ASCII file.

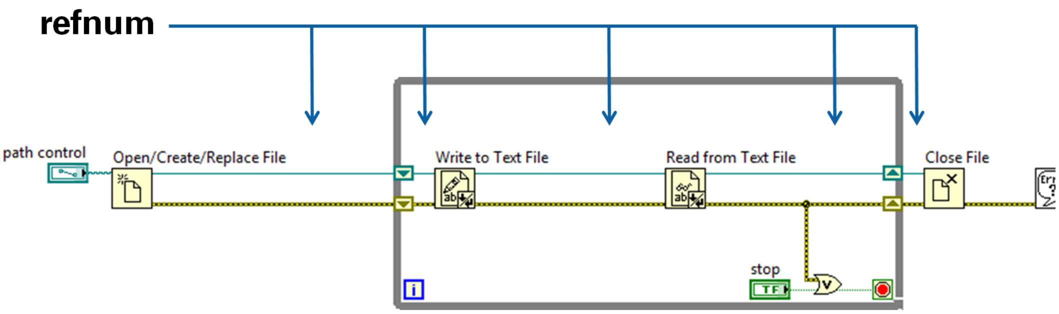




1 Open, Initialize or create file resource—LabVIEW creates a reference number (refnum) as a unique identifier for the resource.
2 Read/Write to file
3 Close file resource—The refnum becomes obsolete
4 Check for errors—Display any errors from the file resource.

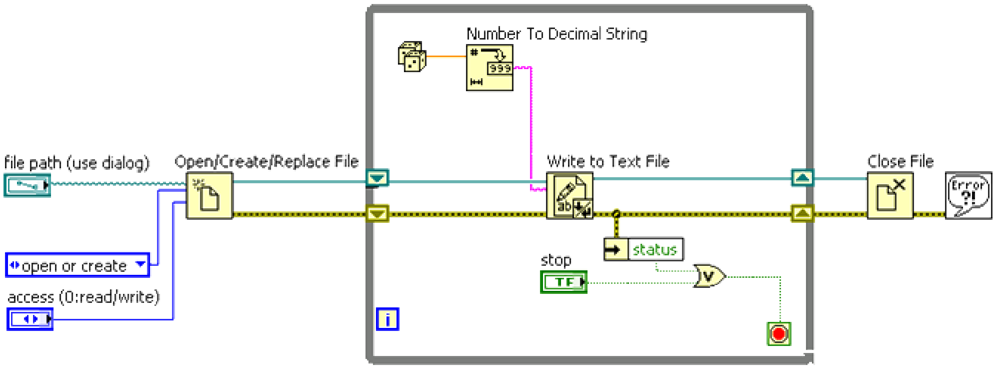
File Refnums

File refnums identify unique file I/O sessions.



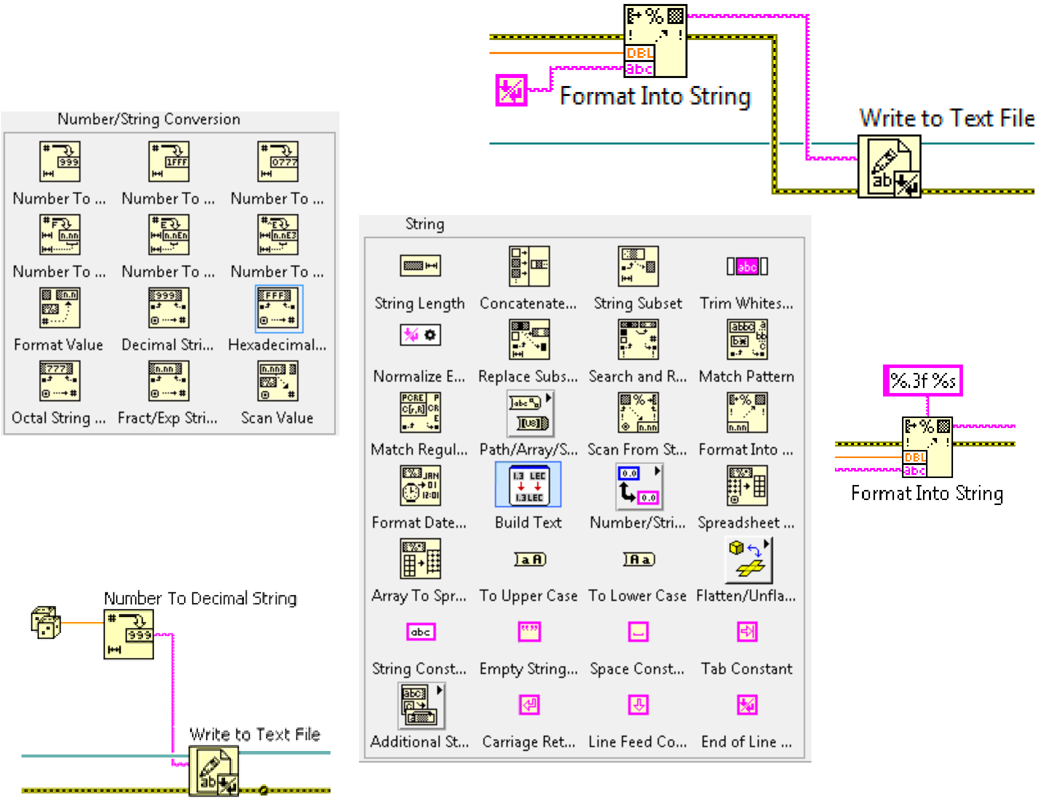
Streaming Data to Disk

Disk streaming is a technique for keeping files open while you perform multiple write operations. Use low-level functions when file I/O is occurring within a loop to save memory resources.



String Functions

To write data to a text file, you must first convert the data into a string data type. Use the items from the String palette to convert numerics and other data types to text.



Tip Refer to the *LabVIEW Help* for details about formatting a string using the Format Into String function.

Exercise 9-2 Temperature Monitor VI—Logging Data

Goal

Modify a VI to create an ASCII file using disk streaming.

Scenario

You have been given a VI that plots the current temperature and the average of the last three temperatures. Modify the VI to log the current temperature to an ASCII file.

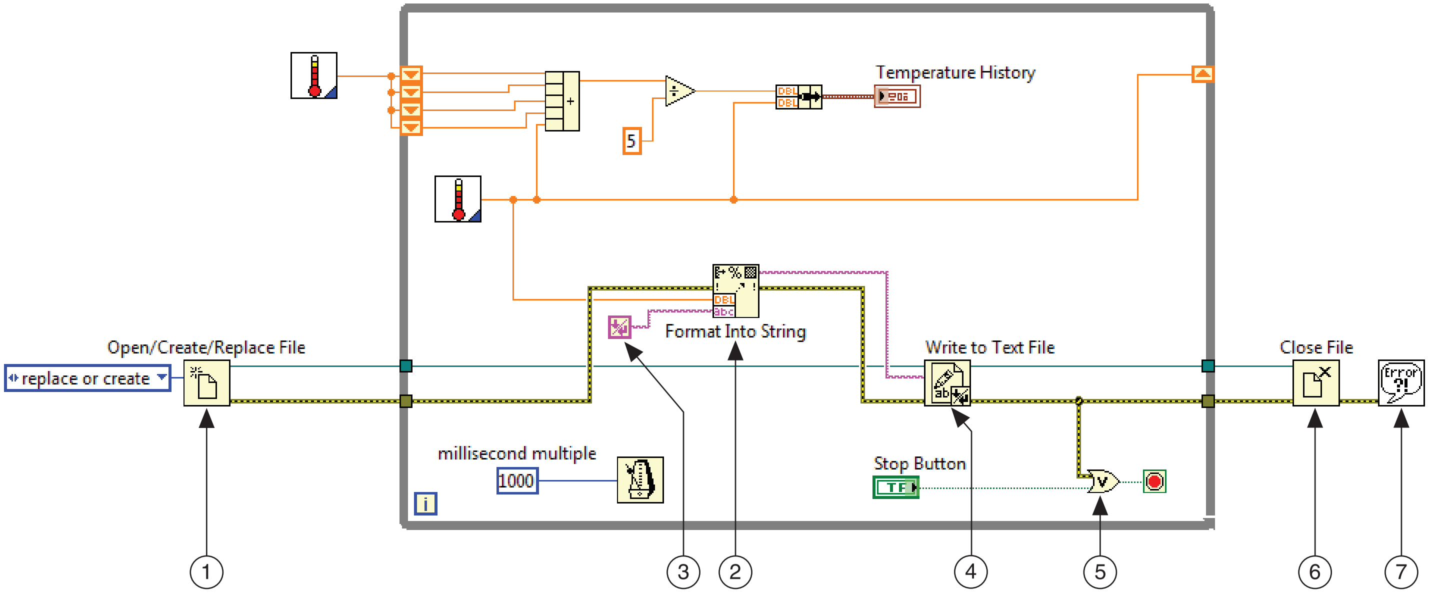
Implementation

1. Open `Temperature Monitor.lvproj` in the `<Exercises>\LabVIEW Core 1\Temperature Monitor` directory.

2. Open **Temperature Monitor.vi** from the **Project Explorer** window.

3. Modify the block diagram as shown in Figure 9-3.

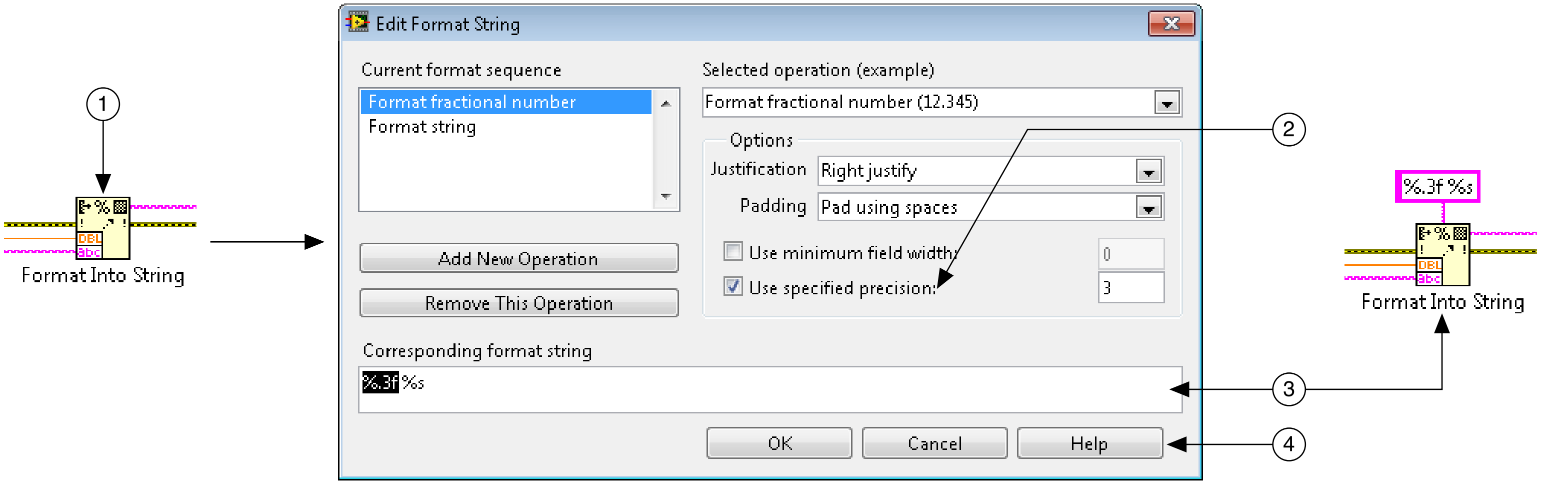
Figure 9-3. Temperature Monitor VI with Logging Block Diagram



1 **Open/Create/Replace File**—Creates or replaces an existing file for the data log. Right-click the **operation** input and select **Create»Constant**. Set the constant to **replace or create**.
2 **Format Into String**—Formats temperature data into a string. Expand the node to accept two inputs.
3 **End of Line Constant**—Adds an end-of-line constant after each piece of data so that data values are separated by line breaks.
4 **Write to Text File**—Writes the data to a file.
5 **Or**—Stops the VI when an error occurs or when the **Stop Button** is clicked.
6 **Close File**—Closes the data log file created or replaced when the VI started running.
7 **Simple Error Handler**—Indicates whether an error occurred. If an error occurred, this VI returns a description of the error and optionally displays a dialog box.

4. Configure the Format Into String function as shown in Figure 9-4.

Figure 9-4. Configuring the Format Into String Function



Current format sequence: Format fractional number; Format string

Selected operation (example): Format fractional number (12.345)

Options

Justification: Right justify

Padding: Pad using spaces

☐ Use minimum field width: 0

☑ Use specified precision: 3

Corresponding format string: %.3f %s

1 Format Into String—Double-click the Format Into String function to open the **Edit Format String** dialog box.

2 **Use specified precision**—Place a checkmark in this checkbox and enter 3 in the text box to specify that data have a floating point precision of three digits.

3 **Corresponding format string**—This text box automatically updates based on the configuration you specify. After you click the **OK** button in the dialog box, the block diagram updates to display the format string.

4 **Help** button—Click the **Help** button for more information about format specifier elements, such as %3f, and configuration options for the Format Into String function.

5. Test the VI.

☐ Run the VI.

☐ Give the text file a name and a location.

☐ Click the **Stop** button after the VI has been running for a few samples.

☐ Navigate to the text file created and explore it.

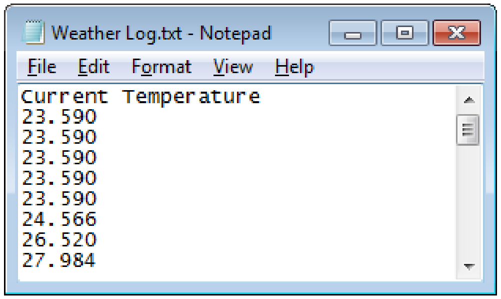
6. Close the VI and text file when you have finished.

Challenge

Objective: Create Log File with Single Header

To improve the usability of the log file, you are asked to include a header at the top of the log file as shown in Figure 9-5.

Figure 9-5. Temperature Monitor VI Log File with Header



```
Current Temperature
23.590
23.590
23.590
23.590
23.590
24.566
26.520
27.984
```

Modify the Temperature Monitor VI to include the header `Current Temperature`.

Hints:

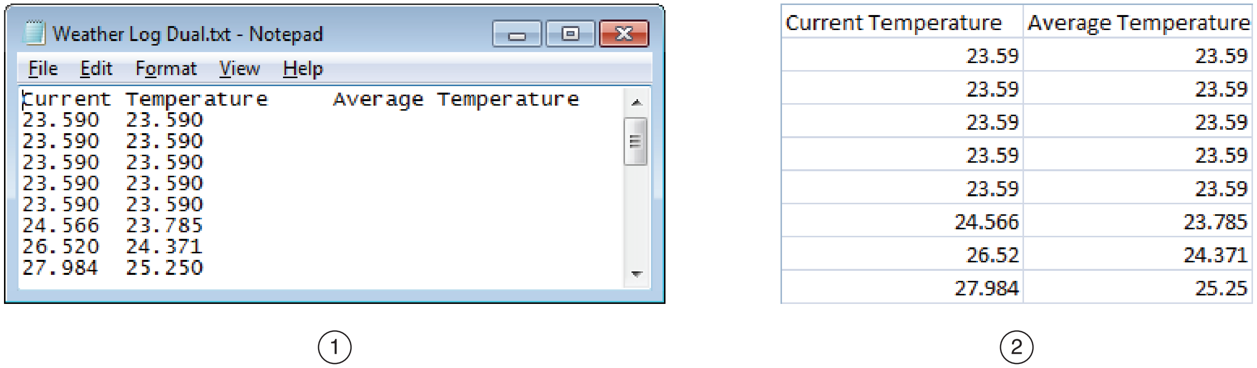
- Because you write the header to the text file only once, you should write to the header outside the While Loop.
- Use the functions on the **Strings** palette to manipulate and format a string for use in a word processing or spreadsheet application.

Challenge

Objective: Create Log File with Two Columns and Headers

Modify the VI to write both the current temperature and the average temperature to the log file. Separate the columns of data with a tab character and place a header at the top of each column as shown in Figure 9-6.

Figure 9-6. Temperature Monitor VI Log File with Two Columns and Headers



```
Current Temperature      Average Temperature
23.590   23.590
23.590   23.590
23.590   23.590
23.590   23.590
23.590   23.590
24.566   23.785
26.520   24.371
27.984   25.250
```

Current Temperature	Average Temperature
23.59	23.59
23.59	23.59
23.59	23.59
23.59	23.59
23.59	23.59
24.566	23.785
26.52	24.371
27.984	25.25

1 Tabbed columns in a text editor.

2 Tabbed columns in a spreadsheet application.

Hint: Use the Format Into String function and expand it to convert and format the data into a string.

End of Exercise 9-2

C. Comparing File Formats

Objective: Recognize different options for logging data to disk.

Common Log File Formats

LabVIEW can use or create the following file formats: Binary, ASCII, LVM, and TDMS.

- **ASCII**—An ASCII file is a specific type of binary file that is a standard used by most programs. It consists of a series of ASCII codes. ASCII files are also called text files.
- **LVM**—The LabVIEW measurement data file (`.lvm`) is a tab-delimited text file you can open with a spreadsheet application or a text-editing application. The `.lvm` file includes information about the data, such as the date and time the data was generated. This file format is a specific type of ASCII file created for LabVIEW.
- **Binary**—Binary files are the underlying file format of all other file formats.
- **TDMS**—This file format is a specific type of binary file created for National Instruments products. It actually consists of two separate files—a binary file that contains data and stores properties about the data, and a binary index file that provides consolidated information on all the attributes and pointers in the binary file.

Comparing File Formats

Use text files when you want to access the file from another application, if disk space and file I/O speed are not crucial, if you do not need to perform random access read or writes, and if numeric precision is not important.

	ASCII	Binary	TDMS	LVM
Easily Exchangeable	X		X	X
Small disk footprint		X	X	
Random read/write access		X	X	X
Inherent attributes		X	X	
High-speed streaming			X	

Additional Resources

Learn More About	LabVIEW Help Topic
File I/O	*File I/O*
Streaming data to disk	*Saving Memory using Disk Streaming*
	Streaming External Data to a TDMS File (Windows)
File Formats for file I/O	*Determining Which File Format to Use*
	Converting Numbers into Strings

Activity 9-1: Lesson Review

1. After opening a file, which output does the Open/Create/Replace File I/O function return?
   a. File path
   b. File name
   c. Refnum out
   d. Task out

2. Which file format is best in an application that requires high-speed streaming?
   a. ASCII
   b. Binary
   c. TDMS
   d. LVM

Activity 9-1: Lesson Review - Answers

1. After opening a file, which output does the Open/Create/Replace File I/O function return?

    a. File path

    b. File name

    c. Refnum out

    d. Task out

2. Which file format is best in an application that requires high-speed streaming?

    a. ASCII

    b. Binary

    c. TDMS

    d. LVM

10 Using Sequential and State Machine Programming

In this lesson you will learn how to identify sequential and state programming and explore the State Machine design pattern.

Topics

- Using Sequential Programming
- Using State Programming
- State Machines

Exercises

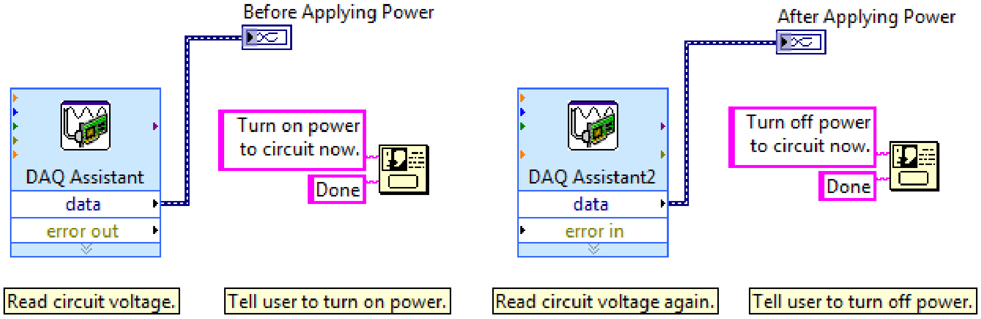
Exercise 10-1 Weather Station Project

A. Using Sequential Programming

Objective: Use dataflow to ensure sequential execution of nodes.

Why Use Sequential Programming?

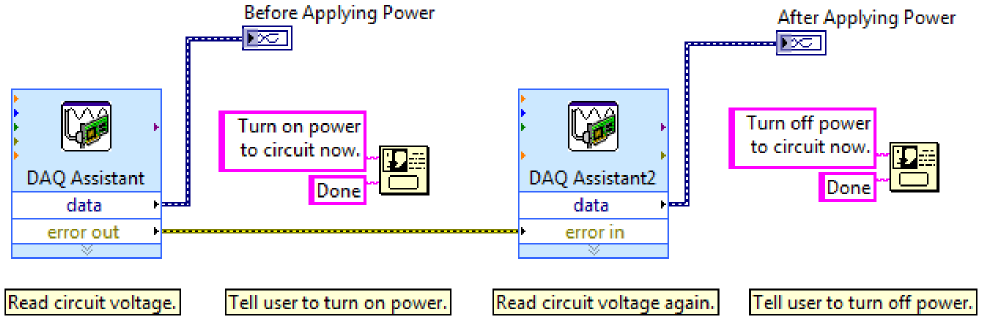
Sequential programming ensures the execution order of tasks. In the following block diagram there is no mechanism to force the execution order of these events. Any one of these events could happen first.


Before Applying Power
After Applying Power
DAQ Assistant
data
error out
Turn on power to circuit now.
Done
DAQ Assistant2
data
error in
Turn off power to circuit now.
Done
Read circuit voltage.
Tell user to turn on power.
Read circuit voltage again.
Tell user to turn off power.

Flow-Through Parameters

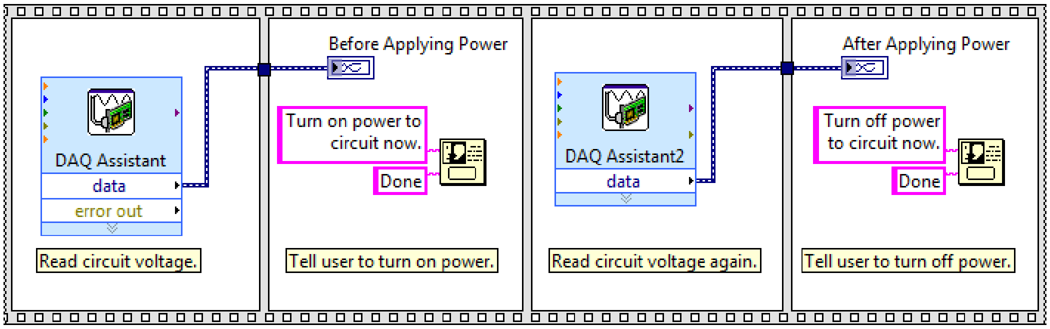
To enforce sequential programming in LabVIEW, you can complete sequential tasks by placing each task in a separate subVI, and wiring the subVIs in the order you want them to execute using the error cluster wires.

However, in this example, only two of the tasks have a error cluster. Using the error clusters, you can force the execution order of the two DAQ Assistants, but not the One Button Dialog functions, as shown in the following figure.


Before Applying Power
After Applying Power
DAQ Assistant
data
error out
Turn on power to circuit now.
Done
DAQ Assistant2
data
error in
Turn off power to circuit now.
Done
Read circuit voltage.
Tell user to turn on power.
Read circuit voltage again.
Tell user to turn off power.

Sequence Structures

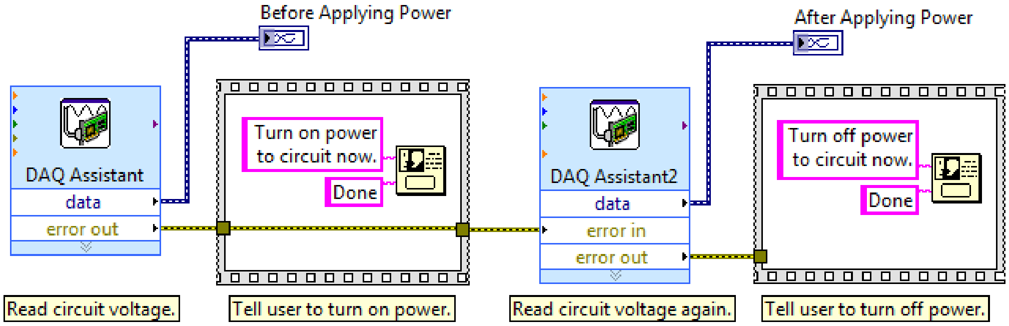
A Sequence structure contains one or more subdiagrams, or frames, that execute in sequential order; a frame cannot begin execution until everything in the previous frame has completed execution.


DAQ Assistant
data
error out
Read circuit voltage.
Before Applying Power
Turn on power to circuit now.
Done
Tell user to turn on power.
DAQ Assistant2
data
Read circuit voltage again.
After Applying Power
Turn off power to circuit now.
Done
Tell user to turn off power.

Avoid Overuse of Sequence Structures

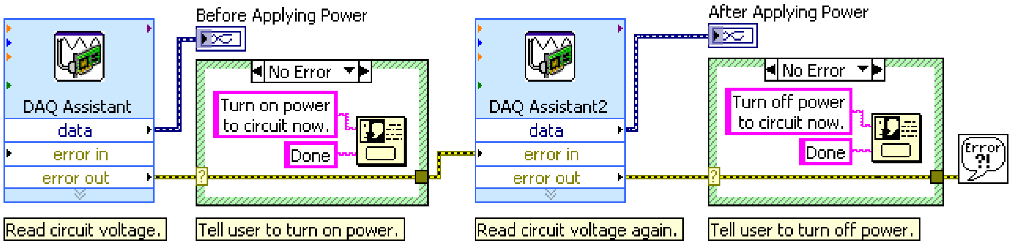
To take advantage of the inherent parallelism in LabVIEW, avoid overusing Sequence structures. Below are some caveats while using Sequence structures:

- Sequence structures guarantee the order of execution, but prohibit parallel operations.
- You cannot stop the execution part way through the sequence.
- Use Sequence structures sparingly because they do not enforce error checking and will continue to go through a sequence even after errors are detected.


Before Applying Power
DAQ Assistant
data
error out
Turn on power to circuit now.
Done
Read circuit voltage.
Tell user to turn on power.
After Applying Power
DAQ Assistant2
data
error in
error out
Turn off power to circuit now.
Done
Read circuit voltage again.
Tell user to turn off power.

Error Case Structure

Error Case structures rely on data flow rather than sequence structures to control the order of execution.



B. Using State Programming

Objective: Describe the functionality represented by a state transition diagram.

Why Use State Programming?

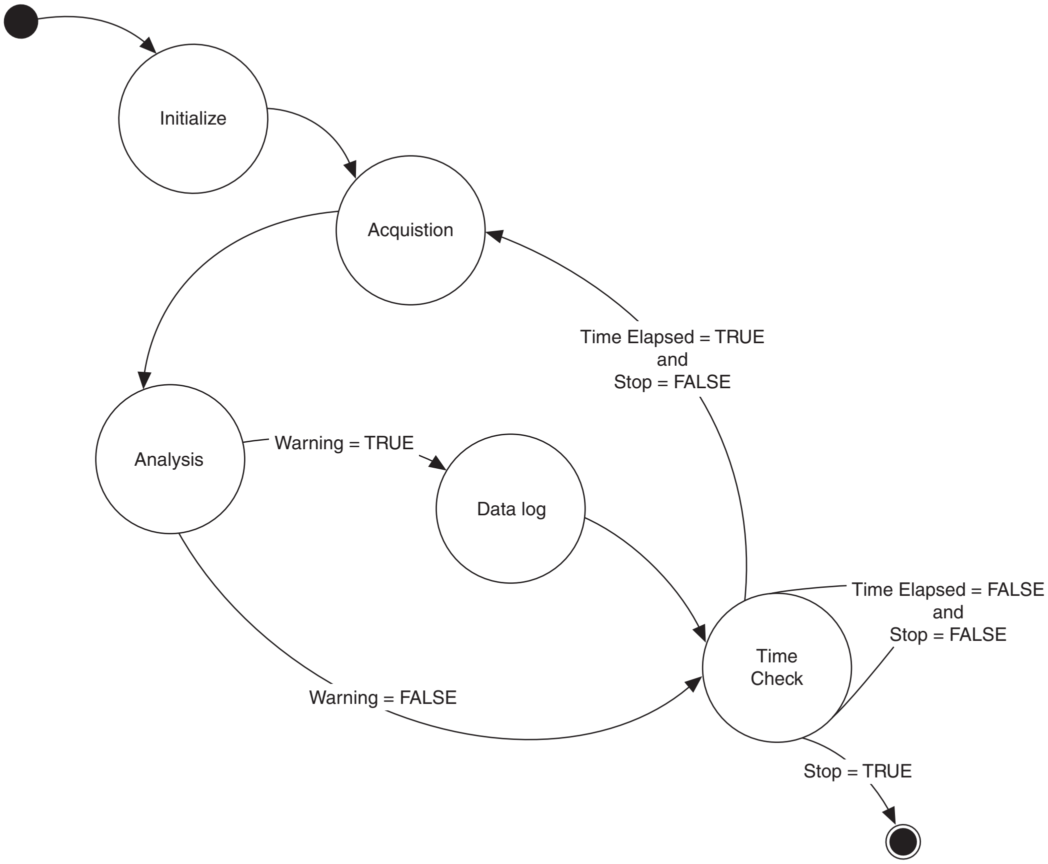
- Change the order of a sequence
- Repeat an item in sequence more often than others
- Execute only if certain conditions are met
- Stop the program immediately

State Transition Diagram

A state transition diagram is a type of flowchart that indicates the states of a program and transitions between states

State Part of a program that satisfies a condition, performs an action or waits for an event.

Transition Condition, action, or event that causes the program to move to the next state.

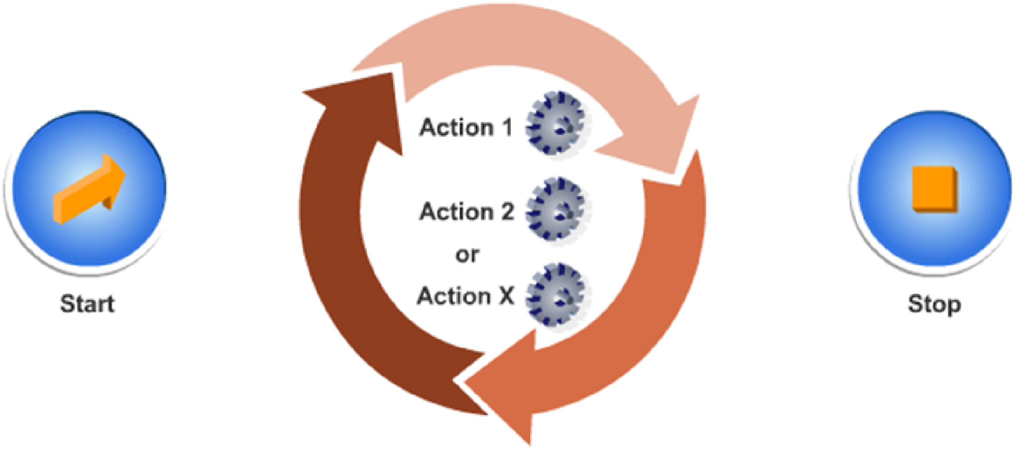


C. State Machines

Objective: Determine when to use a state machine.

What is a State Machine?

A state machine is a common and useful design pattern in LabVIEW that usually has a start-up and shut-down state, but also contains other states. State machines can implement any algorithm that can be explicitly described by a state diagram or flowchart.



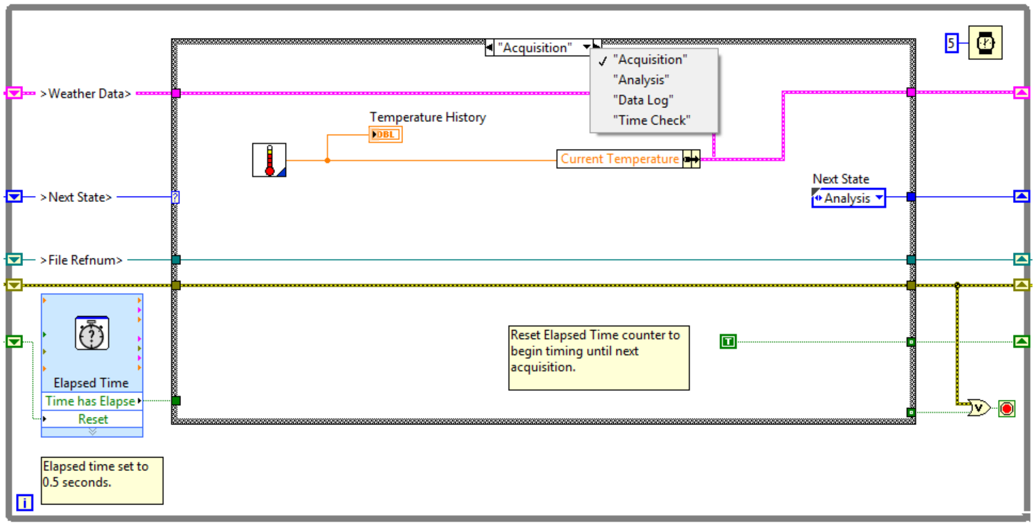
When to Use a State Machine

Use a state machine for the following applications:

- Sequential process
- UI-Driven process

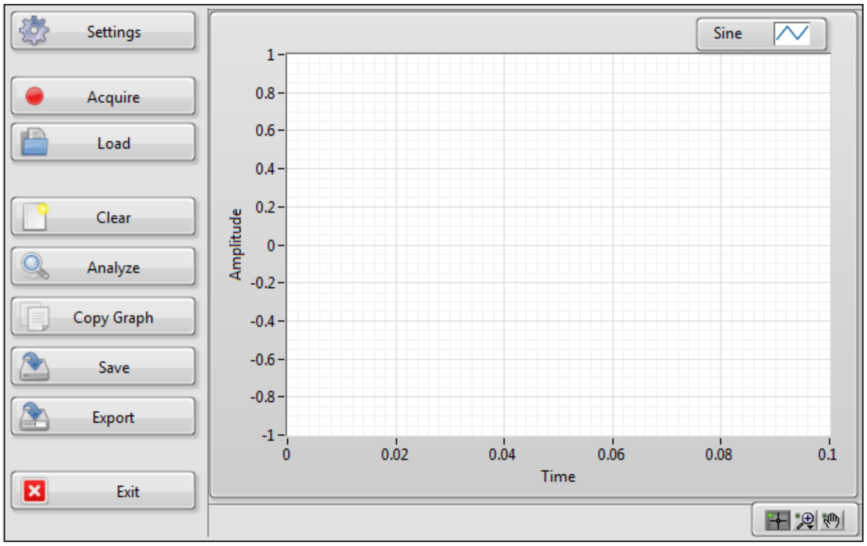
Sequential Process

- A state represents each segment of the process.
- Depending on the result of each state's test, a different state might be called.
- Can happen continually, resulting in an in-depth analysis of the process you are testing



UI-Driven Process

- Different user actions process different code and act as a state in the state machine.
- Each segment can lead to another segment for further processing or wait for another user action.
- The state machine constantly monitors the user for the next action to take.
- Applications require an initialization state, followed by a default state, where many different actions can be performed. The actions performed can depend on previous and current inputs and states. A shutdown state commonly performs clean up actions.



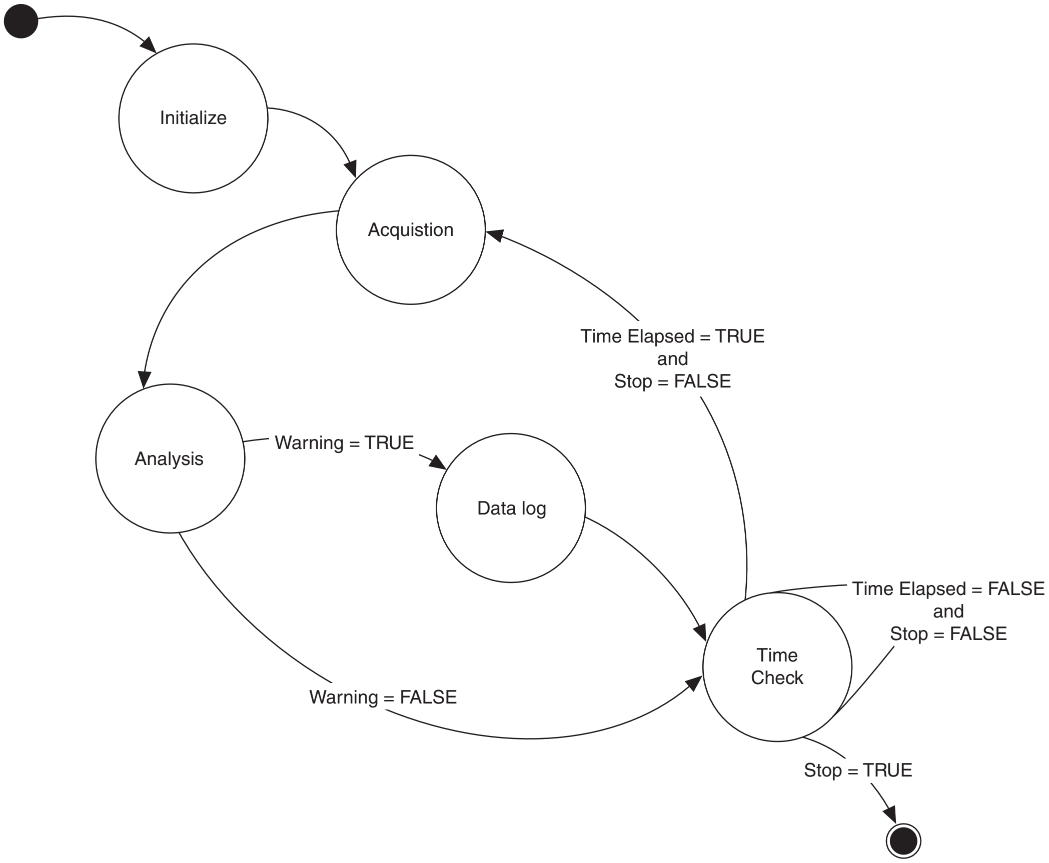
Multimedia: Building State Machines

Complete the multimedia module, *Building State Machines*, available in the `<Exercises>\LabVIEW Core 1\Multimedia\` folder to learn about the following topics.

- State Machine Infrastructure
- State Machine Transitions
  - Single Default
  - Select Function
  - Case Structure
  - Transition Array

Course Project Transition Diagram

Consider the following example. An application acquires a temperature every half second, analyzes each temperature to determine if the temperature is too high or too low, and alerts the user if there is a danger of heatstroke or freeze. The application logs the data if a warning occurs. If the user has not clicked the stop button, the entire process repeats. The following figure shows the state transition diagram for this temperature warning application.



Exercise 10-1 Weather Station Project

Goal

Create a VI that implements a state machine using a type-defined enum.

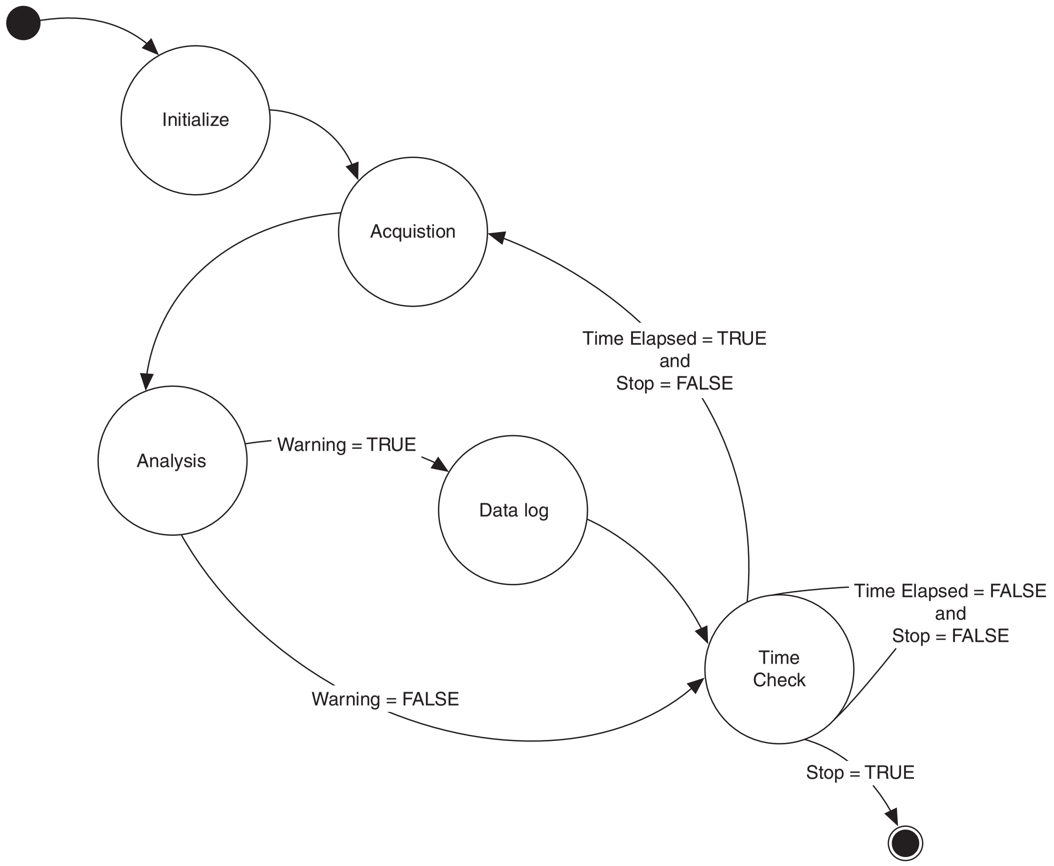
Scenario

You must design a VI for a user interface state machine. The VI acquires a temperature every half second, analyzes each temperature to determine if the temperature is too high or too low, and alerts the user if there is a danger of heatstroke or freeze. The program logs the data if a warning occurs. If the user has not clicked the stop button, the entire process repeats. The state machine must also allow for expansion, because processes may be added in the future.

Design

Use a flowchart and states list to create the VI in this exercise. The flowchart in Figure 10-1 illustrates the data flow for this design.

Figure 10-1. Temperature Warnings VI Flowchart



The following table describes the states in this state machine.

State	Description	Next State
Acquisition	Set time to zero, acquire data from the temperature sensor	Analysis
Analysis	Read front panel controls and determine warning level	Data Log if a warning occurs Time Check if no warning occurs
Data Log	Log the data in a tab-delimited ASCII file	Time Check
Time Check	Check whether time is greater than or equal to .5 seconds	Acquisition if time has elapsed Time Check if time has not elapsed

Implementation

1. Open `Weather Station.lvproj` in the `<Exercises>\LabVIEW Core 1\Weather Station` directory.

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

Figure 10-2 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.

Figure 10-2. Weather Station UI VI Front Panel Window

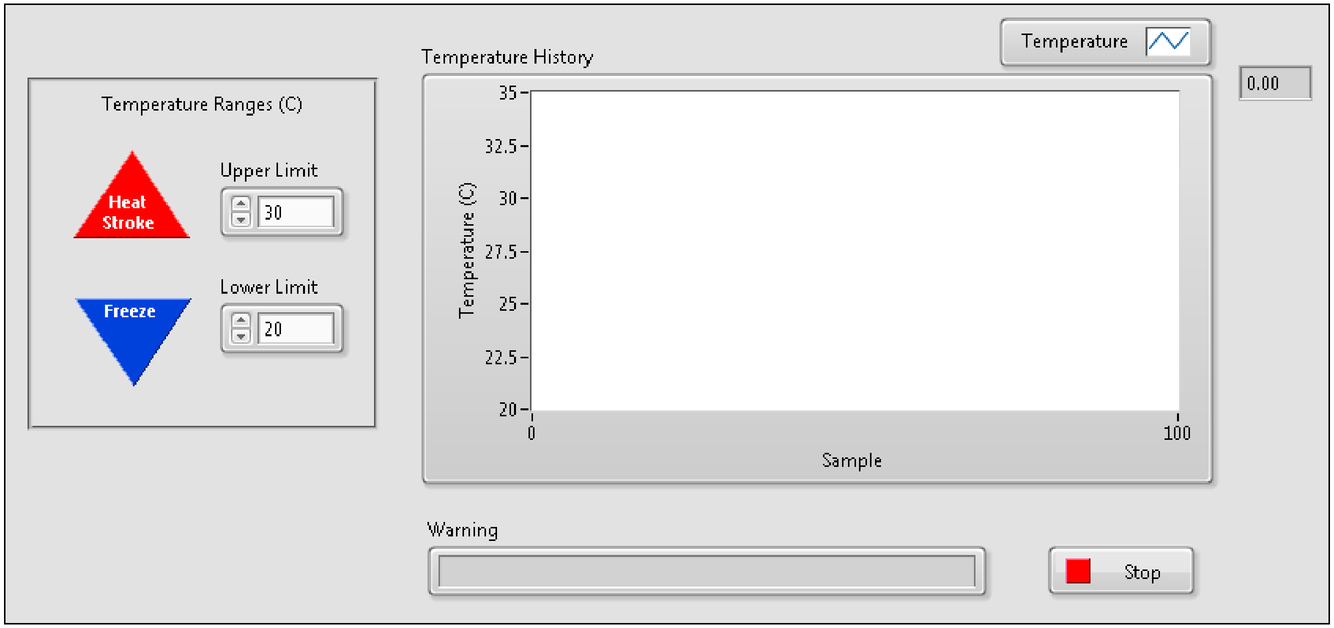
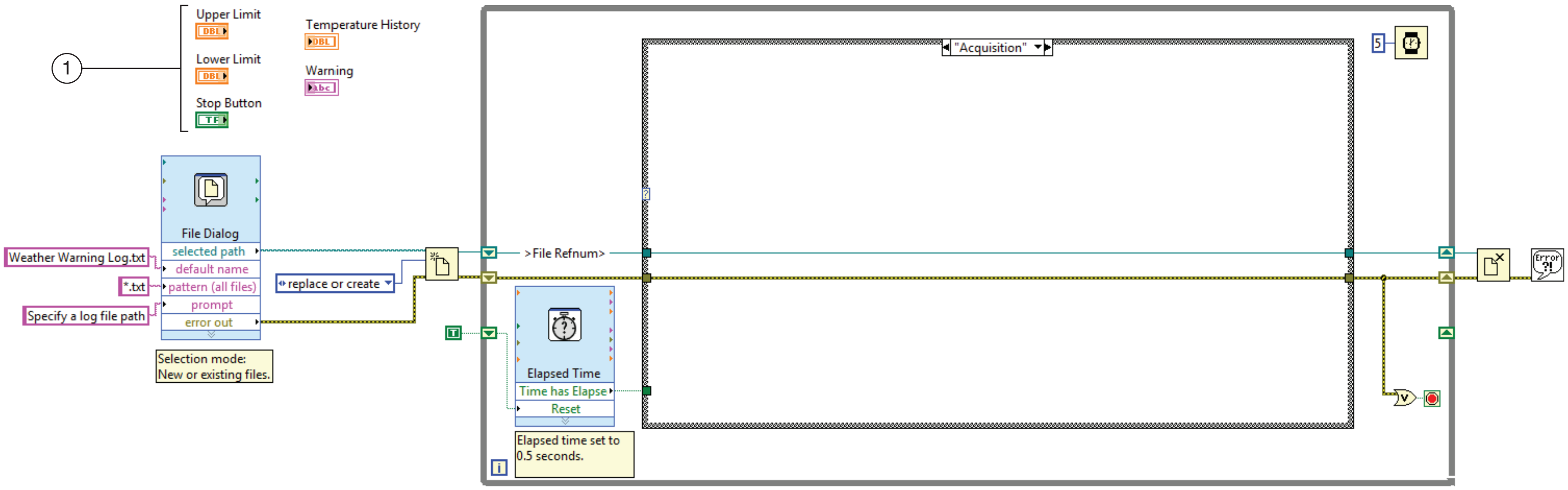


Figure 10-3 shows the starting point of the block diagram for the Weather Station UI VI. You edit this block diagram to implement a state machine for the Weather Station application.

Figure 10-3. Weather Station UI VI Block Diagram Starting



1 You use these controls and indicators to program different cases.

3. Create a new type definition to control the Weather Station application.

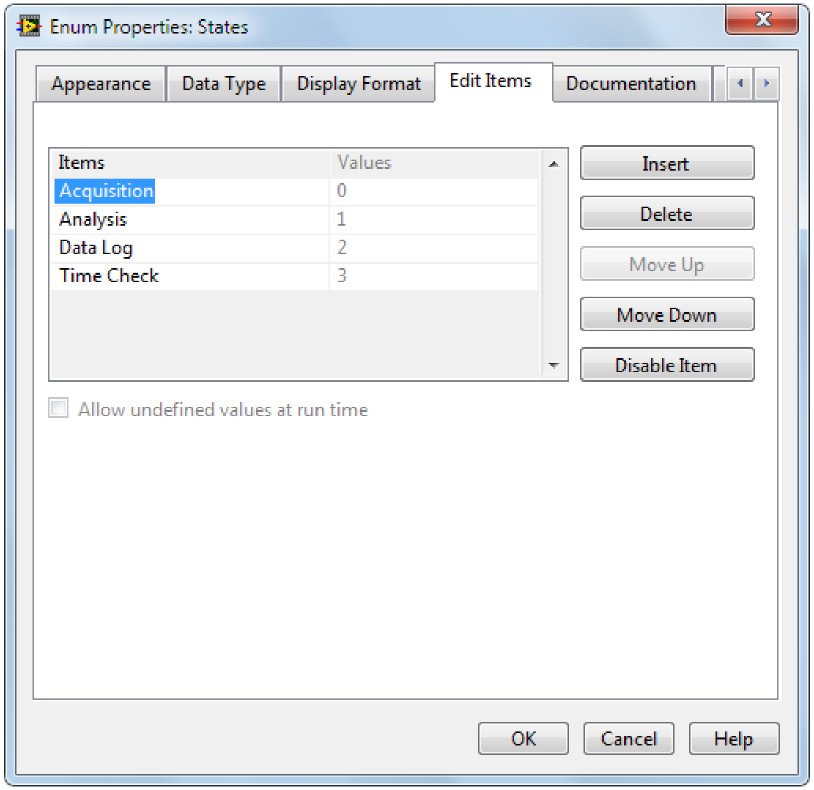
   □ Open the block diagram and create an Enum constant to the left of the While Loop.

   □ Type `Acquisition` in the constant.

   □ Right-click the constant and select **Edit Items** from the shortcut menu.

   □ Add the items shown in Figure 10-4 and click **OK**.

Figure 10-4. Weather Station States Type Def



Enum Properties: States

Items	Values
Acquisition	0
Analysis	1
Data Log	2
Time Check	3

□ Allow undefined values at run time

   □ 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**.

   □ Change the label on the **Enum** control to `States`.

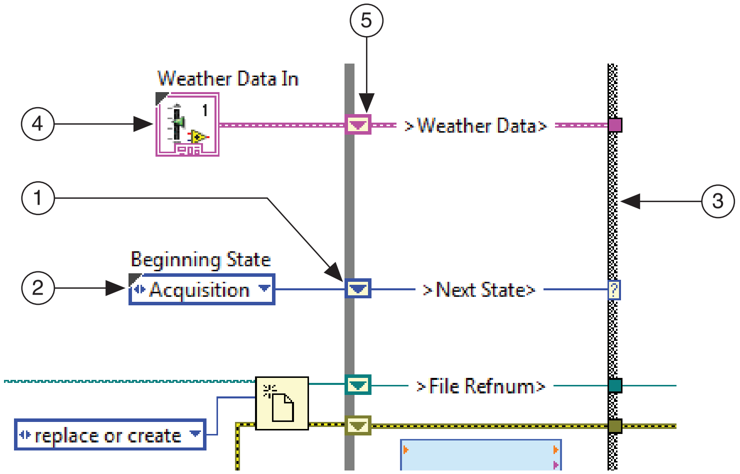
   □ Save the type definition as `Weather Station States.ctl` in the `<Exercises>\LabVIEW Core 1\Weather Station\Supporting Files` directory.

   □ Close the control editor window.

   □ In the **Project Explorer** window, notice that **Weather Station States.ctl** has been added to your **Supporting Files** folder because that folder is an auto-populating folder.

5. Control the state machine with the type-defined enum and update the framework as shown in Figure 10-5.

Figure 10-5. Weather Station UI VI Block Diagram

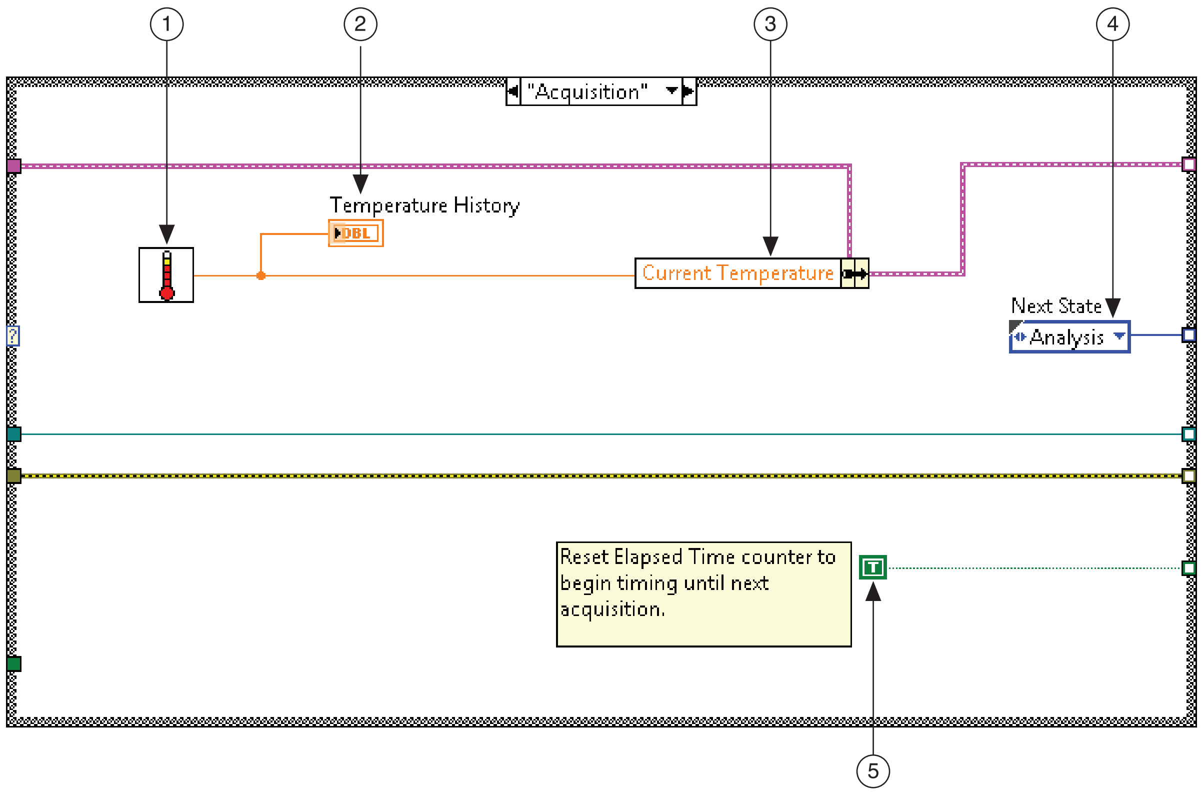


1 **Shift Register**—Right-click the While Loop and select **Add Shift Register**.
2 Enum type definition constant—Right-click and select **Visible Items»Label**. Change the label to `Beginning State`. Wire the **Beginning State** constant to the shift register to initialize the shift register to the Acquisition state. Wire the shift register to the case selector 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 In**—Drag **Weather Data.ctl** from the **Project Explorer** window to the block diagram to create a type definition cluster constant. Right-click the cluster and select **View Cluster As Icon**.
5 **Shift Register**—Place a shift register on the While Loop and wire the **Weather Data In** constant to it.

Note After you finish wiring the Acquisition case in step 6, some tunnels are empty because not all cases are wired yet.

6. Complete the Acquisition state shown in Figure 10-6.

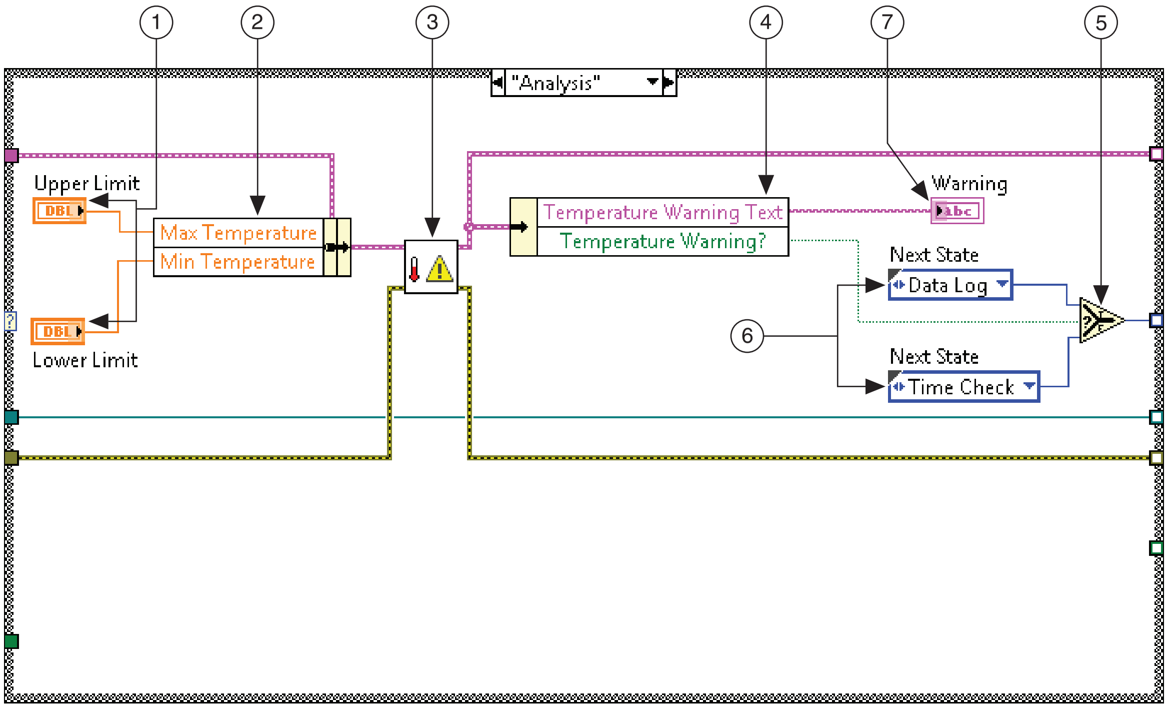
Figure 10-6. Weather Station UI VI Acquisition State



1 **Thermometer** or **Thermometer (Demo)**—Drag one of these VIs from the **Shared Files** folder in the **Project Explorer** window to the block diagram. Drag the Thermometer VI if you have hardware and drag the Thermometer (Demo) VI if you do not have hardware.
2 **Temperature History**—Move this indicator into the Acquisition state of the Case structure.
3 **Bundle By Name**—Wire the **Temperature Value** output of the Thermometer VI to the **Current Temperature** input.
4 Next State enum—<Ctrl>-click the **Beginning State** enum and drag a copy into the Acquisition case. Rename this copy of the Weather Station States type definition `Next State`. Set the enum to **Analysis** 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 Acquisition case.

7. Complete the Analysis case as shown in Figure 10-7.

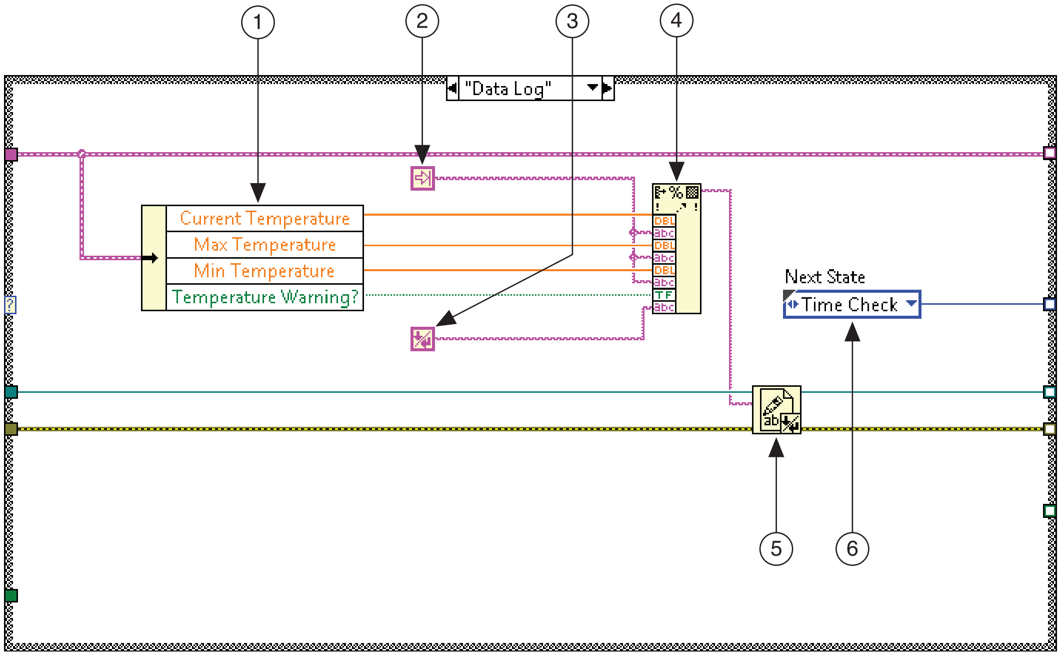
Figure 10-7. Weather Station UI VI—Analysis Case



1 **Upper Limit** and **Lower Limit**—Move these controls from outside the While Loop.
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 Temperature Warnings VI.
3 **Temperature Warnings**—Drag the Temperature Warnings VI from the **Supporting Files** folder in the **Project Explorer** window.
4 **Unbundle By Name**—Returns the value of specific items from the cluster.
5 **Select**—Determines which state to execute next depending on whether or not a warning occurs.
6 Weather Station States—Wire two copies of the Weather Station States type definition to the Select function. You can create these copies from the **Beginning State** enum.
7 **Warning**—Move this indicator from outside the While Loop.

8. Complete the Data Log case as shown in Figure 10-8.

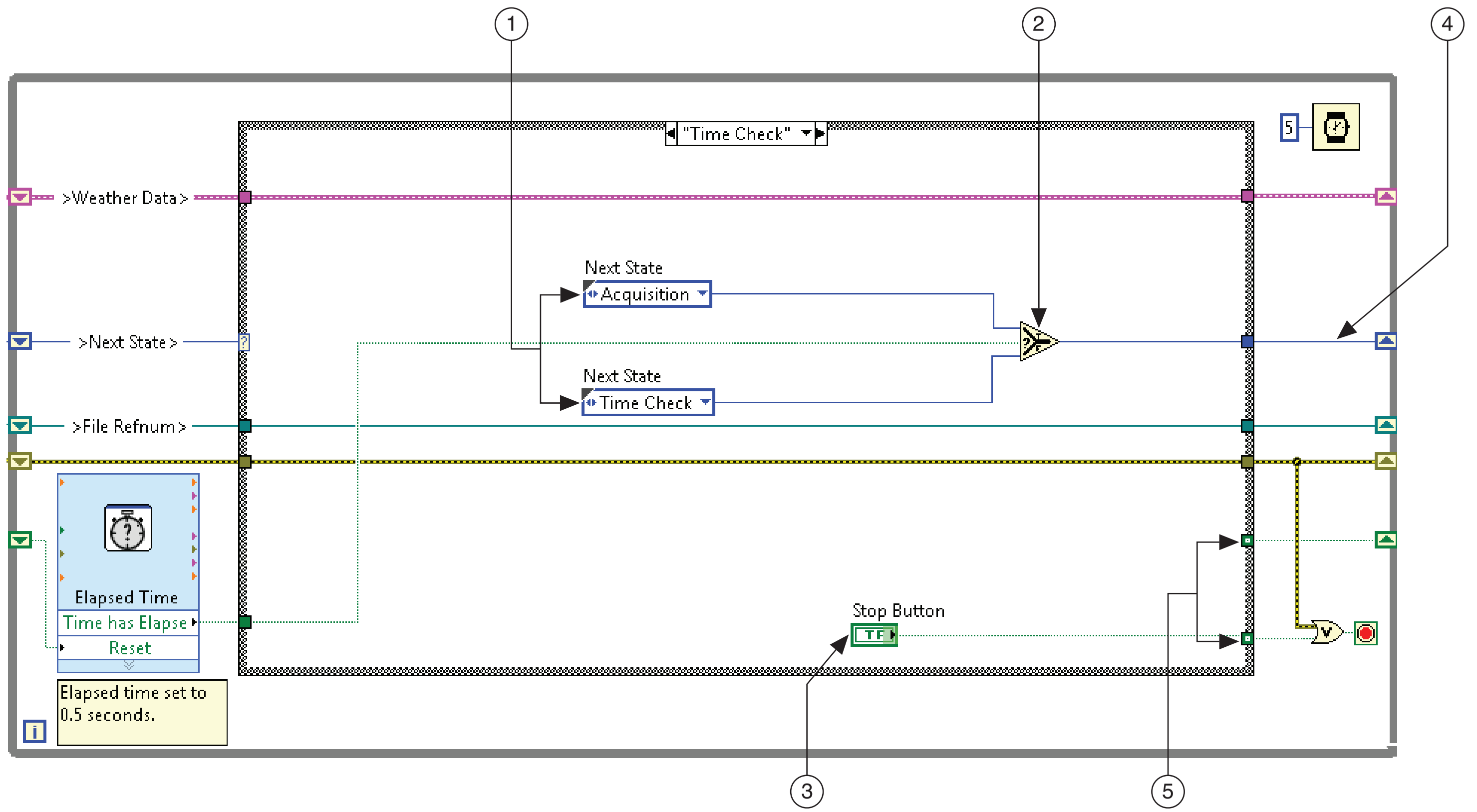
Figure 10-8. Weather Station UI VI—Data Log Case



1 **Unbundle By Name**—Returns the value of specific items from the cluster.
2 **Tab Constant**—Inserts a tab in the string. The log file you create contains tabs between values.
3 **End of Line Constant**—Inserts a platform-specific end-of-line value at the end of the string. The log file you create uses the End of Line constant to insert line breaks between data.
4 **Format Into String**—Expand the node to accept eight inputs.
5 **Write to Text File**—Writes the text that you formatted into a log file.
6 Next State—Create a copy of the Weather Station States enum, label it `Next State`, and set the next state to **Time Check**.

9. Complete the Time Check case as shown in Figure 10-9.

Figure 10-9. Weather Station UI VI—Time Check Case



1 Next State—Wire two copies of the Weather Station States type definition to the Select function.
2 **Select**—Determines which state to execute next depending on whether or not time has elapsed.
3 **Stop Button**—Move the **Stop Button** terminal from outside the While Loop. Wire the **Stop Button** terminal to the Or function outside of the Case structure.
4 Next State wire—Wire the tunnel for the Next State wire to the shift register.
5 Use default if unwired—Right-click these tunnels and select **Use Default If Unwired**.

10. Save and test the VI.

Test

1. Run the VI.

   □ Name the log file when prompted.

   □ Enter values for the **Upper Limit** and **Lower Limit** controls and observe the behavior of the VI. Does it behave as expected?

2. Stop the VI.

3. Navigate to the `Weather Warning Log.txt` file and open it.

4. Notice the changes in the upper and lower limit values and the placement of tabs and line breaks.

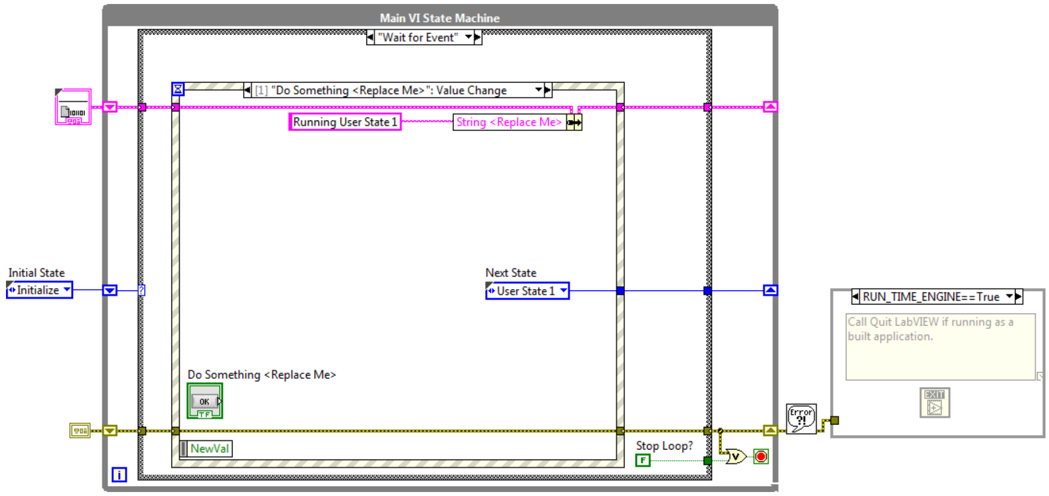
5. Close the log file.

6. Save and close the VI and the project.

End of Exercise 10-1

Event-Based State Machine

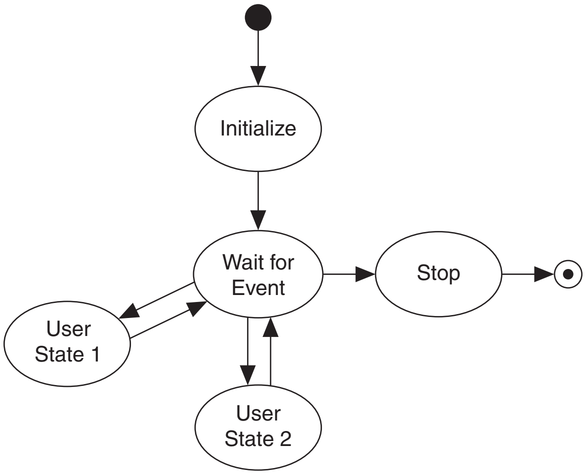
Event-based state machines combine a user interface event handler with the transition ability of the state machine. Event-based state machines include a “Wait on Event” case to process user-interface events.



Demonstration: Simple State Machine Project Template

LabVIEW provides the Simple State Machine project template to simplify the process of creating an application that uses the event-based state machine design pattern.

- The Simple State Machine template is a customizable application which is in the form of a `.lvproj` file with supporting VIs and type definition controls.
- The application is based on the event-based state machine design pattern.



Additional Resources

Learn More About	LabVIEW Help Topic or ni.com
Sequential Design	*Sequence Structures: Executing Sections of Code Sequentially*
State Machines	*Creating VIs from Templates and Sample Projects* *Examples, VI Templates, Project Templates, and Sample Projects*
Events	*Events in LabVIEW* *Event Structure* *Caveats and Recommendations when Using Events in LabVIEW*

Activity 10-1: Lesson Review

1. When using a Sequence structure, you can stop the execution in the middle of a sequence.
   a. True
   b. False

2. Which of the following are benefits of using a state machine instead of a sequential structure?
   a. You can change the order of the sequence.
   b. You can repeat individual items in the sequence.
   c. You can set conditions to determine when an item in the sequence should execute.
   d. You can stop the program at any point in the sequence.

Activity 10-1: Lesson Review - Answers

1. When using a Sequence structure, you can stop the execution in the middle of a sequence.

   a. True

   b. False

   You cannot stop the execution in the middle of a sequence.

2. Which of the following are benefits of using a state machine instead of a sequential structure?

   a. You can change the order of the sequence.

   b. You can repeat individual items in the sequence.

   c. You can set conditions to determine when an item in the sequence should execute.

   d. You can stop the program at any point in the sequence.

A 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 onsite 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, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, and instrument drivers. 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.