LabVIEW™ Core 1 Participant Guide

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

Trademarks

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

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

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

TETRIX by Pitsco is a trademark of Pitsco, Inc.

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

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

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

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

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

Xilinx is the registered trademark of Xilinx, Inc.

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

FireWire® is the registered trademark of Apple Inc.

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

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

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

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

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

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

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

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

Patents

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

Worldwide Technical Support and Product Information

ni.com

Worldwide Offices

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

National Instruments Corporate Headquarters

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

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

Table of Contents

Appendix A
Additional Information and Resources

Student Guide

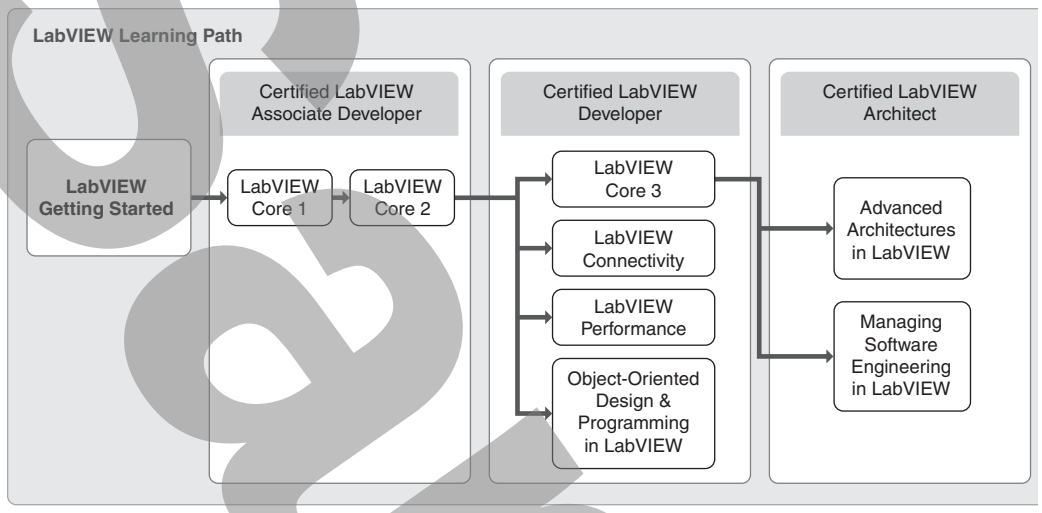
In this student guide, you will learn about the LabVIEW Learning Path, the course description, and the items you need to get started in the LabVIEW Core 1 course.

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.



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

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

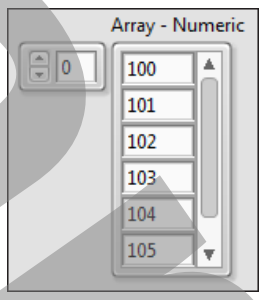
Objective: Identify when to use arrays and learn how to create and initialize arrays.

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.

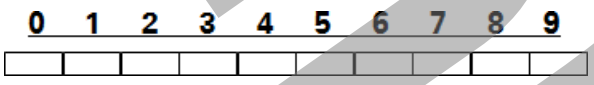


Note Array indexes in LabVIEW are zero-based. The index of the first element in the array, regardless of its dimension, is zero.

1D and 2D Examples

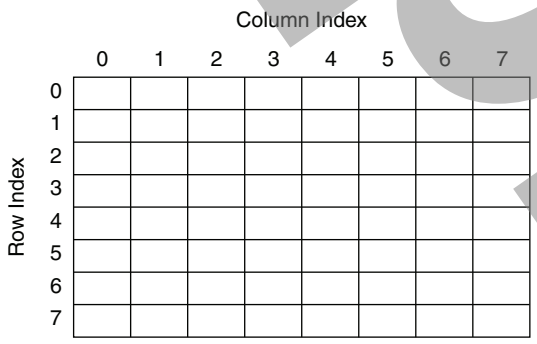
Arrays can have multiple dimensions.

- 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:



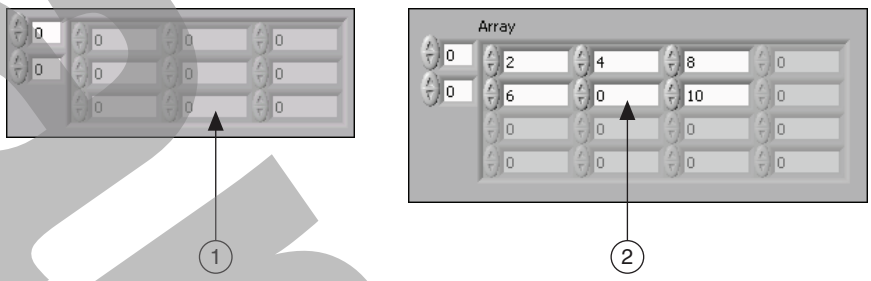
For example, LabVIEW represents a table of data with rows and columns as a 2D array.

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

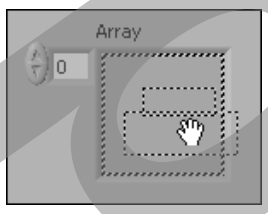
An uninitialized array contains a fixed number of dimensions but no elements. An initialized defines the number of elements in each dimension and the contents of each element.



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.

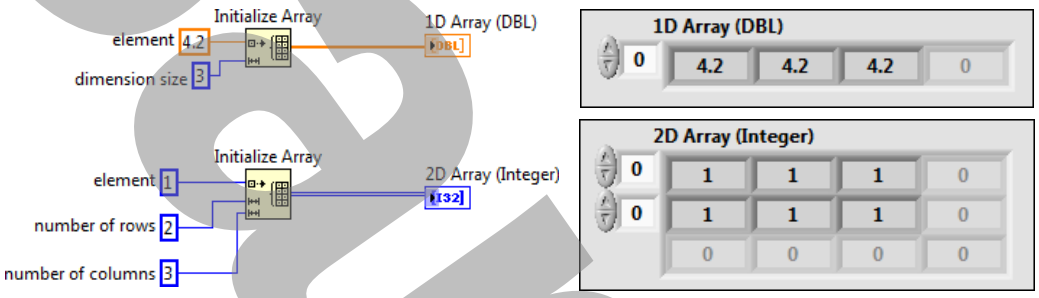
Multimedia: Common Array Functions

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.

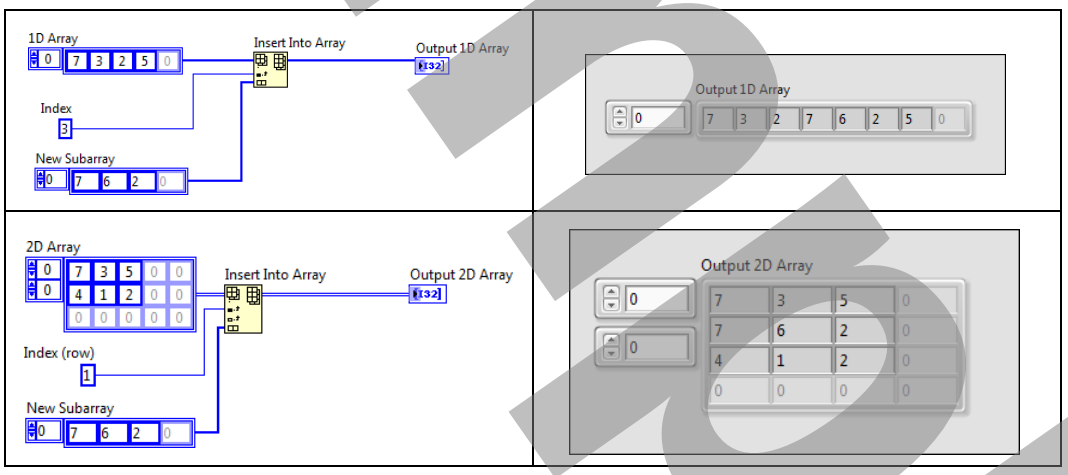
Initialize Array

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



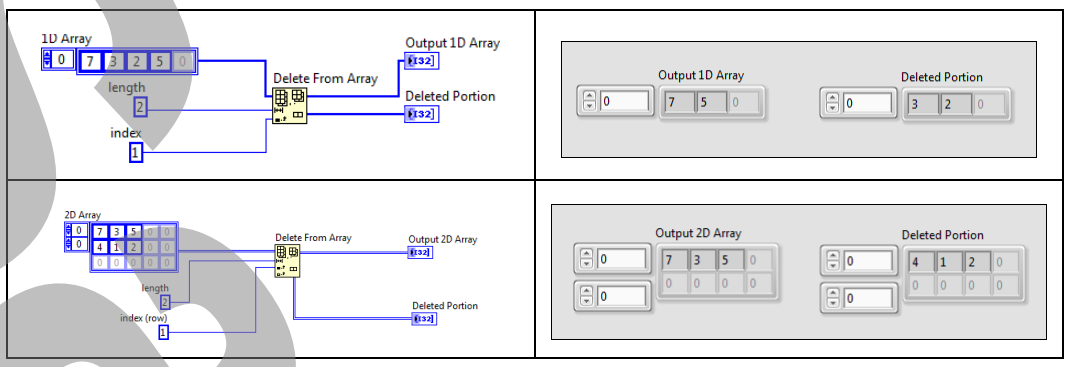
Insert Into Array

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



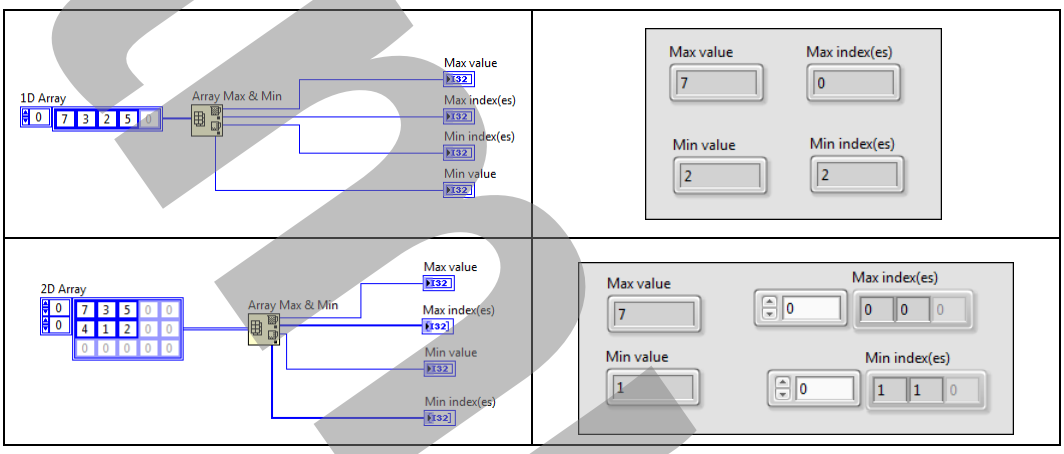
Delete From Array

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**.



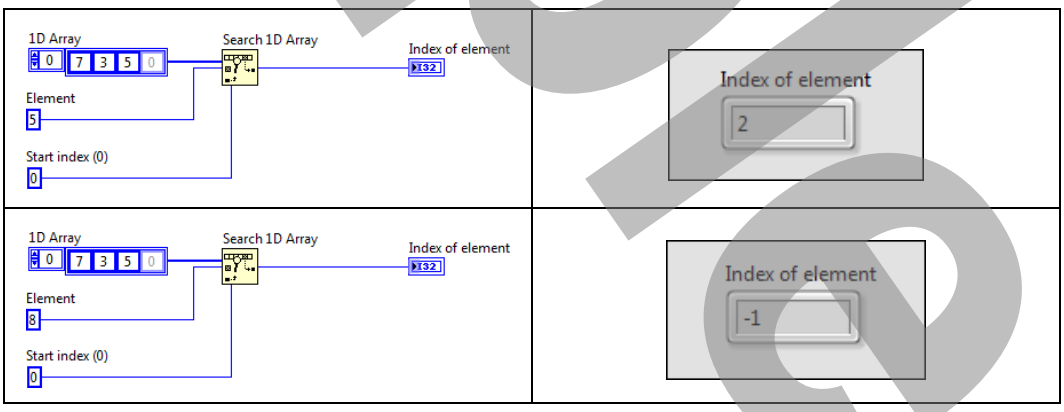
Array Max & Min

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



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.

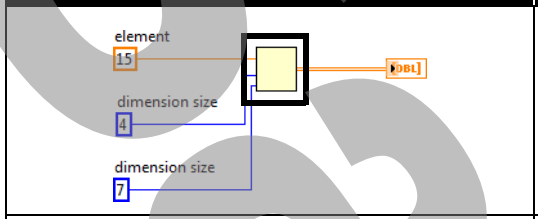
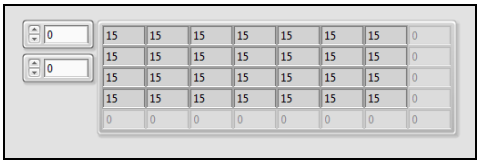
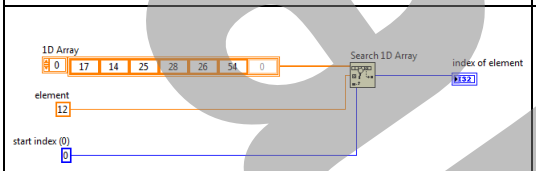
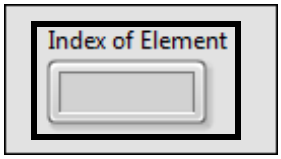
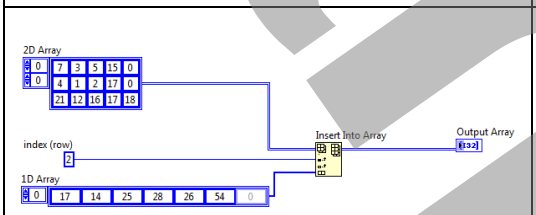
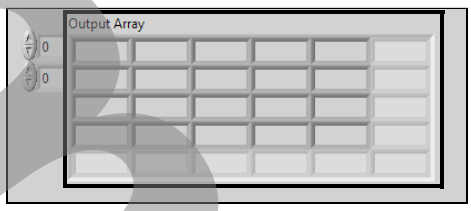
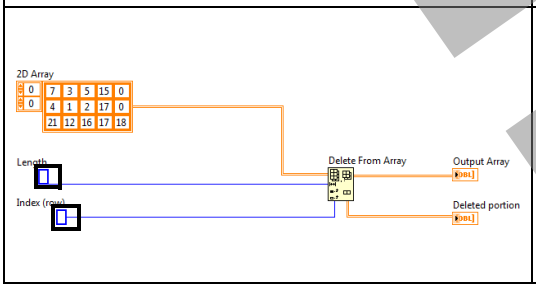
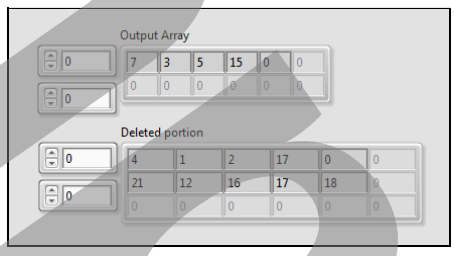


Activity 5-1: Using Array Functions

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
	
	
	
	

C. Polymorphism

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

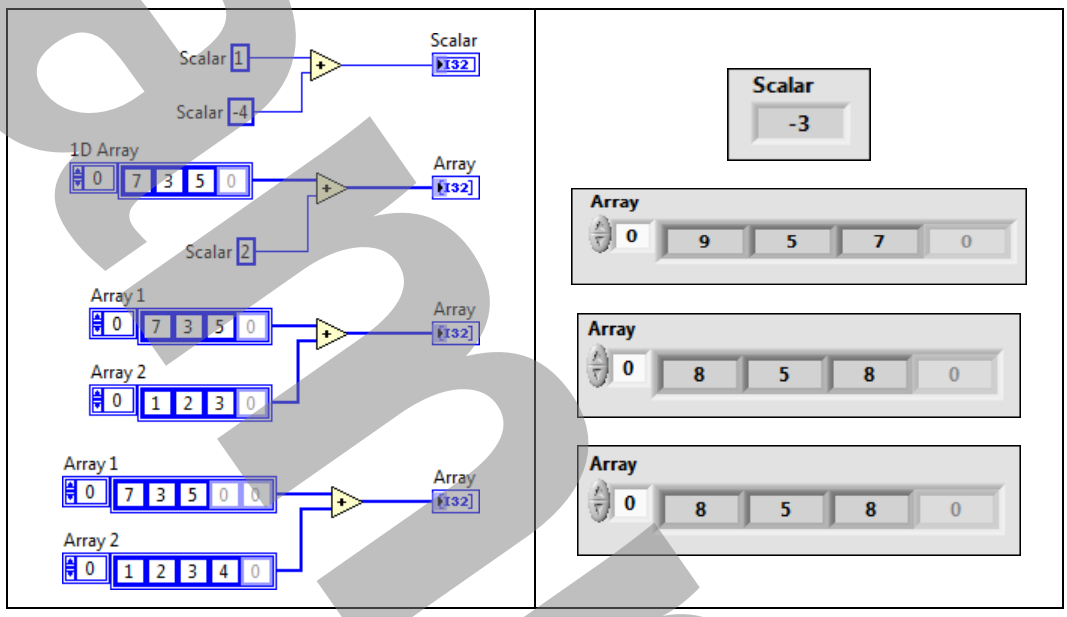
Polymorphism

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.



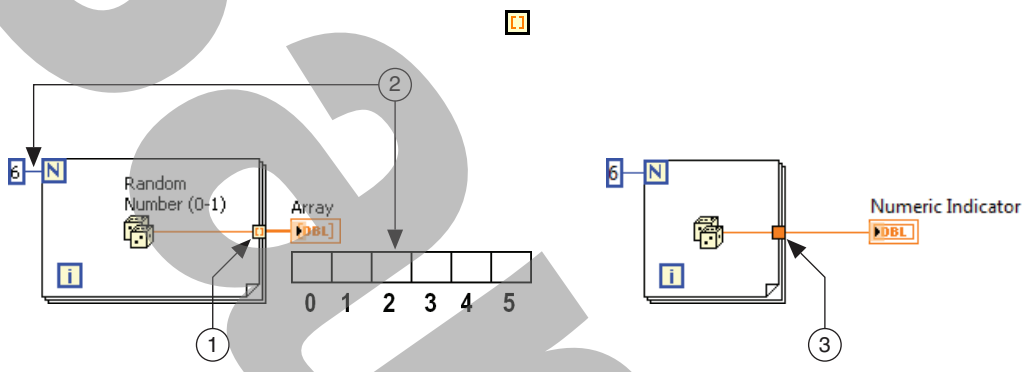
D. Auto-Indexing

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

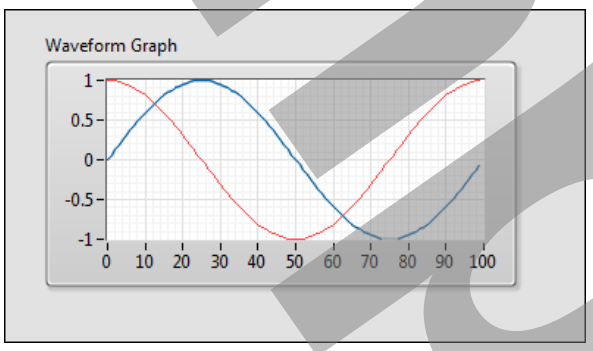
If you wire an array to or from a For Loop or While Loop, you can link each iteration of the loop to an element in that array by enabling auto-indexing. The tunnel image changes from a solid square to the image to indicate auto-indexing.

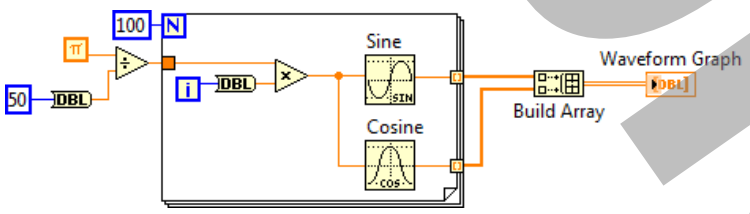


1 Right-click the tunnel and select **Enable Indexing** or **Disable Indexing** to toggle the state of the tunnel.
2 Auto-indexed output arrays are always equal in size to the number of iterations.
3 Only one value (the last iteration) is passed out of the loop when auto-indexing is disabled.

Waveform Graphs

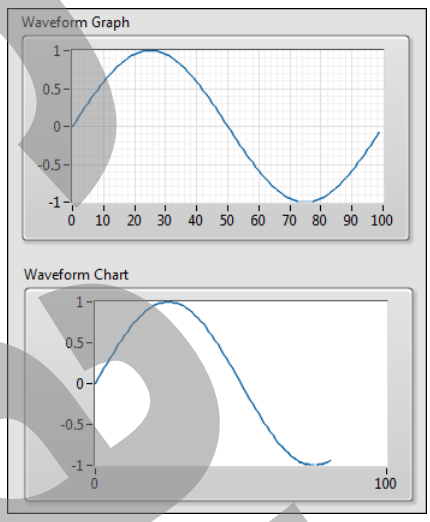
A waveform graph collects the data in an array and then plots the data to the graph.

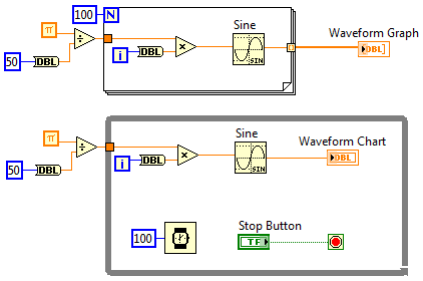




Charts vs. Graphs—Single-Plot

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

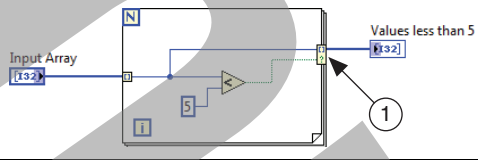




Auto-Indexing with a Conditional Tunnel

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

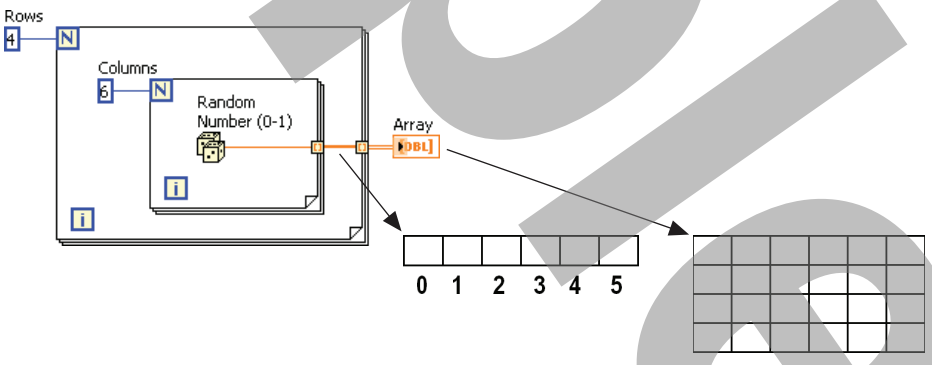
For example, consider the following block diagram. The array **Input Array** contains the following elements: 7, 2, 0, 3, 1, 9, 5, and 7. Because of the conditional tunnel, the **Values less than 5** array contains only the elements 2, 0, 3, and 1 after this loop completes all iterations.



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

Creating Two-Dimensional Arrays

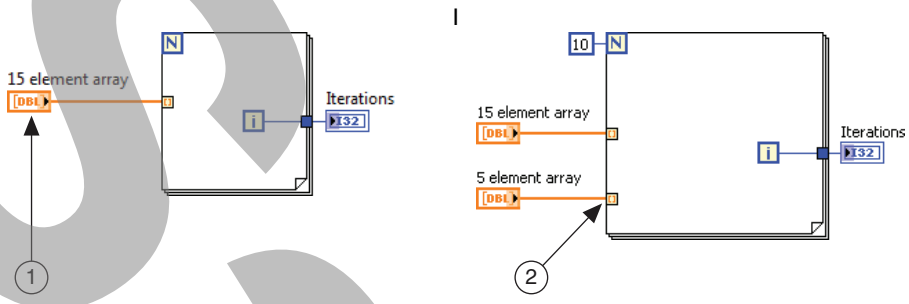
Use two For Loops, nested one inside the other, to create a 2D array.



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

Auto-Indexing Input

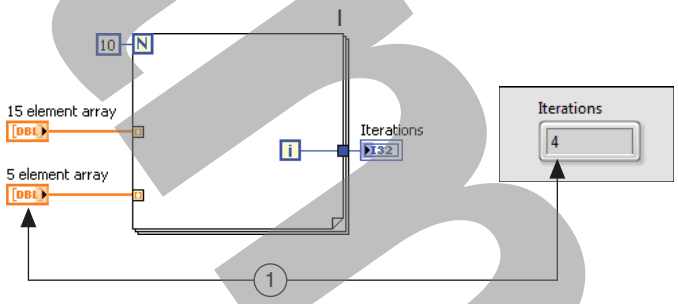
Use an auto-indexing input array to perform calculations on each element in an array. If you wire an array to an auto-indexing tunnel on a For Loop, you do not need to wire the count (N) terminal.



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.

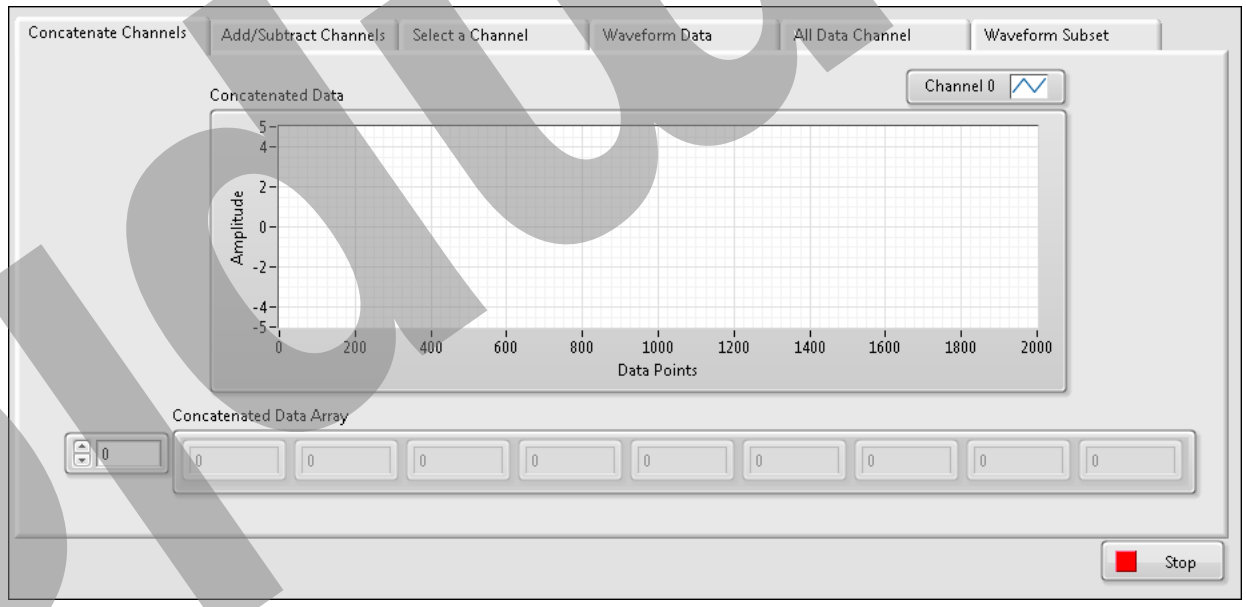
Description

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

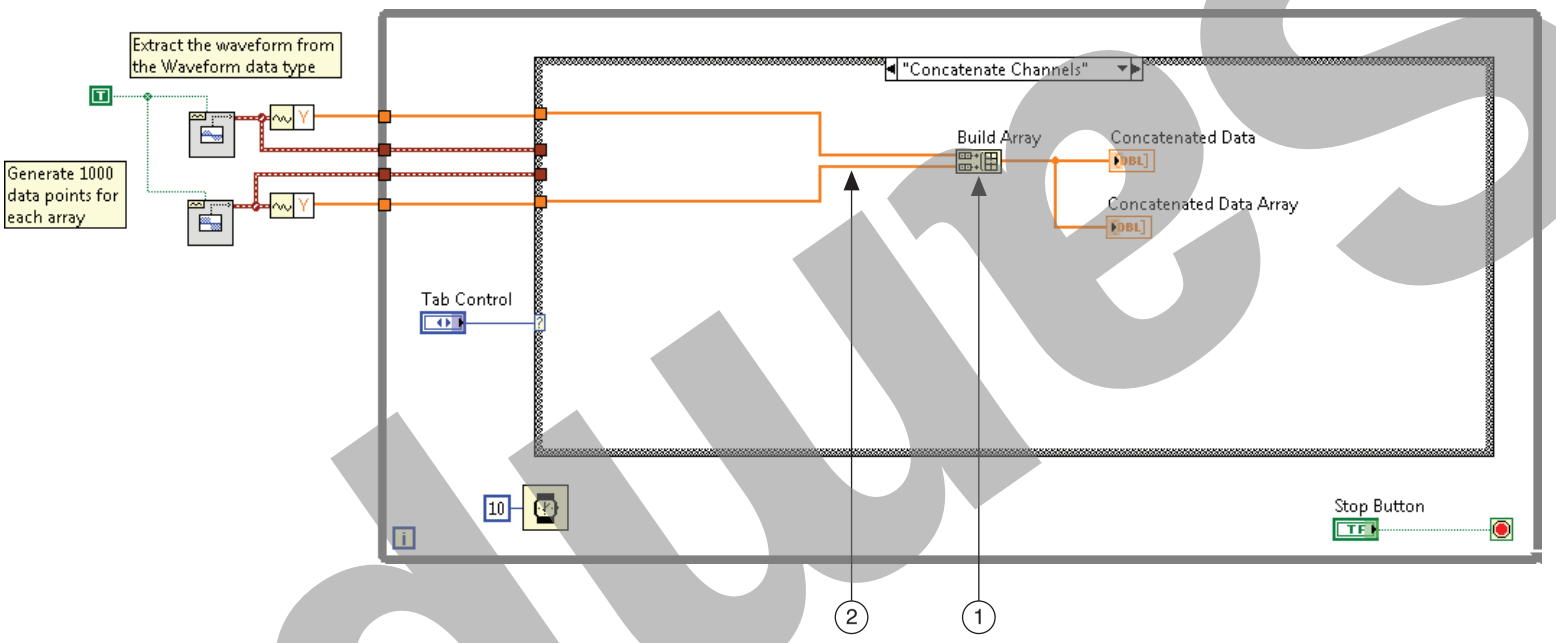
1. Open `Manipulating Arrays.lvproj` in the `<Exercises>\LabVIEW Core 1\Manipulating Arrays` directory.
2. Open **Array Manipulation** VI from the **Project Explorer** window. The front panel, shown in Figure 5-1, is already built for you.

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.

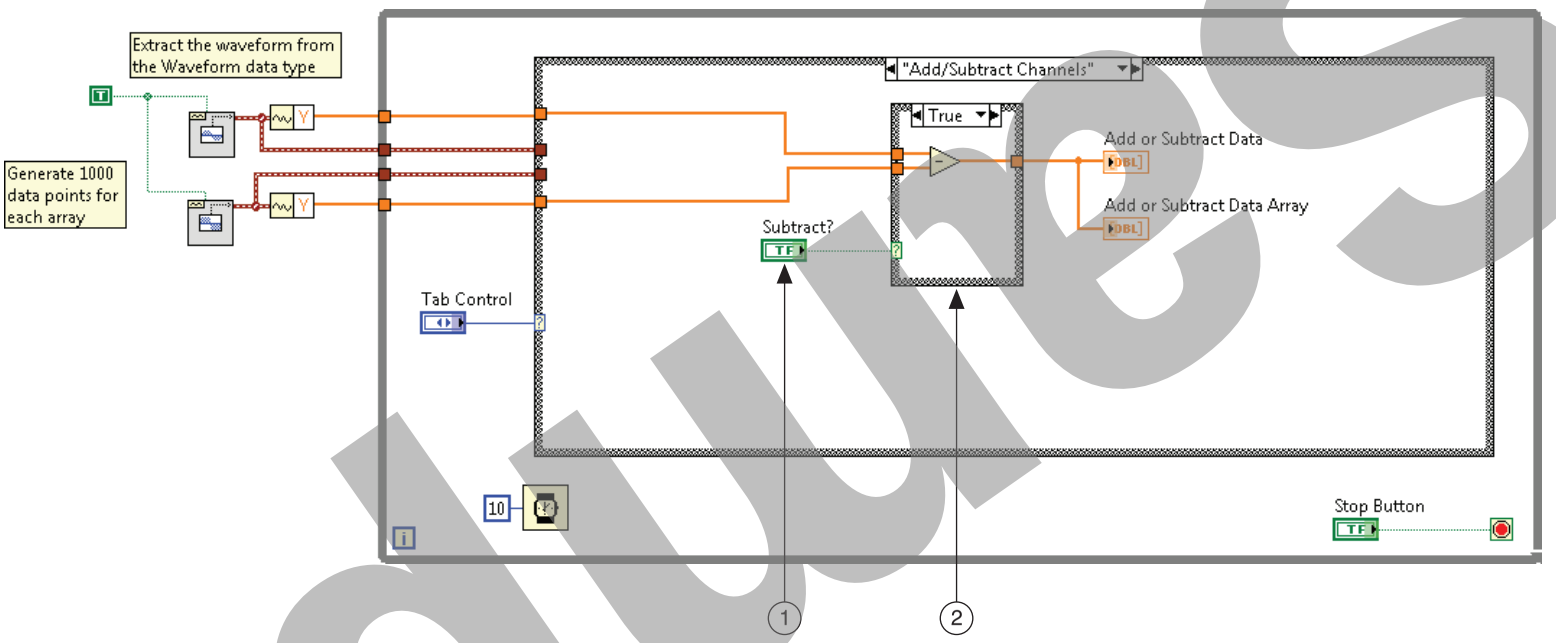
   ☐ Run the VI and notice that the sine wave is concatenated with a square wave.

5. Stop the VI.

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

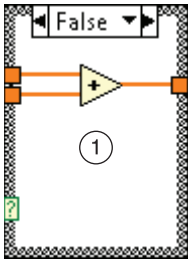
7. Complete the Add/Subtract Channels case as shown in Figure 5-3 and Figure 5-4.

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



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

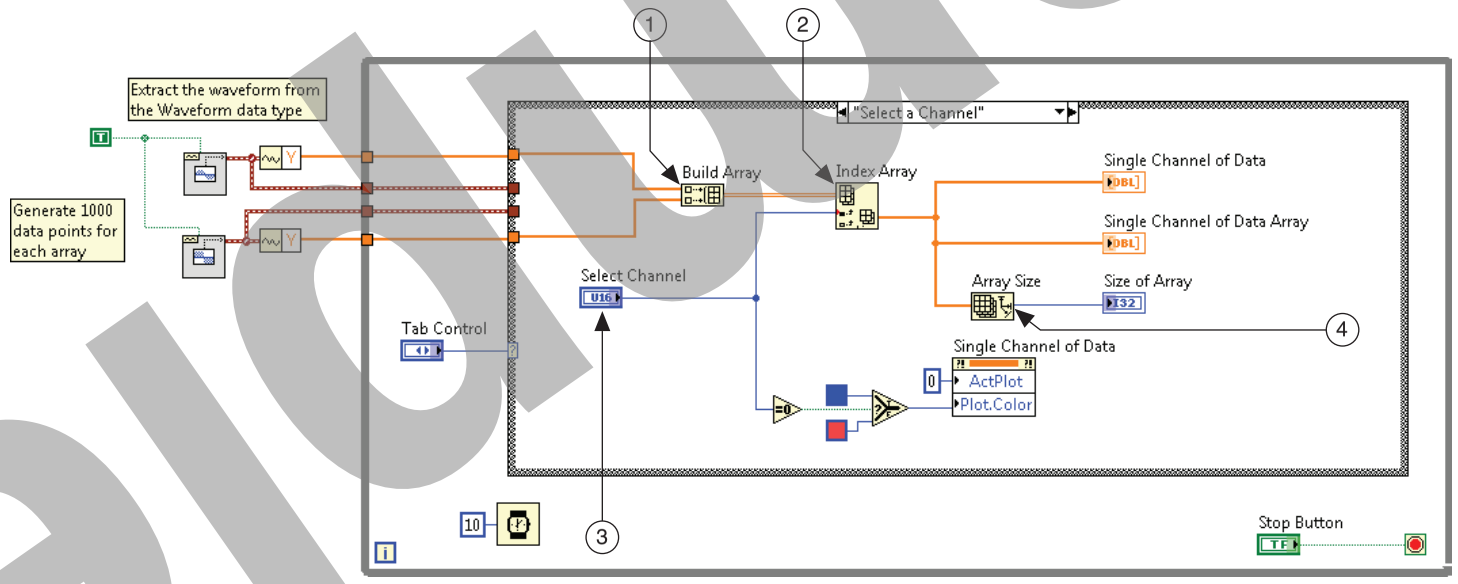


1 When the value of the Subtract? Boolean control is False, the array elements are added.

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



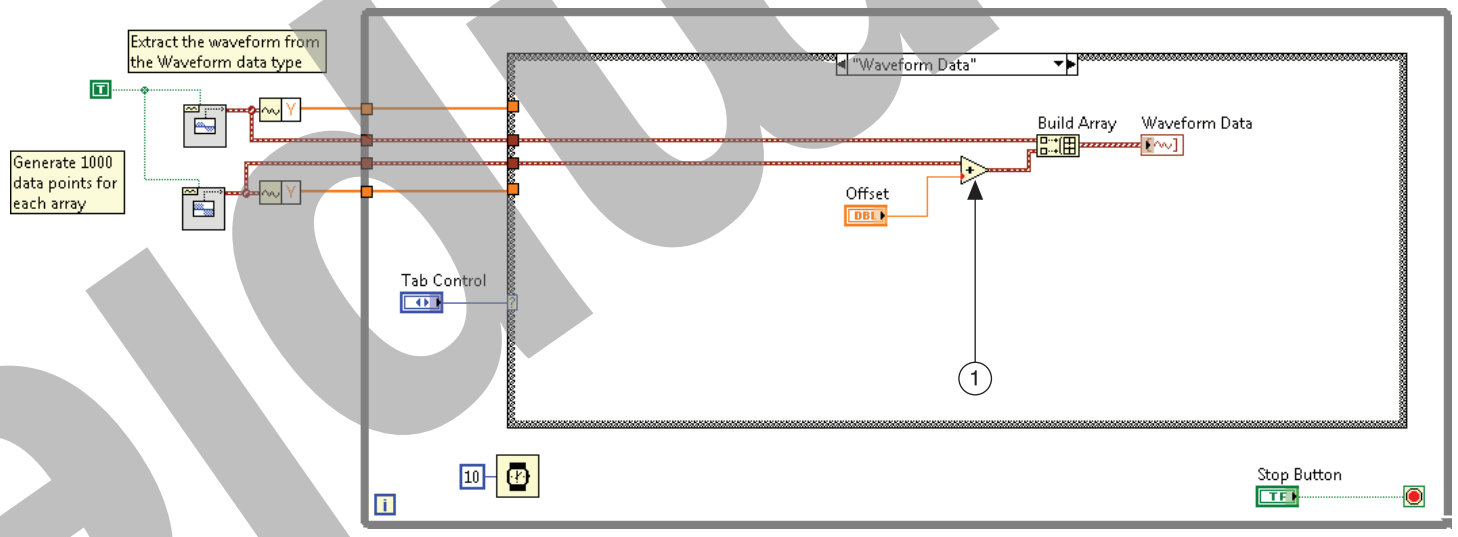
1 **Build Array**—Combines the sine and square waves into one 2D array.
2 **Index Array**—Extracts row 0 or 1 from the 2D array. The output from this function is a 1D array and is the waveform you select with the **Select Channel** control. The waveform is displayed on the **Single Channel of Data** 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



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.

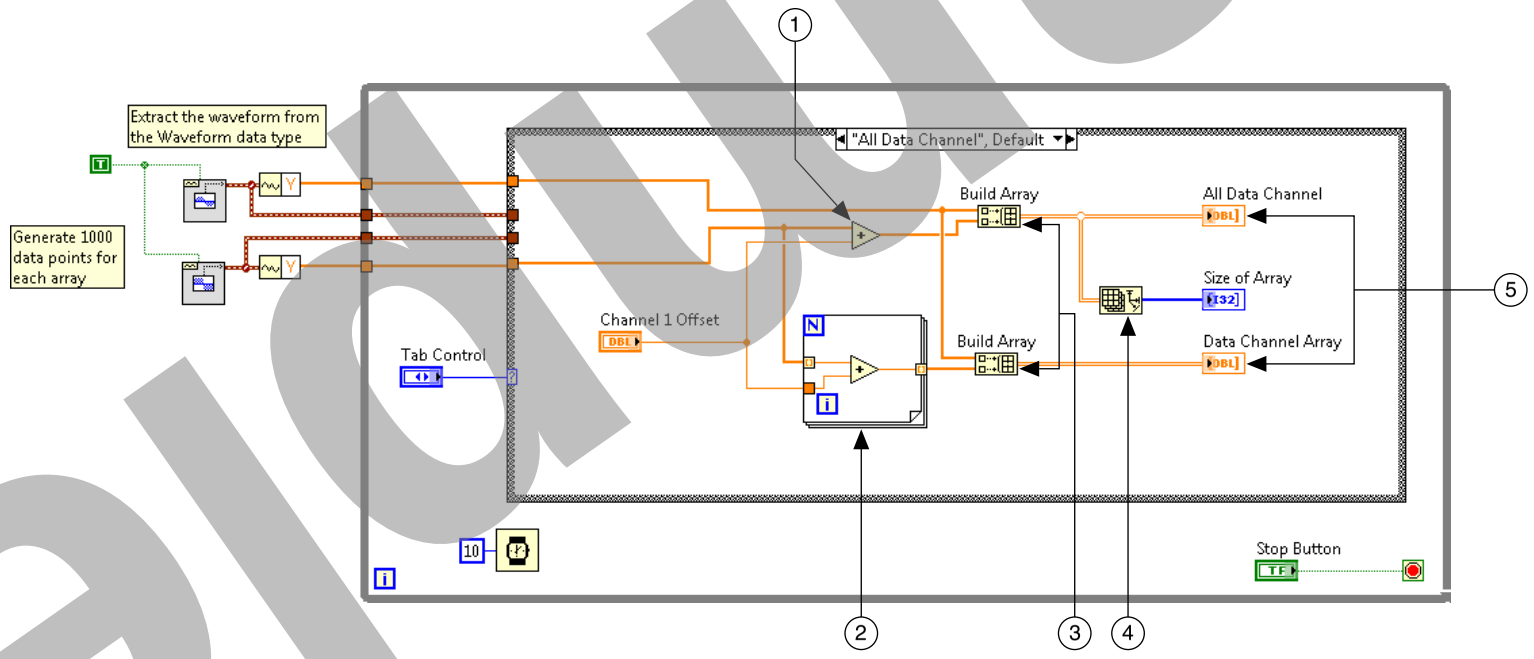
    ☐ Change the value of the **Offset** control and notice the square wave move on the **Waveform Data** chart.

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

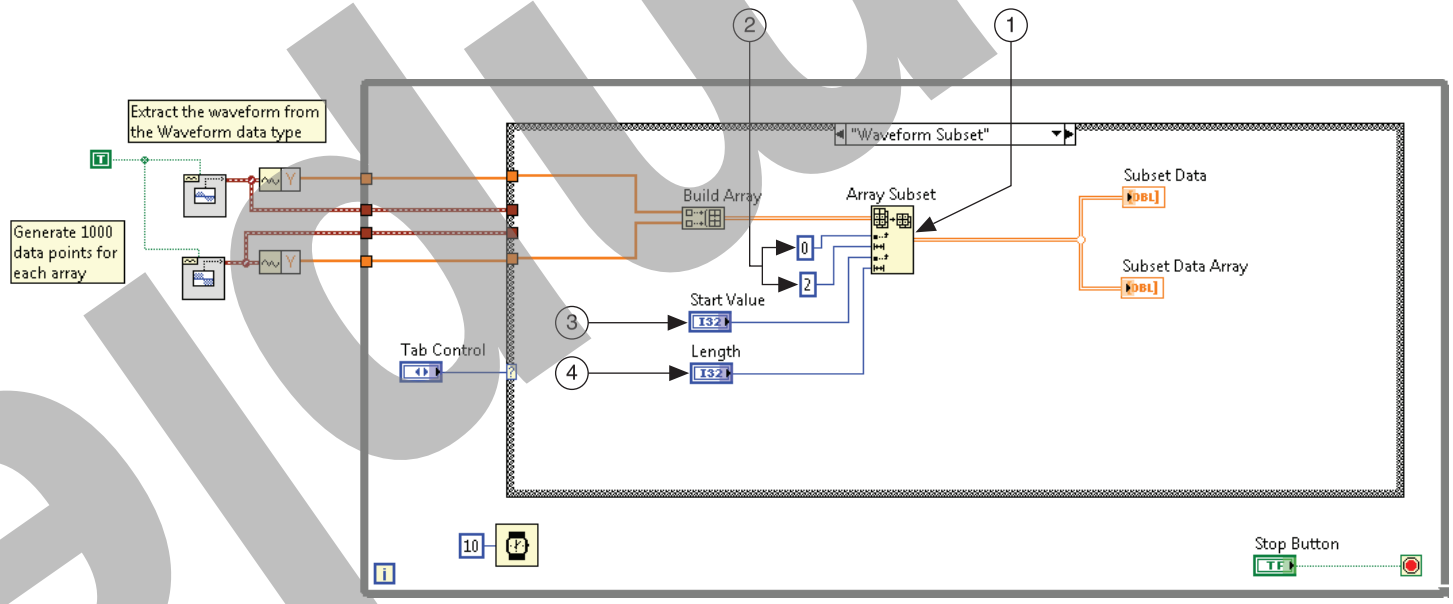


1 **Add**—Modify the same data in one array by adding the value of the Channel 1 Offset to each element of the array.
2 **For Loop**—Extracts each element of the array using auto indexing so that the Add function in the For Loop can add the scalar value.
3 **Build Array**—Takes the two 1D arrays and 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.

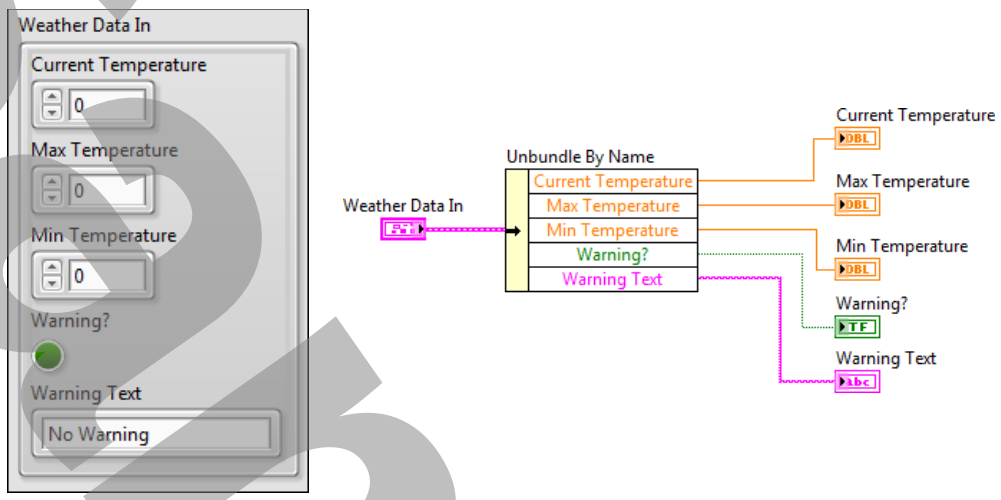
End of Exercise 5-1

E. Clusters

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

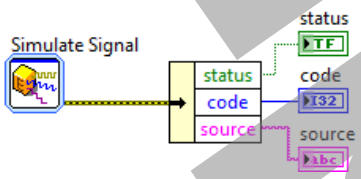
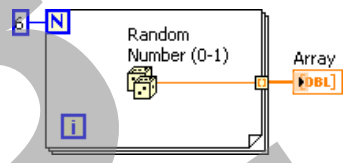
Clusters

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.

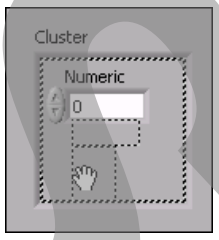
Clusters vs. Arrays

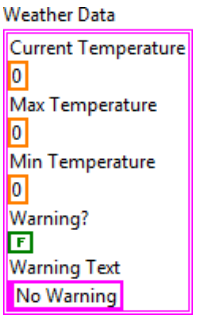
Cluster	Array
	
• Mixed data types • Fixed size	• Vary in size • Contain only one data type

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.

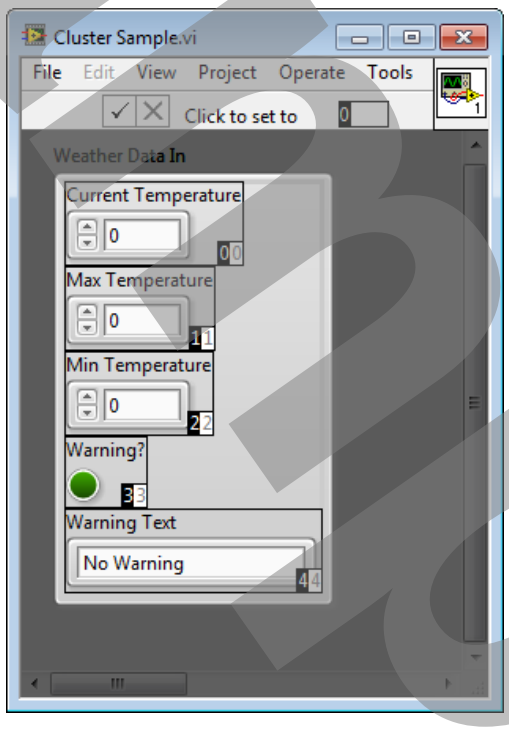




Cluster Order

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.

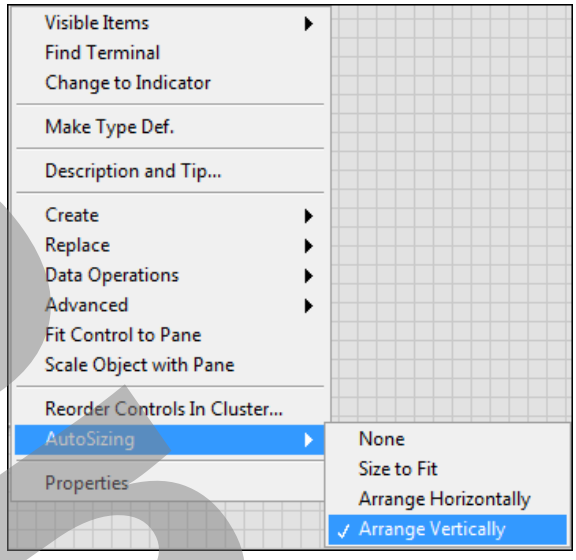
You can view and modify the cluster order by right-clicking the cluster border and selecting Reorder Controls In Cluster from the shortcut menu.



Autosizing Clusters

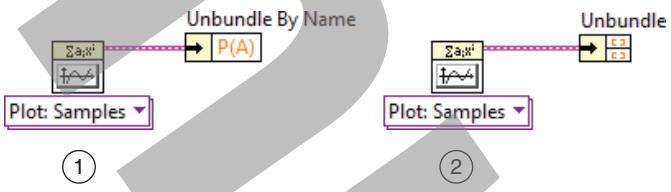
Autosizing helps you arrange elements in clusters.NI recommends the following:

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



Disassembling Clusters

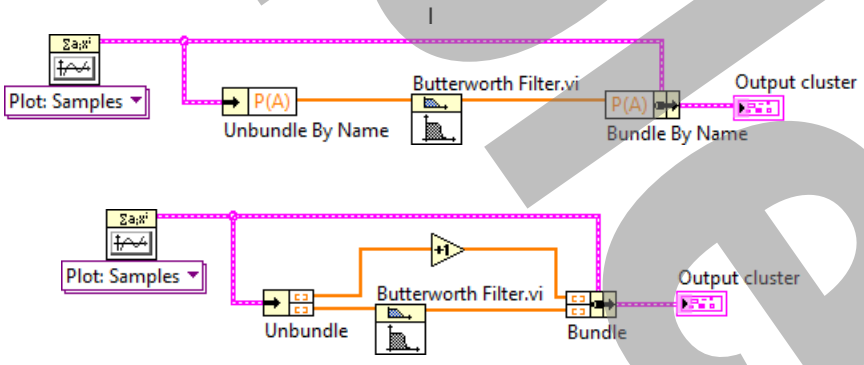
Use the Unbundle and Unbundle By Name functions to return individual cluster elements.



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.

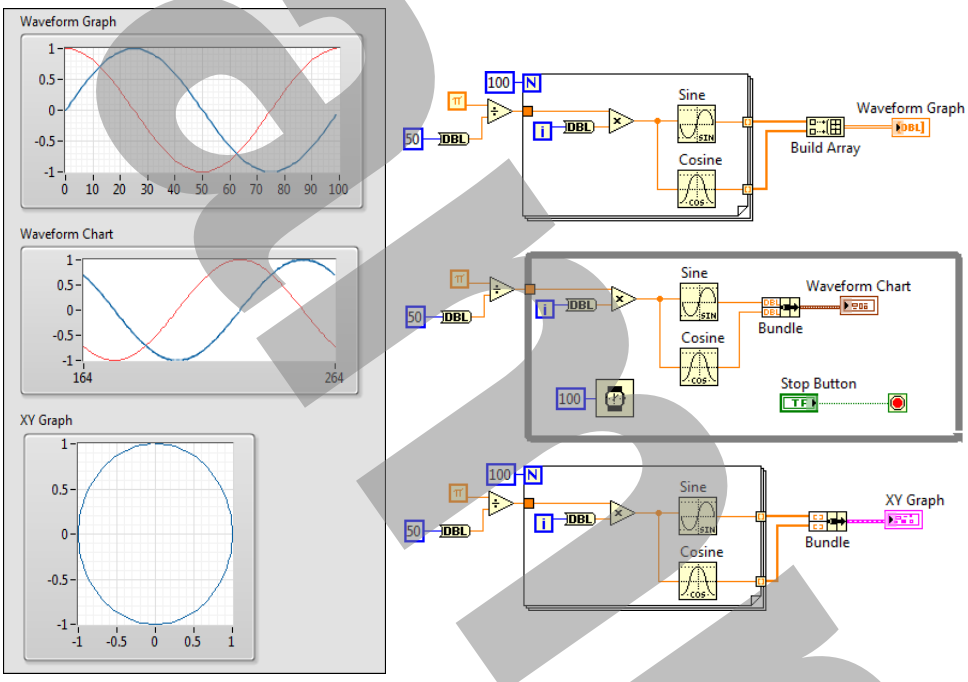


Demonstration: Creating a Cluster on the Block Diagram

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.

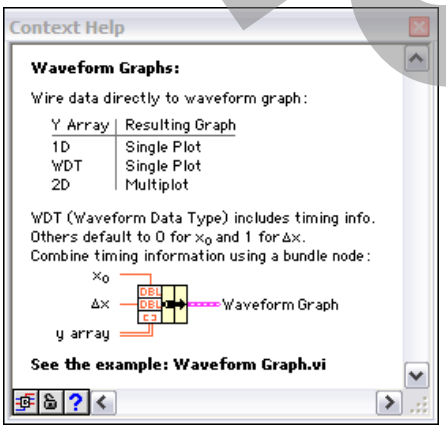
Multi-plot Graphs/Charts and XY Graph

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.



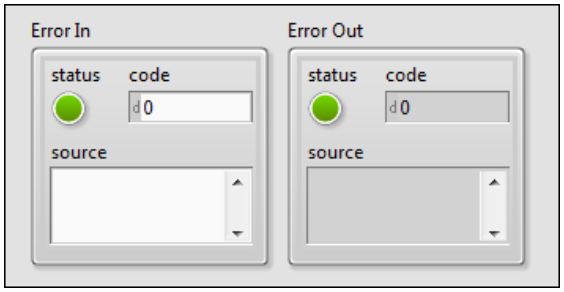
Plotting Data

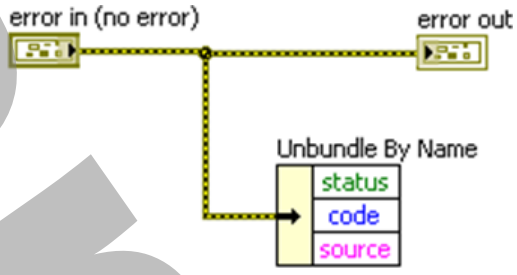
Use the Context Help window to determine how to wire multi-plot data to Waveform Graphs, Charts and XY Graphs.



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.

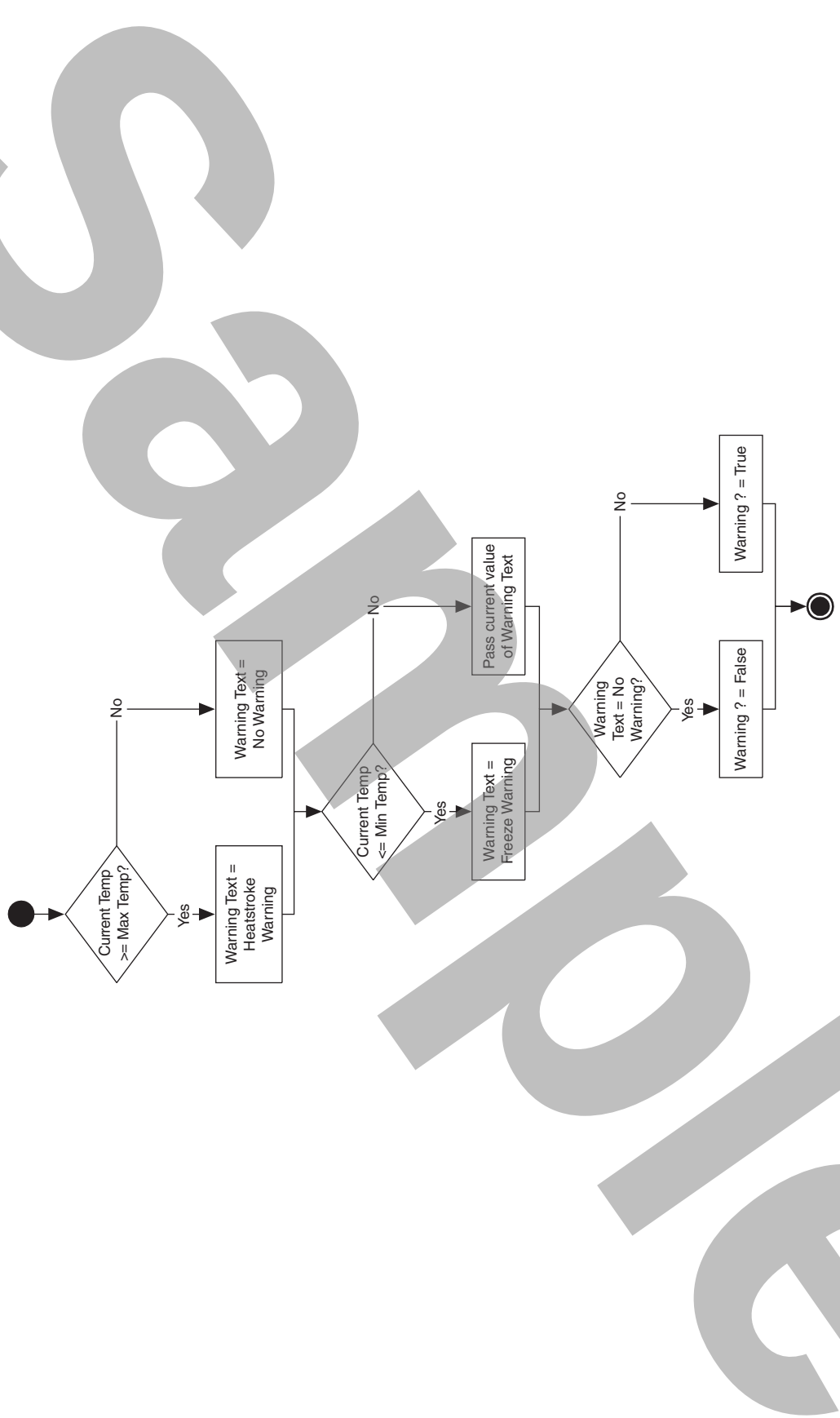
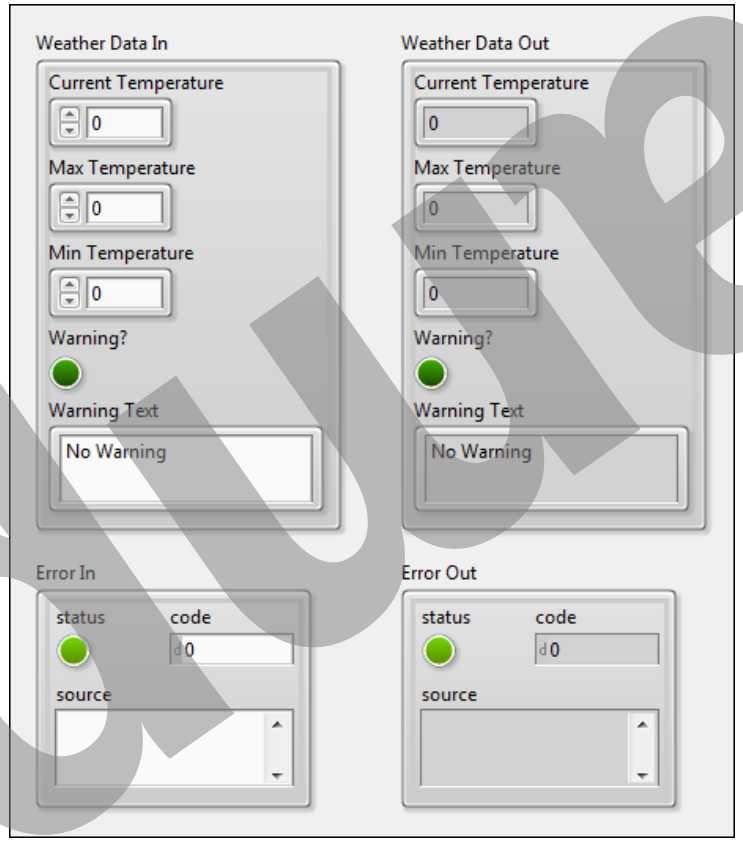


Figure 5-9. Temperature Warnings VI Flowchart

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.

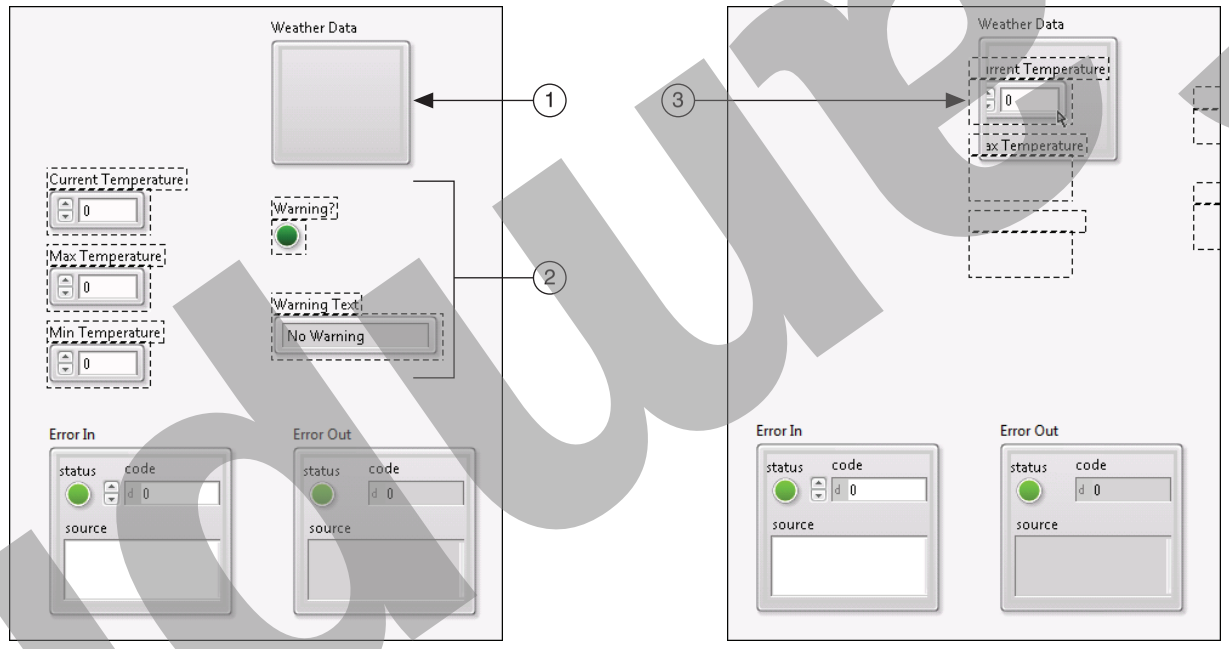
Figure 5-10. Temperature Warnings VI with Clusters Front Panel



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.

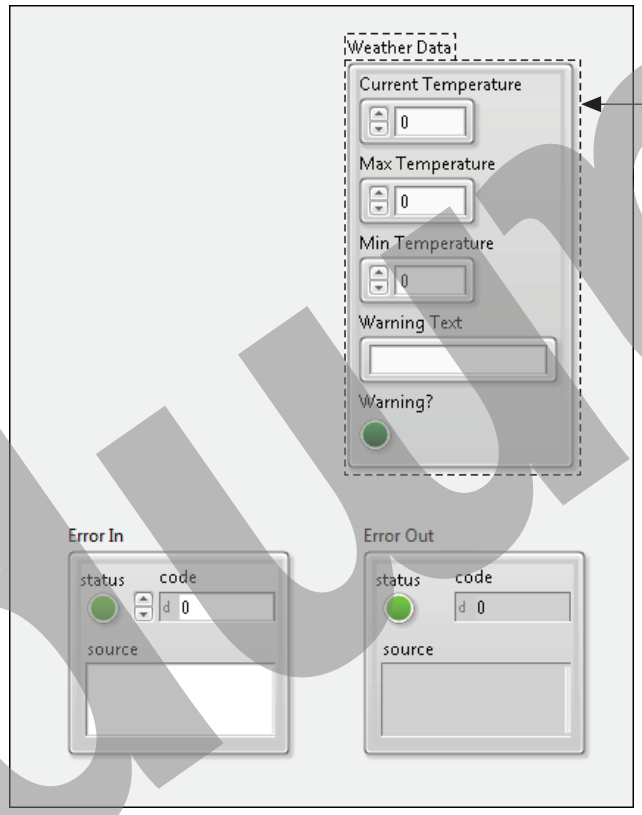
Figure 5-11. Create Cluster



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.

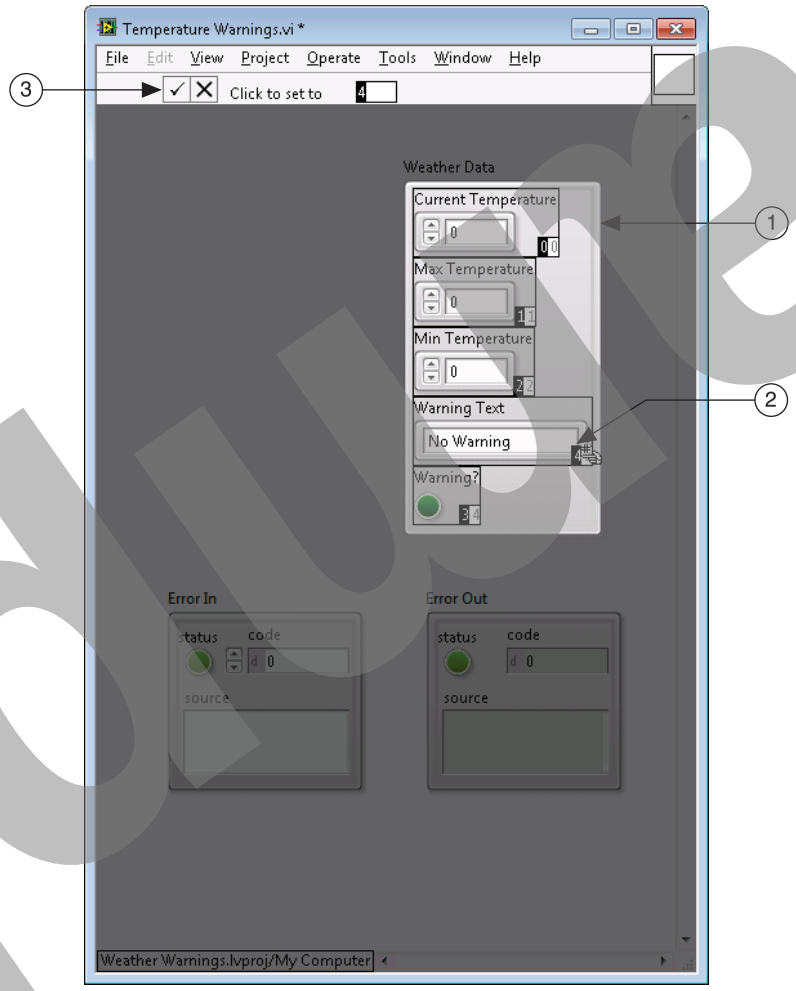
Figure 5-12. Resize Cluster Control



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.

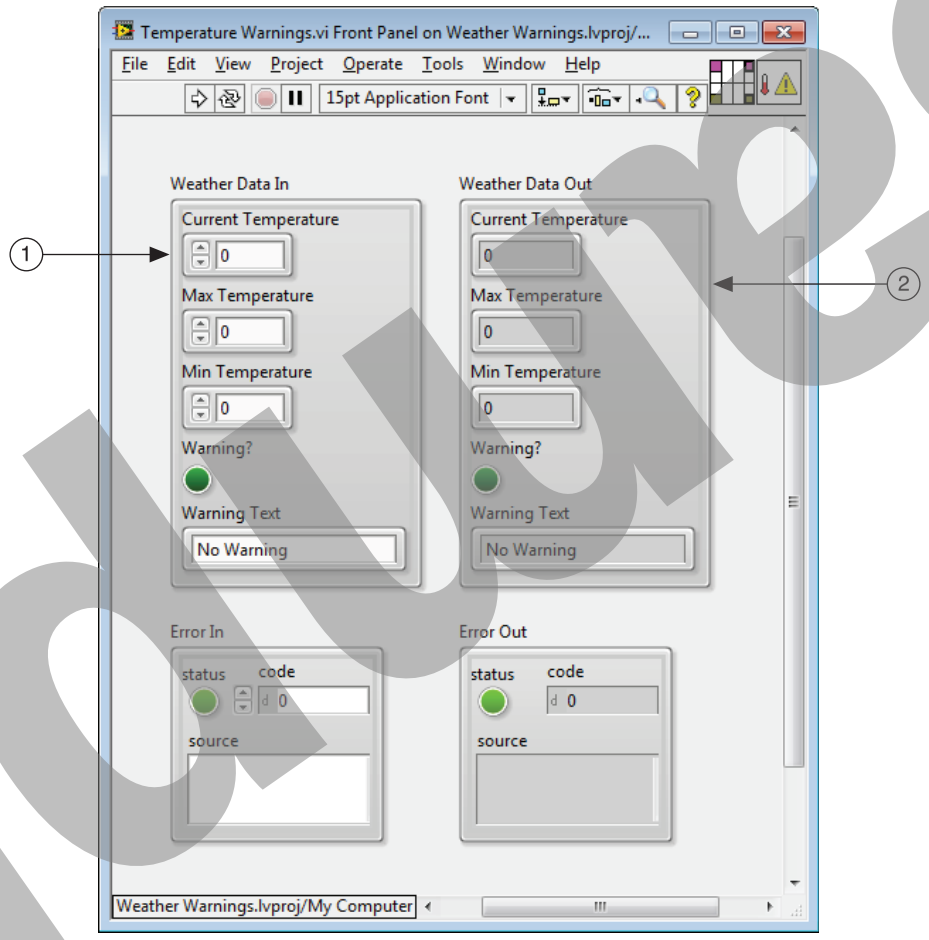
Figure 5-13. Reorder Cluster



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.

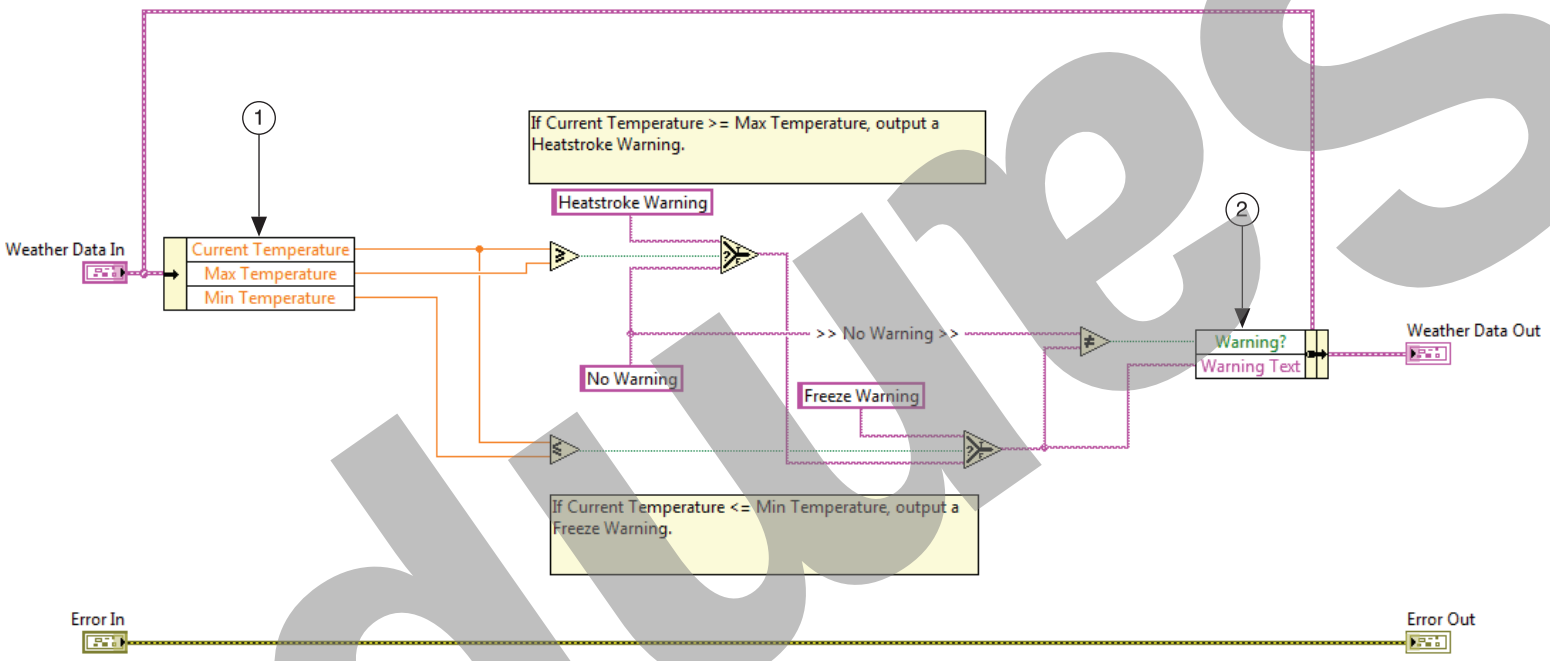
Figure 5-14. Temperature Warnings—Weather Data In and Weather Data Out Clusters



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

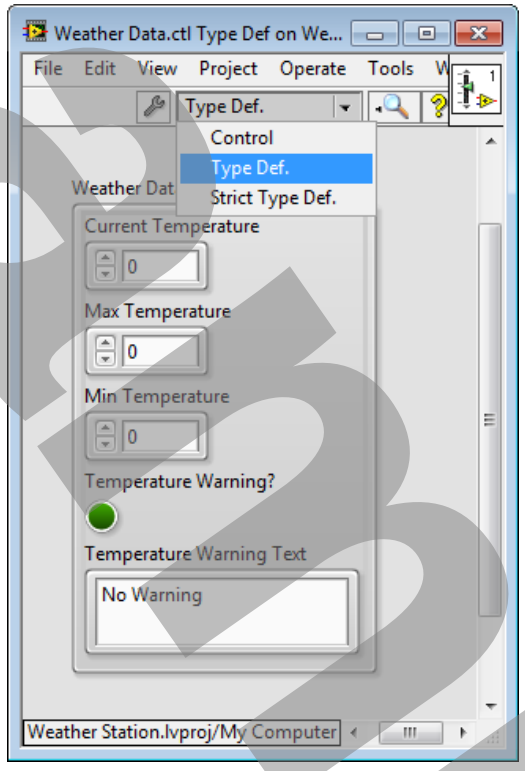
F. Type Definitions

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.

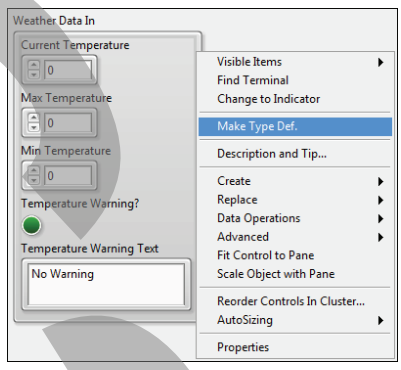
Three types of custom controls—Control, Type Definition, Strict Type Definition.

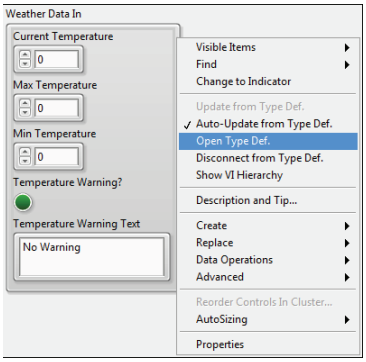


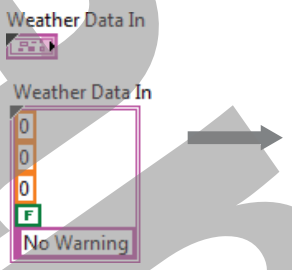
Demonstration: Difference between Control, Type Def and Strict Type Def

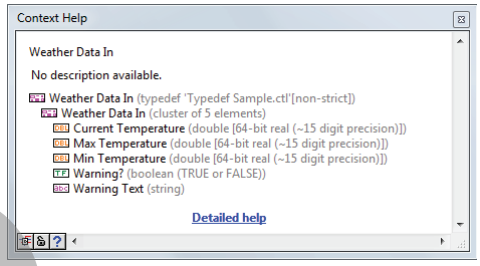
- Custom controls are templates and used as starting points for other similar controls. Changes made to one control does not reflect in other controls.
- Type definitions and strict type definitions link to all the instances of a custom control or indicator to a saved custom control or indicator file. 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









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.

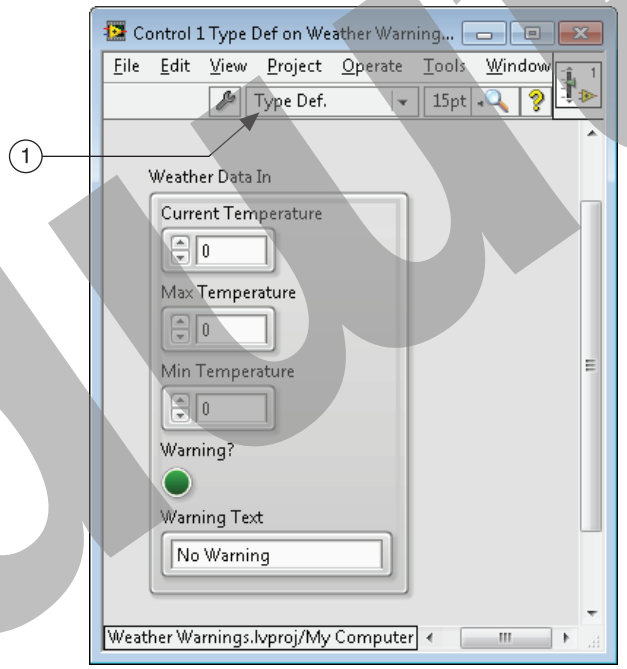
   ☐ Right-click the border of the **Weather Data In** control and select **Make Type Def**.

   ☐ 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



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.

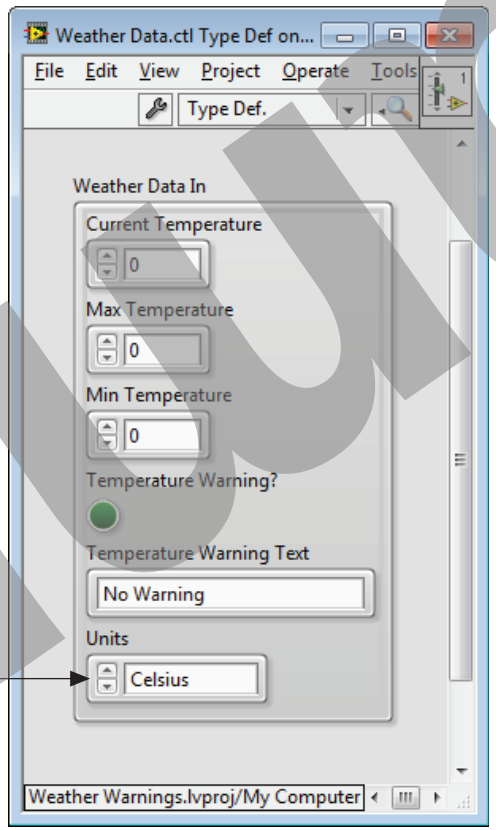
   ☐ Save the Temperature Warnings VI.

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

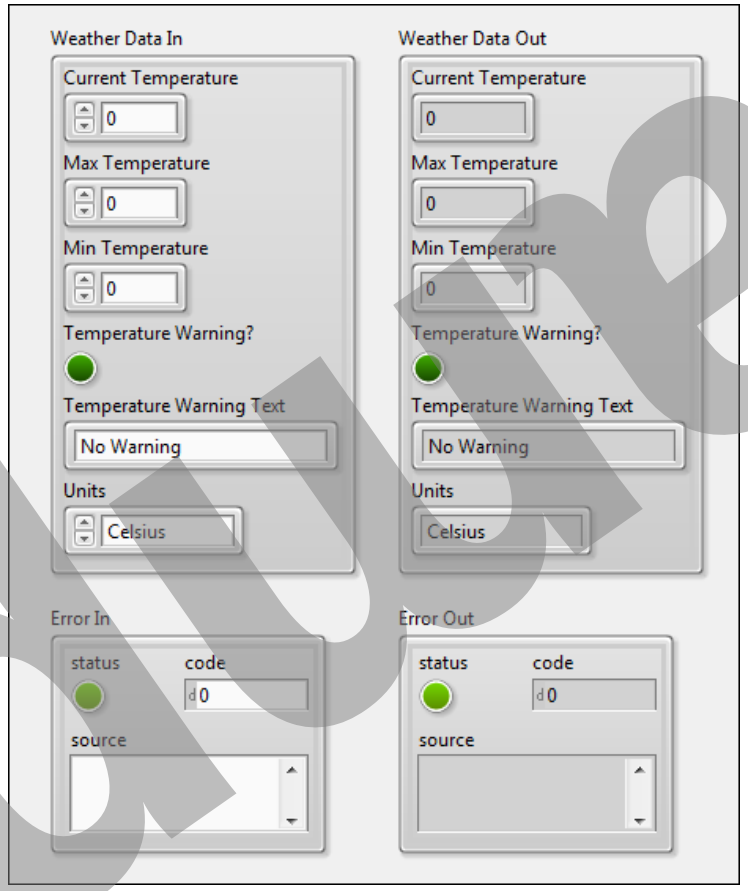


1 **Enum (Silver)**—Place an enum in the cluster and rename it `Units`. Right-click the enum and select **Edit items**. Create an item for `Celsius` and `Fahrenheit`.

   ☐ 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.

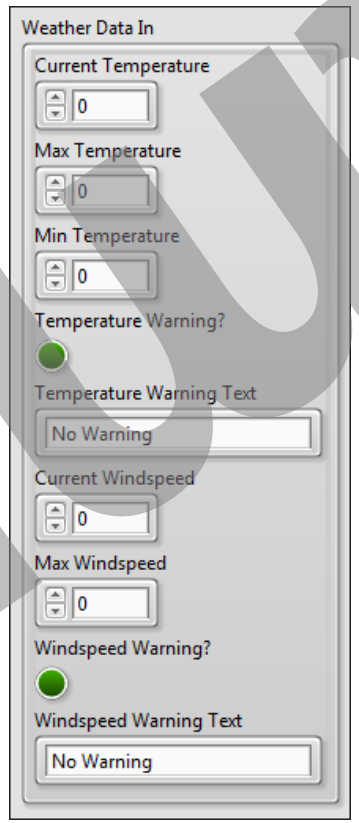
Challenge

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`.

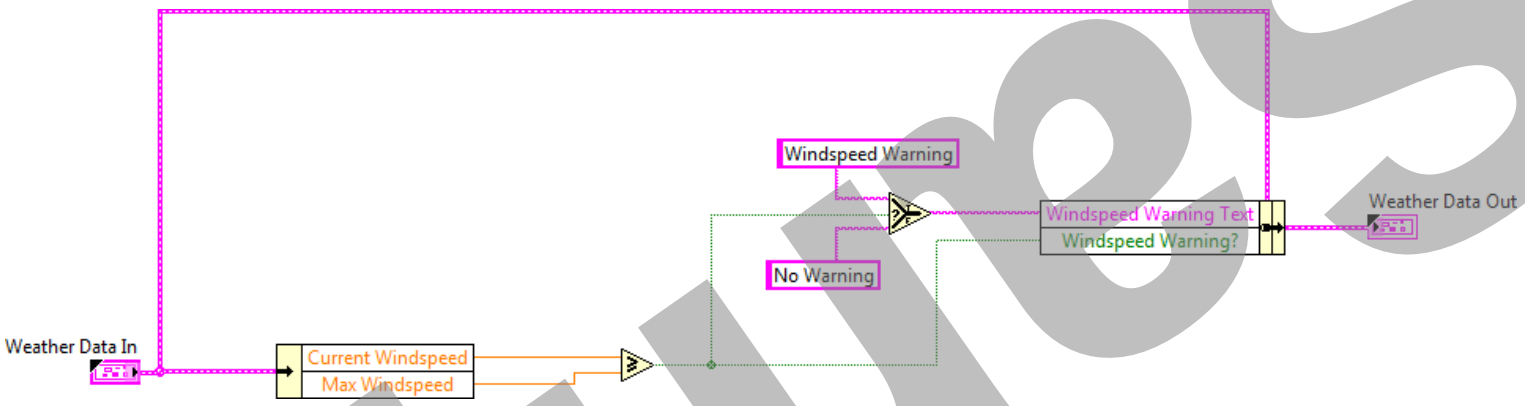
2. Open the Windspeed Warnings VI.
3. Copy the **Weather Data In** cluster from the Temperature Warnings VI to the Windspeed Warnings VI.
4. Right-click the **Weather Data In** cluster and select **Open Type Def** from the shortcut menu.
5. Modify the Weather Data type definition with windspeed controls 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

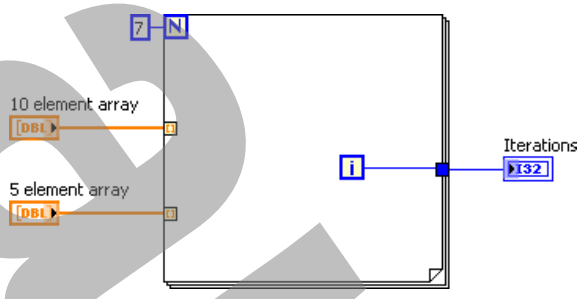
Additional Resources

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

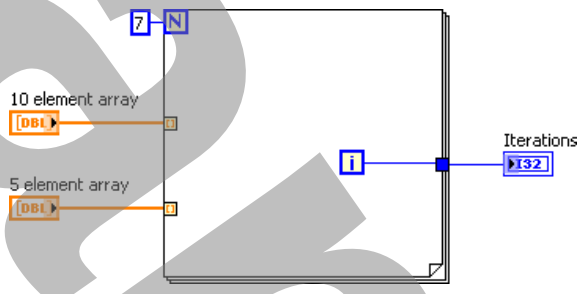
4. You have input data representing a circle: X Position (I16), Y Position (I16), and Radius (I16). In the future, you might need to modify your data to include the color of the circle (U32).

   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.