LabVIEW™ Core 1 Exercises

Course Software Version 2010
August 2010 Edition
Part Number 325291B-01

Worldwide Technical Support and Product Information
ni.com

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

Worldwide Offices

Australia 1800 300 800, Austria 43 662 457990-0, Belgium 32 (0) 2 757 0020, Brazil 55 11 3262 3599, Canada 800 433 3488, China 86 21 5050 9800, Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 358 (0) 9 725 72511, France 01 57 66 24 24, Germany 49 89 7413130, India 91 80 41190000, Israel 972 3 6393737, Italy 39 02 41309277, Japan 0120-527196, Korea 82 02 3451 3400, Lebanon 961 (0) 1 33 28 28, Malaysia 1800 887710, Mexico 01 800 010 0793, Netherlands 31 (0) 348 433 466, New Zealand 0800 553 322, Norway 47 (0) 66 90 76 60, Poland 48 22 328 90 10, Portugal 351 210 311 210, Russia 7 495 783 6851, Singapore 1800 226 5886, Slovenia 386 3 425 42 00, South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 (0) 8 587 895 00, Switzerland 41 56 2005151, Taiwan 886 02 2377 2222, Thailand 662 278 6777, Turkey 90 212 279 3031, United Kingdom 44 (0) 1635 523545

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

Contents

Student Guide

Lesson 1
Setting Up Your Hardware

Lesson 2
Navigating LabVIEW

Lesson 3
Troubleshooting and Debugging VIs

Lesson 4
Implementing a VI

Lesson 5
Relating Data

Lesson 6
Managing Resources

Lesson 7
Developing Modular Applications

Lesson 8
Common Design Techniques and Patterns

Lesson 9
Using Variables

Appendix A
Measurement Fundamentals

Appendix B
Additional Information and Resources

5

Relating Data

Exercise 5-1 Concept: 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. For each part of this exercise, begin with the `Array Investigation.vi` located in the `<Exercises>\LabVIEW Core 1\Manipulating Arrays` directory. The front panel of this VI is shown in Figure 5-1.

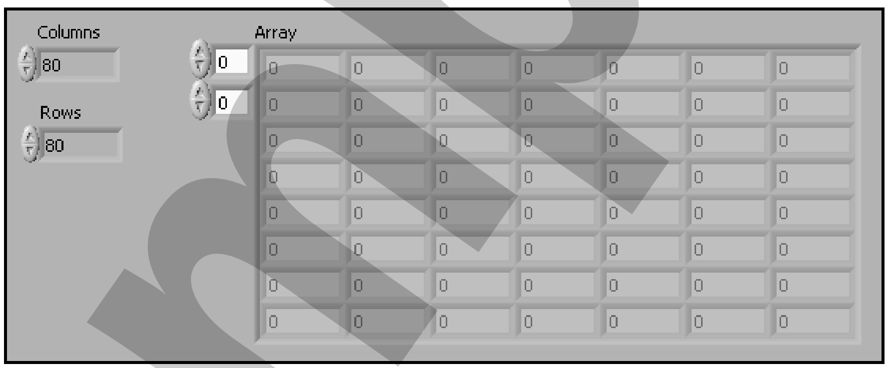


Figure 5-1. Array Investigation VI Front Panel

Figure 5-2 shows the block diagram of this VI.

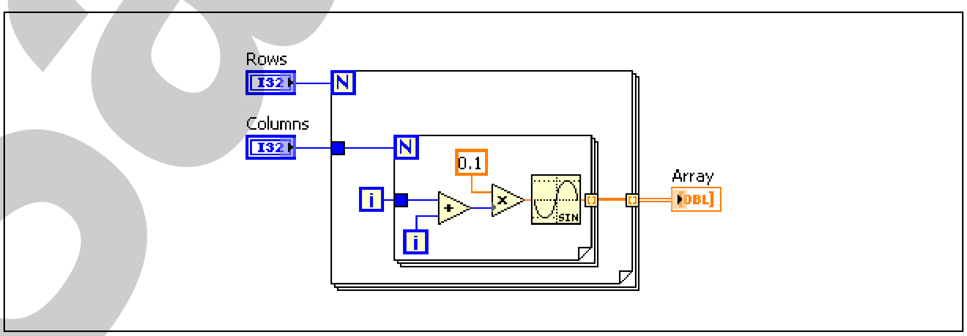


Figure 5-2. Array Investigation VI Block Diagram

This exercise is divided into three parts. You are given the scenario for each part first. Refer to the end of this exercise for detailed implementation instructions for each part.

Part 1: Iterate, Modify, and Graph Array

Modify the Array Investigation VI so that after the array is created, the array is indexed into For Loops where you multiply each element of the array by 100 and coerce each element to the nearest whole number. Graph the resulting 2D array to an intensity graph.

Part 2: Simplified Iterate, Modify, and Graph Array

Modify the Array Investigation VI or the solution from Part 1 to accomplish the same goals without using the nested For Loops.

Part 3: Create Subset Arrays

Modify the Array Investigation VI so that the VI creates a new array that contains the contents of the third row, and another new array that contains the contents of the second column.

Part 1: Implementation

Modify the Array Investigation VI so that after the array is created, the array is indexed into For Loops where you multiply each element of the array by 100 and coerce each element to the nearest whole number. Graph the resulting 2D array on an intensity graph.

1. Open `Array Investigation.vi` located in the `<Exercises>\LabVIEW Core 1\Manipulating Arrays` directory.

2. Save the VI as `Array Investigation Part 1.vi`.

3. Add an intensity graph to the front panel of the VI and autoscale the X and Y axes, as shown in Figure 5-3. To verify that autoscaling is enabled for the axes, right-click the intensity graph and select **X Scale»AutoScale X** and **Y Scale»AutoScale Y** and ensure these items are checked.

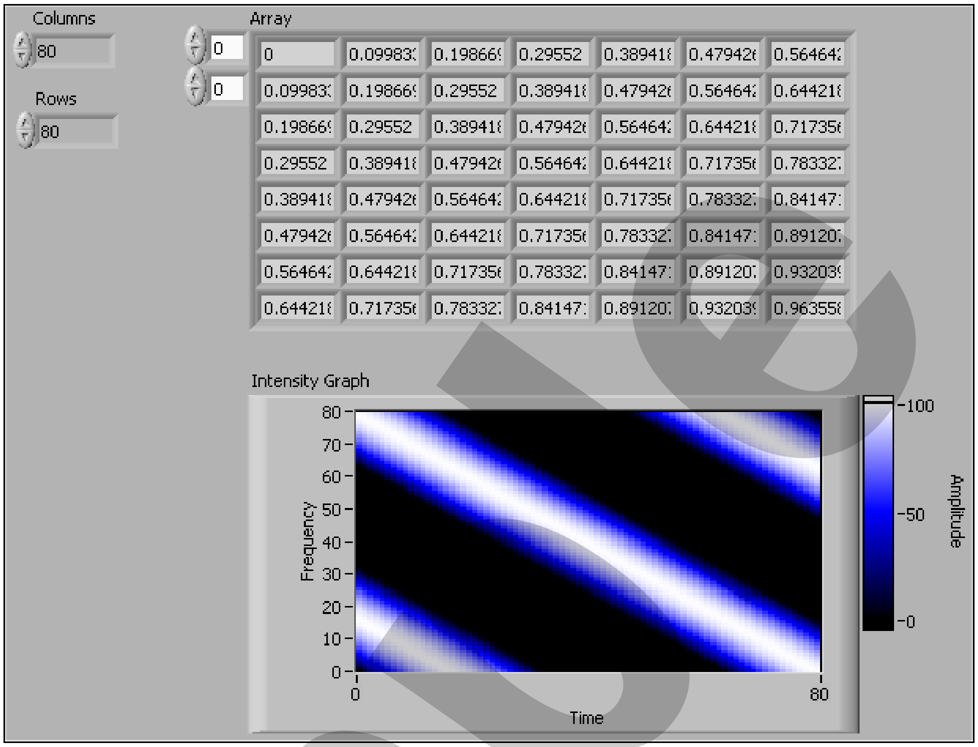


Figure 5-3. Array Investigation Part 1 VI Front Panel

4. Open the block diagram of the VI.

In the following steps, you create a block diagram similar to Figure 5-4.

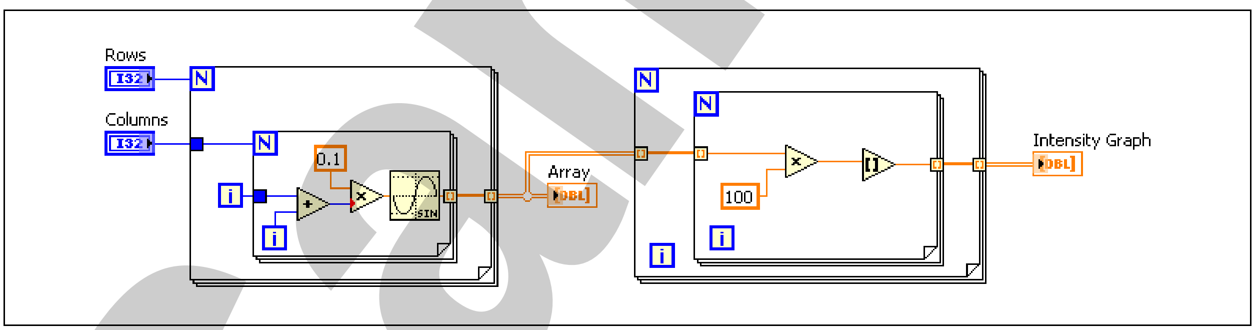
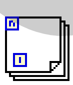


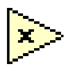
Figure 5-4. Array Investigation Part 1 VI Block Diagram

5. Iterate the Array.



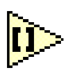
- ❑ Add a For Loop to the right of the existing code.
- ❑ Add a second For Loop inside the first For Loop.
- ❑ Wire the array indicator terminal to the interior For Loop border. This creates an auto-indexed input tunnel on both For Loops.

6. Multiply each element of the array by 100.



   - ❑ Add a Multiply function to the interior For Loop.
   - ❑ Wire the indexed input tunnel to the **x** input of the Multiply function.
   - ❑ Right-click the **y** input and select **Create»Constant** from the shortcut menu.
   - ❑ Enter `100` in the constant.

7. Round each element to the nearest whole number.



   - ❑ Add a Round To Nearest function to the right of the Multiple function.
   - ❑ Wire the output of the Multiply function to the input of the Round To Nearest function.

8. Create a 2D array on the output of the For Loops to recreate the modified array.
   - ❑ Wire the output of the Round To Nearest function to the outer For Loop. This creates an auto-indexed output tunnel on both For Loops.

9. Wire the output array to the Intensity Graph indicator.

10. Switch to the front panel.

11. Save the VI.

12. Enter values for **Rows** and **Columns**.

13. Run the VI.

Part 2: Implementation

Modify Part 1 to accomplish the same goals without using the nested For Loops.

1. Open `Array Investigation Part 1.vi` if it is not still open.

2. Save the VI as `Array Investigation Part 2.vi`.

3. Open the block diagram.

4. Right-click the border of the interior For Loop, containing the Multiply function and the Round to Nearest function, and select **Remove For Loop**.

5. Right-click the border of the remaining For Loop and select **Remove For Loop** from the shortcut menu. Your block diagram should resemble Figure 5-5.

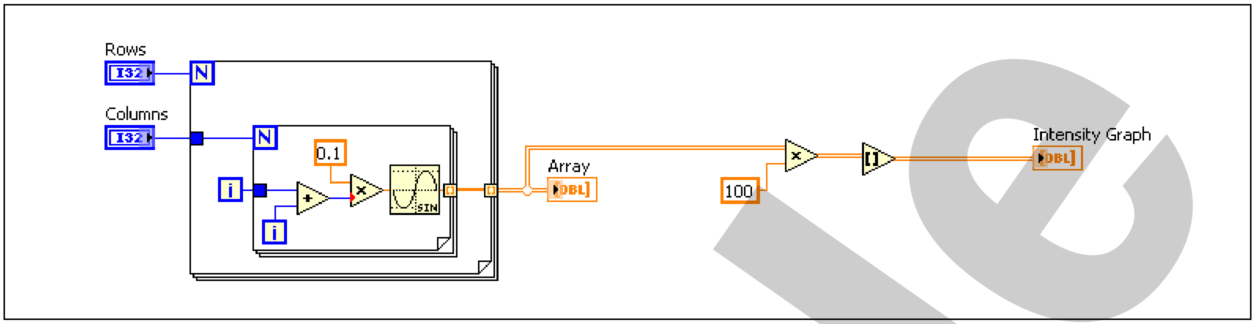


Figure 5-5. Array Investigation Part 2 VI Block Diagram

6. Save the VI.

7. Switch to the front panel.

8. Enter values for **Rows** and **Columns**.

9. Run the VI.

Notice that the VI behaves the same as the solution for Part 1. This is because mathematical functions are polymorphic. For example, because the **x** input of the Multiply function is a two-dimensional array, and the **y** input is a scalar, the Multiply function multiplies each element in the array by the scalar, and outputs an array of the same dimension as the **x** input.

Part 3: Implementation

Modify Array Investigation VI so that the VI creates a new array that contains the contents of the third row, and another new array that contains the contents of the second column.

1. Open `Array Investigation.vi` located in the `<Exercises>\LabVIEW Core 1\Manipulating Arrays` directory.

2. Save the VI as `Array Investigation Part 3.vi`.

3. Open the block diagram of the VI.

In the following steps, you build a block diagram similar to that shown in Figure 5-6.

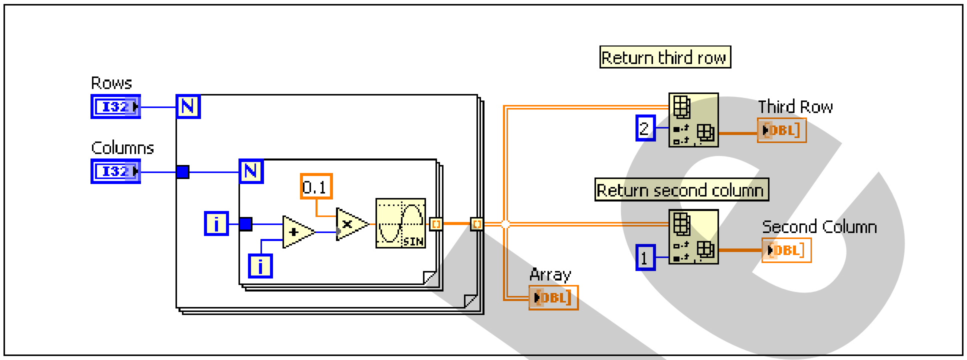
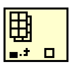


Figure 5-6. Array Investigation Part 3 VI Block Diagram

4. Retrieve the third row of data from **Array** using the Index Array function.

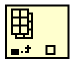


- ❑ Add the Index Array function to the block diagram.
- ❑ Wire **Array** to the **array** input of the Index Array function.

Tip The Index Array function accepts an *n*-dimensional array. After you wire the input array to the Index Array function, the input and output terminal names change to match the dimension of the array wired. Therefore, wire the input array to the Index Array function before wiring any other terminals.

- ❑ Right-click the **index(row)** input of the Index Array function.
- ❑ Select **Create»Constant** from the shortcut menu.
- ❑ Enter `2` in the constant to retrieve the third row. Remember that the index begins at zero.
- ❑ Right-click the **subarray** output of the Index Array function.
- ❑ Select **Create»Indicator** from the shortcut menu.
- ❑ Name the indicator `Third Row`.

5. Retrieve the second column of data from the Array using the Index Array function.



❑ Add another Index Array function to the block diagram.

❑ Wire **Array** to the **array** input of the Index Array function.

❑ Right-click the **disabled index(col)** input of the Index Array function.

❑ Select **Create»Constant**.

❑ Enter `1` in the constant to retrieve the second column because the index begins at zero.

❑ Right-click the **subarray** output of the Index Array function.

❑ Select **Create»Indicator**.

❑ Name the indicator `Second Column`.

6. Save the VI.
7. Switch to the front panel.
8. Enter values for **Rows** and **Columns**.
9. Run the VI.

End of Exercise 5-1

Exercise 5-2 Concept: Clusters

Goal

Create clusters on the front panel window, reorder clusters, and use the cluster functions to assemble and disassemble clusters.

Description

In this exercise, follow the instructions to experiment with clusters, cluster order, and cluster functions. The VI you create has no practical applications, but is useful for understanding cluster concepts.

1. Open a blank VI.
2. Save the VI as `Cluster Experiment.vi` in the `<Exercises>\LabVIEW Core 1\Clusters` directory.

In the following steps, you create a front panel similar to Figure 5-7.

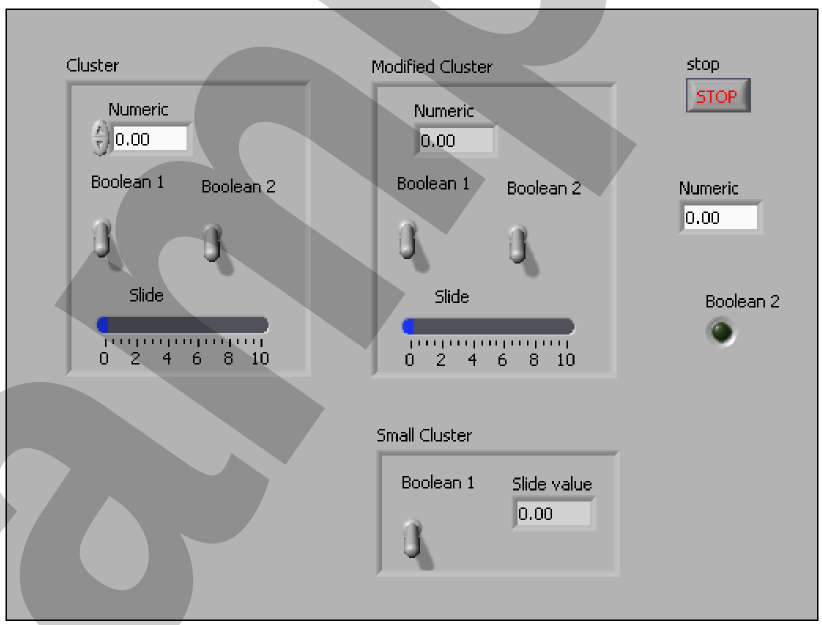


Figure 5-7. Cluster Experiment VI Front Panel

3. Add a stop button to the front panel window.
4. Add a numeric indicator to the front panel window.
5. Add a round LED to the front panel.
6. Rename the LED `Boolean 2`.

7. Create a cluster named **Cluster**, containing a numeric, two toggle switches, and a slide.

   ❑ Add a cluster shell to the front panel.

   ❑ Add a numeric control to the cluster.

   ❑ Add two vertical toggle switches to the cluster.

   ❑ Rename the **Boolean** toggle switches to `Boolean 1` and `Boolean 2`.

   ❑ Add a horizontal fill slide to the cluster.

8. Create **Modified Cluster**, containing the same contents as **Cluster**, but indicators instead of controls.

   ❑ Create a copy of **Cluster**.

   ❑ Relabel the copy `Modified Cluster`.

   ❑ Right-click the shell of **Modified Cluster**, and select **Change to Indicator** from the shortcut menu.

9. Create **Small Cluster**, containing a Boolean indicator and a numeric indicator.

   ❑ Create a copy of **Modified Cluster**.

   ❑ Relabel the copy `Small Cluster`.

   ❑ Delete the second toggle switch.

   ❑ Delete the horizontal fill slide indicator.

   ❑ Right-click **Small Cluster** and select **Autosizing»Size to Fit**.

   ❑ Relabel the numeric indicator to `Slide value`.

   ❑ Resize the cluster as needed.

10. Verify the cluster order of **Cluster**, **Modified Cluster**, and **Small Cluster**.

   - ❑ Right-click the boundary of **Cluster** and select **Reorder Controls in Cluster** from the shortcut menu.
   - ❑ Confirm the cluster order shown in Figure 5-8.

   - ❑ Click the **Confirm** button on the toolbar to set the cluster order and exit the cluster order edit mode.
   - ❑ Right-click the boundary of **Modified Cluster** and select **Reorder Controls in Cluster** from the shortcut menu.
   - ❑ Confirm the cluster orders shown in Figure 5-8. **Modified Cluster** should have the same cluster order as **Cluster**.

   - ❑ Click the **Confirm** button on the toolbar to set the cluster order and exit the cluster order edit mode.

   - ❑ Right-click the boundary of **Small Cluster** and select **Reorder Controls in Cluster** from the shortcut menu. Click the **Confirm** button on the toolbar to set the cluster order and exit the cluster order edit mode.
   - ❑ Confirm the cluster orders shown in Figure 5-8.

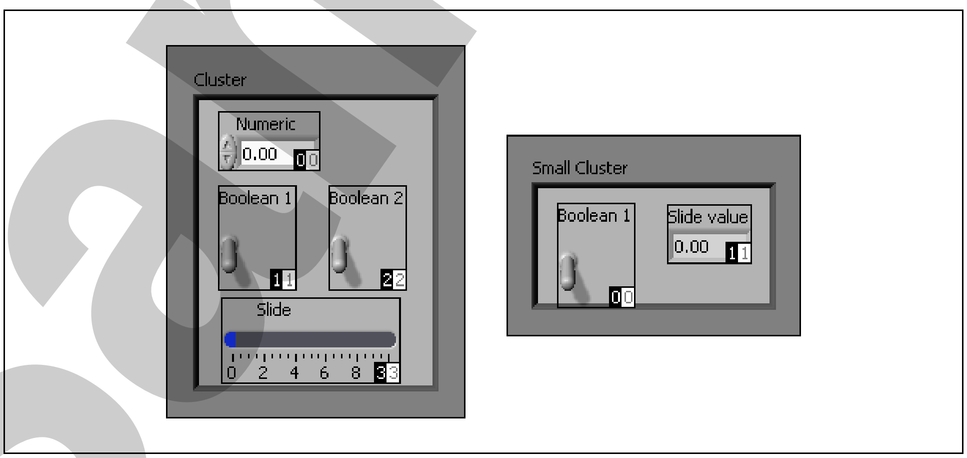


Figure 5-8. Cluster Orders

In the following steps, build the block diagram shown in Figure 5-9.

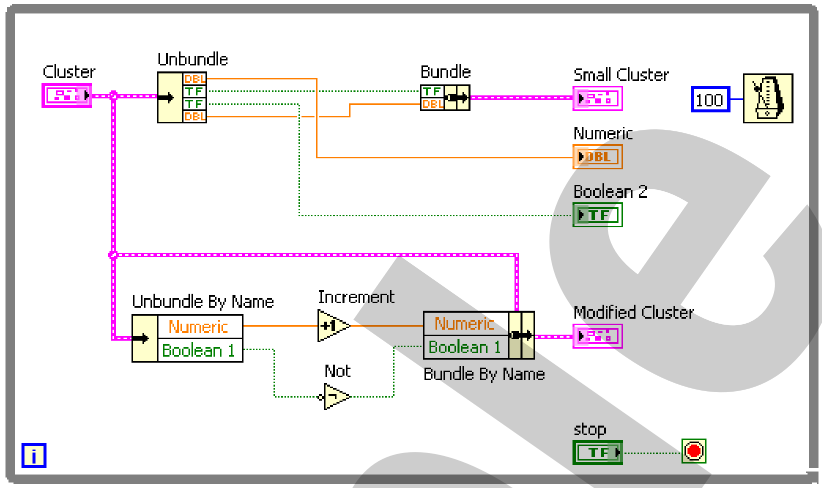
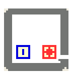
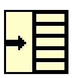


Figure 5-9. Cluster Experiment VI Block Diagram



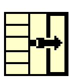
11. Add the While Loop from the **Structures** palette to the block diagram.

12. Disassemble **Cluster**.



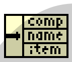
- ❑ Add the Unbundle function to the block diagram.
- ❑ Wire **Cluster** to the input of the Unbundle function to resize the function automatically.

13. Assemble **Small Cluster**.

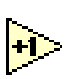


- ❑ Add the Bundle function to the block diagram.
- ❑ Wire the Bundle function as shown in Figure 5-9.

14. Assemble **Modified Cluster**.

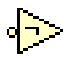


- ❑ Add the Unbundle by Name function to the block diagram.
- ❑ Wire the **Cluster** to the Unbundle by Name function.
- ❑ Resize the Unbundle by Name function to have two output terminals.
- ❑ Select **Numeric** in the first node, and **Boolean 1** in the second node. If a label name is not correct, use the Operating tool to select the correct item.

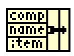


- ❑ Add the Increment function to the block diagram.

- ❑ Wire the **Numeric** output of the Unbundle By Name function to the input of the Increment function. This function adds one to the value of **Numeric**.

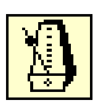


- ❑ Add the Not function to the block diagram.
- ❑ Wire the **Boolean 1** output of the Unbundle By Name function to the **x** input of the Not function. This function returns the logical opposite of the value of **Boolean**.



- ❑ Add the Bundle by Name function to the block diagram.
- ❑ Wire **Cluster** to the input cluster input.
- ❑ Resize this function to have two input terminals.
- ❑ Select **Numeric** in the first node and **Boolean 1** in the second node. If a label name is not correct, use the Operating tool to select the correct item.
- ❑ Wire the output of the Increment function to Numeric.
- ❑ Wire the output of the Not function to Boolean 1.
- ❑ Wire the output of the Bundle By Name function to the Modified Cluster indicator.

15. Add a wait function to provide the processor with time to complete other tasks.



- ❑ Add the Wait Until Next ms Multiple function to the block diagram.
- ❑ Right-click the **millisecond multiple** terminal of the Wait Until Next ms Multiple function.
- ❑ Select **Create»Constant** from the shortcut menu.
- ❑ Enter `100` in the constant.

16. Complete the block diagram and wire the objects as shown in Figure 5-9.
17. Save the VI.
18. Display the front panel.
19. Run the VI.

20. Enter different values in **Cluster** and notice how values entered in **Cluster** affect the **Modified Cluster** and **Small Cluster** indicators. Is this the behavior you expected?

21. Click the **Stop** button when you are done.

22. Change the cluster order of **Modified Cluster**. Run the VI. How did the changed order affect the behavior?

23. Close the VI. Do not save changes.

End of Exercise 5-2

Exercise 5-3 Concept: Type Definition

Goal

Explore the differences between a type definition and a strict type definition.

Description

1. Open a blank VI.
2. Create a custom control with a strict type definition status.
   - ❑ Add a numeric control to the front panel window and rename it as Strict Type Def Numeric.
   - ❑ Right-click the control and select **Advanced»Customize** from the shortcut menu to open the Control Editor.
   - ❑ Select **Strict Type Def.** from the **Control Type** pull-down menu.
   - ❑ Right-click the numeric control and select **Representation» Unsigned Long** from the shortcut menu.
   - ❑ Select **File»Save**.
   - ❑ Name the control Strict Type Def Numeric.ctl in the <Exercises>\LabVIEW Core 1\Type Definition directory.
   - ❑ Close the Control Editor window.
   - ❑ Click **Yes** when asked if you would like to replace the original control.
3. Explore the strictly defined custom numeric.
   - ❑ Right-click the Strict Type Def Numeric control and select **Properties** from the shortcut menu. Notice that the only options available are Appearance, Documentation, and Key Navigation. All other properties are defined by the strict type definition.
   - ❑ Click **Cancel** to exit the **Properties** dialog box.
   - ❑ Right-click the Strict Type Def Numeric control again. Notice that representation is not available on the shortcut menu. Also notice that you can open the type definition or disconnect from the type definition.

4. Edit the strict type def control.

   ❑ Right-click the Strict Type Def Numeric control and select **Open Type Def.** from the shortcut menu.

   ❑ Right-click the numeric control and select **Representation»DBL** from the shortcut menu in the Control Editor window.

   ❑ Select **File»Save**.

   ❑ Close the Control Editor window.

   ❑ Select **Help»Show Context Help** to open the Context Help window.

   ❑ Hover your mouse over the control on the VI and notice that it changed from a U32 numeric data type to a DBL numeric data type.

   ❑ Right-click the Strict Type Def Numeric control and select **Open Type Def.** from the shortcut menu.

   ❑ Change the physical appearance of the numeric control by resizing it in the Control Editor window.

   ❑ Select **File»Save**.

   ❑ Close the Control Editor window.

   ❑ Notice that editing the strict type def control updates the size of the numeric control on the VI front panel.

5. Create a custom control with a type definition status.

   ❑ Add another numeric control to the front panel window and rename it as `Type Def Numeric`.

   ❑ Right-click the control and select **Advanced»Customize** from the shortcut menu to open the Control Editor.

   ❑ Select **Type Def.** from the **Control Type** pull-down menu.

   ❑ Right-click the numeric control and select **Representation» Unsigned Long** from the shortcut menu.

   ❑ Select **File»Save**.

   ❑ Name the control `Type Def Numeric.ctl` in the `<Exercises>\LabVIEW Core 1\Type Definition` directory.

- ❑ Close the Control Editor window.
- ❑ Click **Yes** when asked if you would like to replace the original control.

6. Explore the type defined custom numeric.

   - ❑ Right-click the Type Def Numeric control and select **Properties** from the shortcut menu. Notice that more items are available, such as Data Entry and Display Format.
   - ❑ Click **Cancel** to exit the **Properties** dialog box.
   - ❑ Right-click the Type Def Numeric control again. Notice that **Representation** is dimmed on the shortcut menu because the type definition defines the data type. Also notice that you can choose whether to auto-update with the type definition.

7. Edit the type def control.

   - ❑ Right-click the Type Def Numeric control and select **Open Type Def.** from the shortcut menu.
   - ❑ Right-click the Type Def Numeric control and select **Representation»DBL** from the shortcut menu in the Control Editor window.
   - ❑ Select **File»Save**.
   - ❑ Close the Control Editor window.
   - ❑ Select **Help»Show Context Help** to open the Context Help window.
   - ❑ Hover your mouse over the Type Def Numeric control on the VI and notice that it changed from a U32 numeric data type to a DBL numeric data type.
   - ❑ Right-click the Type Def Numeric control and select **Open Type Def.** from the shortcut menu.
   - ❑ Change the physical appearance of the numeric control by resizing it in the Control Editor window.
   - ❑ Select **File»Save**.

- ❑ Close the Control Editor window.
- ❑ Notice that resizing the type def control in the Control Editor did not update the size of the Type Def Numeric control on the VI front panel. Instances of a type def control will only update when the data type of the type definition changes.

8. Add another instance of the custom control to the front panel window and disconnect it from the type definition.

   - ❑ Select **Select a Control** from the **Controls** palette.
   - ❑ Select the `Type Def Numeric.ctl` from the `<Exercises>\LabVIEW Core 1\Type Definition` directory.
   - ❑ Click **OK**.
   - ❑ Right-click the new control and select **Disconnect from Type Def.** from the shortcut menu.
   - ❑ Click **OK**.
   - ❑ Right-click the control again and notice that you can now change the **Representation** because the numeric is no longer linked to the type definition.

9. Close the VI when you are finished. You do not need to save the VI.

End of Exercise 5-3

Notes