LabVIEW™ Core 1 Course Manual

Course Software Version 2010
August 2010 Edition
Part Number 325290B-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

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

Glossary

Index

5

Relating Data

Sometimes it is beneficial to group data related to one another. Use arrays and clusters to group related data in LabVIEW. Arrays combine data of the same data type into one data structure, and clusters combine data of multiple data types into one data structure. Use type definitions to define custom arrays and clusters. This lesson explains arrays, clusters, and type definitions, and applications where using these can be beneficial.

Topics

A. Arrays

B. Clusters

C. Type Definitions

A. Arrays

An array consists of elements and dimensions. Elements are the data that make up the array. A dimension is the length, height, or depth of an array. An array can have one or more dimensions and as many as $(2^{31}) - 1$ elements per dimension, memory permitting.

You can build arrays of numeric, Boolean, path, string, waveform, and cluster data types. Consider using arrays when you work with a collection of similar data and when you perform repetitive computations. Arrays are ideal for storing data you collect from waveforms or data generated in loops, where each iteration of a loop produces one element of the array.

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

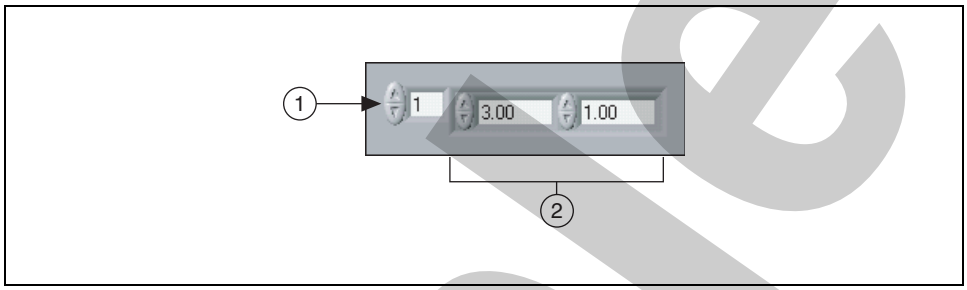
Restrictions

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 multiplot XY graphs. Refer to the clusters section of this lesson for more information about clusters.

An example of a simple array is a text array that lists the twelve months of the year. LabVIEW represents this as a 1D array of strings with twelve elements.

Array elements are ordered. An array uses an index so you can readily access any particular element. The index is zero-based, which means it is in the range 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.

Figure 5-1 shows an example of an array of numerics. The first element shown in the array (`3.00`) is at index 1, and the second element (`1.00`) is at index 2. The element at index 0 is not shown in this image, because element 1 is selected in the index display. The element selected in the index display always refers to the element shown in the upper left corner of the element display.



1 Index Display 2 Element Display

Figure 5-1. Array Control of Numerics

Creating Array Controls and Indicators

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.

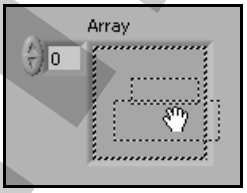


Figure 5-2. Placing a Numeric Control in an Array Shell

If you attempt to drag an invalid control or indicator into the array shell, you are unable to place the control or indicator in the array shell.

You must insert an object in the array shell before you use the array on the block diagram. Otherwise, the array terminal appears black with an empty bracket and has no data type associated with it.

Two-Dimensional Arrays

The previous examples use 1D 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. Figure 5-3 shows an 8 column by 8 row 2D array, which contains 8 × 8 = 64 elements.

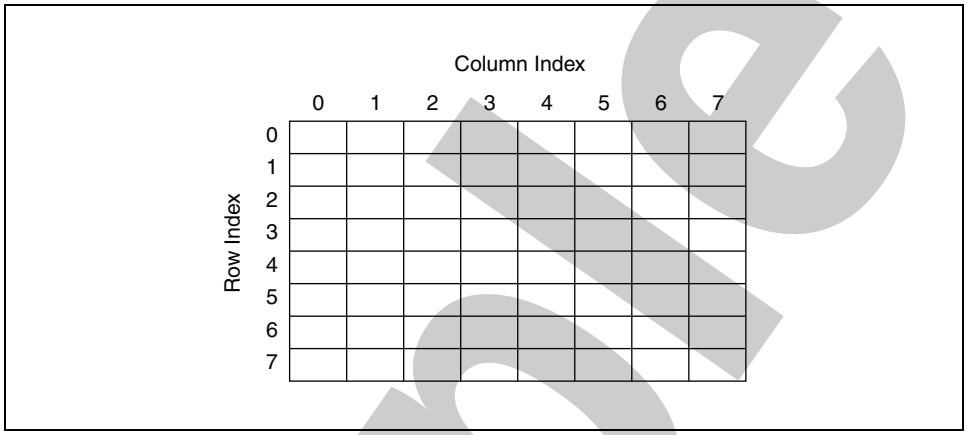


Figure 5-3. 2D Array

To add a multidimensional array to the front panel, right-click the index display and select **Add Dimension** from the shortcut menu. You also can resize the index display until you have as many dimensions as you want.

Initializing Arrays

You can initialize an array or leave it unitialized. When an array is initialized, you defined the number of elements in each dimension and the contents of each element. An uninitialized array contains a fixed number of dimensions but no elements. Figure 5-4 shows an uninitialized 2D array control. Notice that the elements are all dimmed. This indicates that the array is uninitialized.

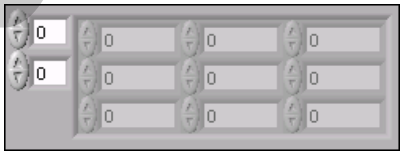


Figure 5-4. 2D Uninitialized Array

In Figure 5-5, six elements are initialized.

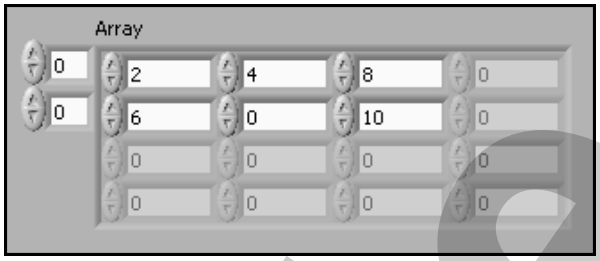


Figure 5-5. An Initialized 2D Array with Six Elements

In a 2D array, after you initialize an element in a row, the remaining elements in that row are initialized and populated with the default value for the data type. For example, in Figure 5-6, if you enter 4 into the element in the first column, third row, the elements in the second and third column in the third row are automatically populated with a 0.

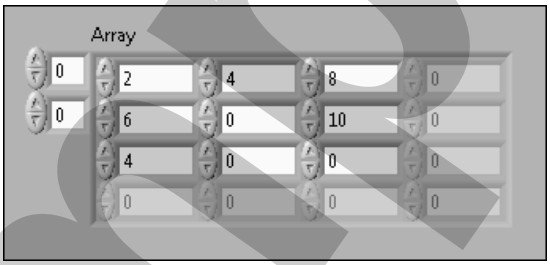


Figure 5-6. An Array That Has Been Autopopulated with Zeroes

Creating Array Constants

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. You can use an array constant to store constant data or as a basis for comparison with another array.

Auto-Indexing Array Inputs

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. Right-click the tunnel and select **Enable Indexing** or **Disable Indexing** from the shortcut menu to toggle the state of the tunnel.

Array Inputs

If you enable auto-indexing on an array wired to a For Loop input terminal, LabVIEW sets the count terminal to the array size so you do not need to wire the count terminal. Because you can use For Loops to process arrays one element at a time, LabVIEW enables auto-indexing by default for every array you wire to a For Loop. You can disable auto-indexing if you do not need to process arrays one element at a time.

In Figure 5-7, the For Loop executes a number of times equal to the number of elements in the array. Normally, if the count terminal of the For Loop is not wired, the run arrow is broken. However, in this case the run arrow is not broken.

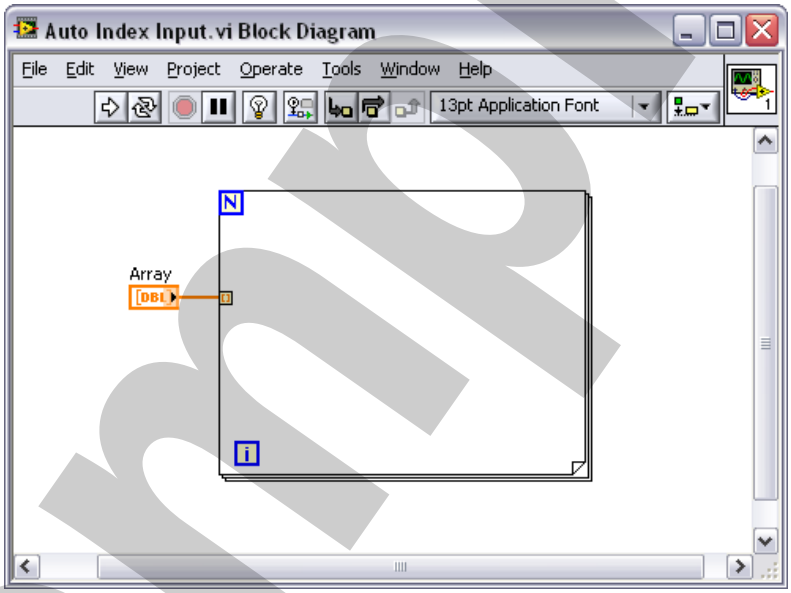


Figure 5-7. Array Used to Set For Loop Count

If you enable auto-indexing for more than one tunnel or if you wire the count terminal, the actual number of iterations becomes the smaller of the choices. For example, if two auto-indexed arrays enter the loop, with 10 and 20 elements respectively, and you wire a value of 15 to the count terminal, the loop still only executes 10 times, indexing all elements of the first array but only the first 10 elements of the second array.

Array Outputs

When you auto-index an array output tunnel, the output array receives a new element from every iteration of the loop. Therefore, auto-indexed output arrays are always equal in size to the number of iterations.

The wire from the output tunnel to the array indicator becomes thicker as it changes to an array at the loop border, and the output tunnel contains square brackets representing an array, as shown Figure 5-8.

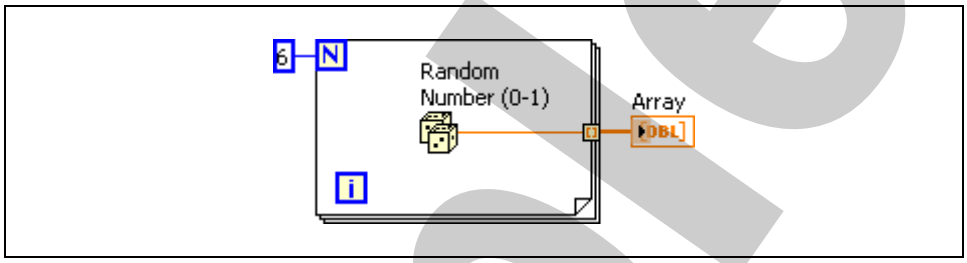


Figure 5-8. Auto-Indexed Output

Right-click the tunnel at the loop border and select **Enable Indexing** or **Disable Indexing** from the shortcut menu to enable or disable auto-indexing. Auto-indexing for While Loops is disabled by default.

For example, disable auto-indexing if you need only the last value passed out of the tunnel.

Creating Two-Dimensional Arrays

You can use two For Loops, nested one inside the other, to create a 2D array. The outer For Loop creates the row elements, and the inner For Loop creates the column elements, as shown in Figure 5-9.

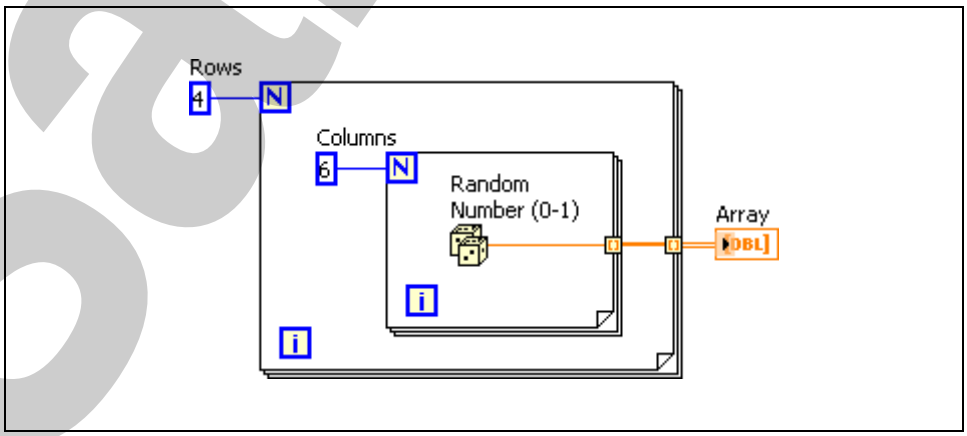


Figure 5-9. Creating a 2D Array

B. Clusters

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

Bundling several data elements into clusters eliminates wire clutter on the block diagram and reduces the number of connector pane terminals that subVIs need. The connector pane has, at most, 28 terminals. If your front panel contains more than 28 controls and indicators that you want to pass to another VI, group some of them into a cluster and assign the cluster to a terminal on the connector pane.

Most clusters on the block diagram have a pink wire pattern and data type terminal. Error clusters have a dark yellow wire pattern and data type terminal. Clusters of numeric values, sometimes referred to as points, have a brown wire pattern and data type terminal. You can wire brown numeric clusters to Numeric functions, such as Add or Square Root, to perform the same operation simultaneously on all elements of the cluster.

Order of Cluster Elements

Although cluster and array elements are both ordered, you must unbundle all cluster elements at once using the Unbundle function. You can use the Unbundle By Name function to unbundle cluster elements by name. If you use the Unbundle by Name function, each cluster element must have a label. Clusters also differ from arrays in that they are a fixed size. Like an array, a cluster is either a control or an indicator. A cluster cannot contain a mixture of controls and indicators.

Creating Cluster Controls and Indicators

Create a cluster control or indicator on the front panel by adding a cluster 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, array, or cluster control or indicator, into the cluster shell.

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

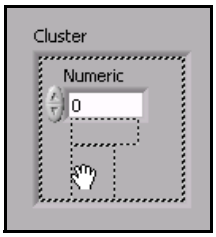


Figure 5-10. Creation of a Cluster Control

Figure 5-11 is an example of a cluster containing three controls: a string, a Boolean switch, and a numeric. A cluster is either a control or an indicator; it cannot contain a mixture of controls and indicators.

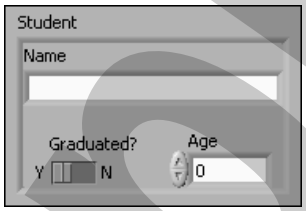


Figure 5-11. Cluster Control Example

Creating Cluster Constants

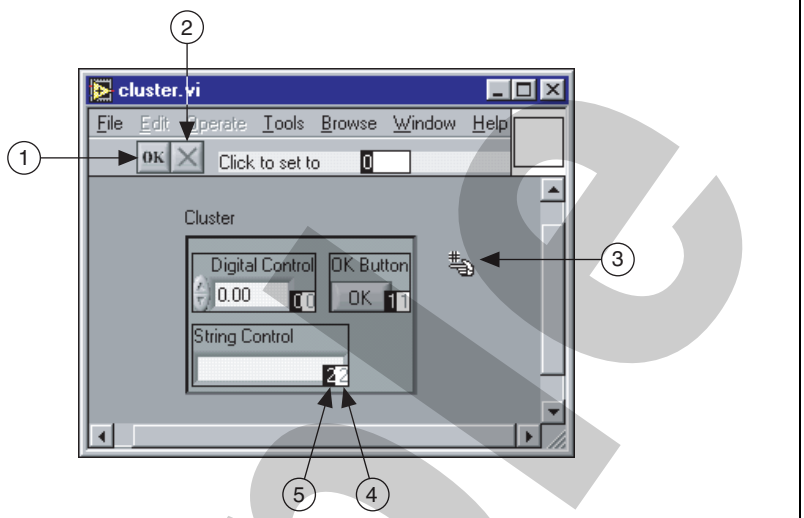
To create a cluster constant on the block diagram, select a cluster constant on the **Functions** palette, place the cluster shell on the block diagram, and place a string constant, numeric constant, a Boolean constant, or cluster constant in the cluster shell. You can use a cluster constant to store constant data or as a basis for comparison with another cluster.

If you have a cluster control or indicator on the front panel window and you want to create a cluster constant containing the same elements on the block diagram, you can either drag that cluster from the front panel window to the block diagram or right-click the cluster on the block diagram and select **Create»Constant** from the shortcut menu.

Cluster Order

Cluster elements have a logical order unrelated to their position in the shell. The first object you place in the cluster is element 0, the second is element 1, and so on. If you delete an element, the order adjusts automatically. 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.

The toolbar and cluster change, as shown in Figure 5-12.



1	Confirm Button	3	Cluster Order Cursor	5	New Order
2	Cancel Button	4	Current Order		

Figure 5-12. Reordering a Cluster

The white box on each element shows its current place in the cluster order. The black box shows the new place in the order for an element. To set the order of a cluster element, enter the new order number in the **Click to set to** text box and click the element. The cluster order of the element changes, and the cluster order of other elements adjusts. Save the changes by clicking the **Confirm** button on the toolbar. Revert to the original order by clicking the **Cancel** button.

Using Cluster Functions

Use the Cluster functions to create and manipulate clusters. For example, you can perform tasks similar to the following:

- Extract individual data elements from a cluster.
- Add individual data elements to a cluster.
- Break a cluster out into its individual data elements.

Use the Bundle function to assemble a cluster, use the Bundle function and Bundle by Name function to modify a cluster, and use the Unbundle function and the Unbundle by Name function to disassemble clusters.

You also can place the Bundle, Bundle by Name, Unbundle, and Unbundle by Name functions on the block diagram by right-clicking a cluster terminal on the block diagram and selecting **Cluster, Class & Variant Palette** from the shortcut menu. The Bundle and Unbundle functions automatically

contain the correct number of terminals. The Bundle by Name and Unbundle by Name functions appear with the first element in the cluster. Use the Positioning tool to resize the Bundle by Name and Unbundle by Name functions to show the other elements of the cluster.

Assembling Clusters

Use the Bundle function to assemble a cluster from individual elements or to change the values of individual elements in an existing cluster without having to specify new values for all elements. Use the Positioning tool to resize the function or right-click an element input and select **Add Input** from the shortcut menu.

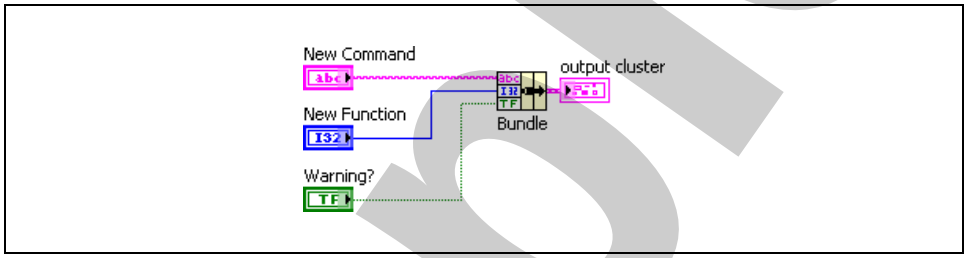


Figure 5-13. Assembling a Cluster on the Block Diagram

Modifying a Cluster

If you wire the cluster input, you can wire only the elements you want to change. For example, the Input Cluster shown in Figure 5-14 contains three controls.

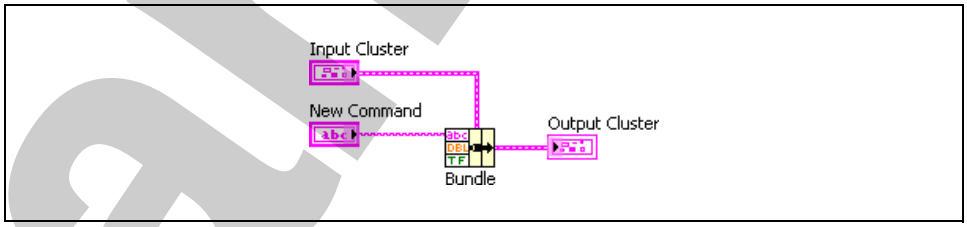


Figure 5-14. Bundle Used to Modify a Cluster

If you know the cluster order, you can use the Bundle function to change the **Command** value by wiring the elements shown in Figure 5-14.

You can also use the Bundle by Name function to replace or access labeled elements of an existing cluster. The Bundle by Name function works like the Bundle function, but instead of referencing cluster elements by their cluster order, it references them by their owned labels. You can access only elements with owned labels. The number of inputs does not need to match the number of elements in **output cluster**.

Use the Operating tool to click an input terminal and select an element from the pull-down menu. You also can right-click the input and select the element from the **Select Item** shortcut menu.

In Figure 5-15, you can use the Bundle by Name function to update the values of **Command** and **Function** with the values of **New Command** and **New Function**.

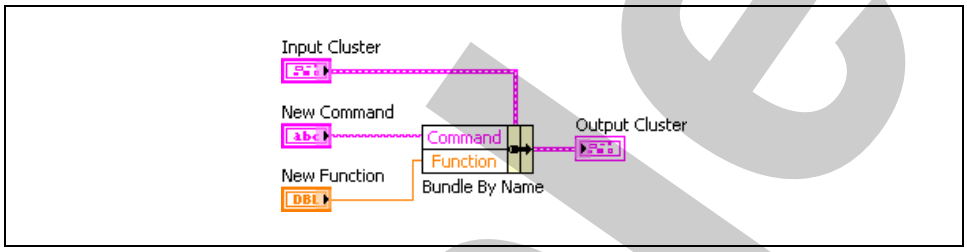


Figure 5-15. Bundle By Name Used to Modify a Cluster

Use the Bundle by Name function for data structures that might change during development. If you add a new element to the cluster or modify its order, you do not need to rewire the Bundle by Name function because the names are still valid.

Disassembling Clusters

Use the Unbundle function to split a cluster into its individual elements.

Use the Unbundle by Name function to return the cluster elements whose names you specify. The number of output terminals does not depend on the number of elements in the input cluster.

Use the Operating tool to click an output terminal and select an element from the pull-down menu. You also can right-click the output terminal and select the element from the **Select Item** shortcut menu.

For example, if you use the Unbundle function with the cluster in Figure 5-16, it has four output terminals that correspond to the four controls in the cluster. You must know the cluster order so you can associate the correct Boolean terminal of the unbundled cluster with the corresponding switch in the cluster. In this example, the elements are ordered from top to bottom starting with element 0. If you use the Unbundle by Name function, you can have an arbitrary number of output terminals and access individual elements by name in any order.

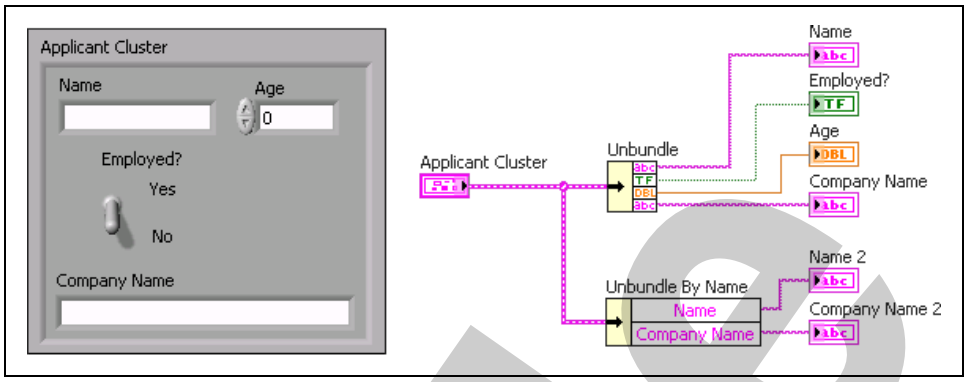


Figure 5-16. Unbundle and Unbundle By Name

Error Clusters

LabVIEW contains a custom cluster called the error cluster. LabVIEW uses error clusters to pass error information. An error cluster contains the following elements:

- **status**—Boolean value that reports TRUE if an error occurred.
- **code**—32-bit signed integer that identifies the error numerically.
- **source**—String that identifies where the error occurred.

For more information about using error clusters, refer to Lesson 3, *Troubleshooting and Debugging VIs*, of this manual and the *Handling Errors* topic of the *LabVIEW Help*.

C. Type Definitions

You can use type definitions to define custom arrays and clusters.

Custom Controls

Use custom controls and indicators to extend the available set of front panel objects. You can create custom user interface components for an application that vary cosmetically from built-in LabVIEW controls and indicators. You can save a custom control or indicator you created in a directory or LLB and use the custom control or indicator on other front panels. You also can create an icon for the custom control or indicator and add it to the **Controls** palette.

Refer to the *Creating Custom Controls, Indicators, and Type Definitions* topic of the *LabVIEW Help* for more information about creating and using custom controls and type definitions.

Use the Control Editor window to customize controls and indicators. For example, you can change the size, color, and relative position of the elements of a control or indicator and import images into the control or indicator.

You can display the Control Editor window in the following ways:

- Right-click a front panel control or indicator and select **Advanced» Customize** from the shortcut menu.
- Use the Positioning tool to select a front panel control or indicator and select **Edit»Customize Control**.
- Use the **New** dialog box.

The Control Editor appears with the selected front panel object in its window. The Control Editor has two modes, edit mode and customize mode.

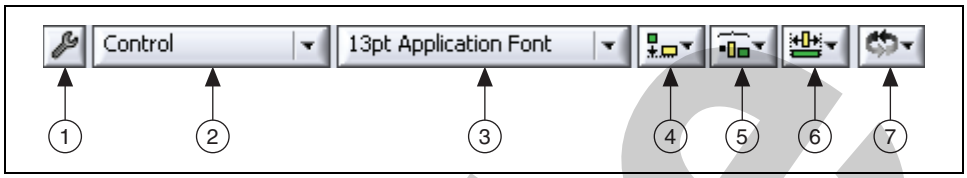
The Control Editor window toolbar indicates whether you are in edit mode or in customize mode. The Control Editor window opens in edit mode. Click the **Change to Customize Mode** button to change to customize mode. To change back to edit mode, click the **Change to Edit Mode** button. You also can switch between modes by selecting **Operate»Change to Customize Mode** or **Operate»Change to Edit Mode**.

Use edit mode to change the size or color of a control or indicator and to select options from its shortcut menu, just as you do in edit mode on a front panel.

Use customize mode to make extensive changes to controls or indicators by changing the individual parts of a control or indicator.

Edit Mode

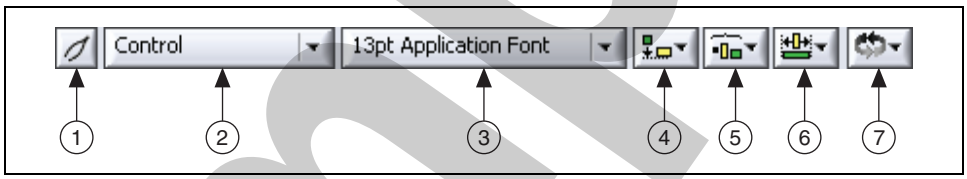
In the edit mode, you can right-click the control and manipulate its settings as you would in the LabVIEW programming environment.



1	Edit Mode	5	Distribute Objects
2	Type Definition Status	6	Resize Objects
3	Text	7	Reorder Objects
4	Align Objects		

Customize Mode

In the customize mode, you can move the individual components of the control around with respect to each other. For a listing of what you can manipulate in customize mode, select **Window»Show Parts Window**.



1	Customize Mode	5	Distribute Objects
2	Type Definition Status	6	Resize Objects
3	Text	7	Reorder Objects
4	Align Objects		

One way to customize a control is to change its type definition status. You can save a control as a control, a type definition, or a strict type definition, depending on the selection visible in the **Type Def. Status** ring. The control option is the same as a control you would select from the **Controls** palette. You can modify it in any way you need to, and each copy you make and change retains its individual properties.

Saving Custom Controls

After creating a custom control, you can save it for use later. By default, controls saved on disk have a `.ctl` extension.

You also can use the Control Editor to save controls with your own default settings. For example, you can use the Control Editor to modify the defaults of a waveform graph, save it, and later recall it in other VIs.

Type Definitions

Use type definitions and strict type definitions to link 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, which is useful if you use the same custom control or indicator in several VIs.

When you place a custom control or indicator in a VI, no connection exists between the custom control or indicator you saved and the instance of the custom control or indicator in the VI. Each instance of a custom control or indicator is a separate, independent copy. Therefore, changes you make to a custom control or indicator file do not affect VIs already using that custom control or indicator. If you want to link instances of a custom control or indicator to the custom control or indicator file, save the custom control or indicator as a type definition or strict type definition. All instances of a type definition or a strict type definition link to the original file from which you created them.

When you save a custom control or indicator as a type definition or strict type definition, any data type changes you make to the type definition or strict type definition affect all instances of the type definition or strict type definition in all the VIs that use it. Also, cosmetic changes you make to a strict type definition affect all instances of the strict type definition on the front panel.

Type definitions identify the correct data type for each instance of a custom control or indicator. When the data type of a type definition changes, all instances of the type definition automatically update. In other words, the data type of the instances of the type definition change in each VI where the type definition is used. However, because type definitions identify only the data type, only the values that are part of the data type update. For example, on numeric controls, the data range is not part of the data type. Therefore, type definitions for numeric controls do not define the data range for the instances of the type definitions. Also, because the item names in ring controls do not define the data type, changes to ring control item names in a type definition do not change the item names in instances of the type definition. However, if you change the item names in the type definition for an enumerated type control, the instances update because the item names are part of the data type. An instance of a type definition can have its own unique caption, label, description, tip strip, default value, size, color, or style of control or indicator, such as a knob instead of a slide.

If you change the data type in a type definition, LabVIEW converts the old default value in instances of the type definition to the new data type, if possible. LabVIEW cannot preserve the instance default value if the data type changes to an incompatible type, such as replacing a numeric control or indicator with a string control or indicator. When the data type of a type definition changes to a data type incompatible with the previous type definition, LabVIEW sets the default value of instances to the default value you specify in the `.ctl` file. If you do not specify a default value, LabVIEW uses the default value for the data type. For example, if you change a type definition from a numeric to a string type, LabVIEW replaces any default values associated with the old numeric data type with empty strings.

Strict Type Definitions

A strict type definition forces everything about an instance to be identical to the strict type definition, except the caption, label, description, tip strip, and default value. As with type definitions, the data type of a strict type definition remains the same everywhere you use the strict type definition. Strict type definitions also define other values, such as range checking on numeric controls and the item names in ring controls. The only VI Server properties available for strict type definitions are those that affect the appearance of the control or indicator, such as Visible, Disabled, Key Focus, Blinking, Position, and Bounds.

You cannot prevent an instance of a strict type definition from automatically updating unless you remove the link between the instance and the strict type definition.

Type definitions and strict type definitions create a custom control using a cluster of many controls. If you need to add a new control and pass a new value to every subVI, you can add the new control to the custom control cluster. This substitutes having to add the new control to the front panel of each subVI and making new wires and terminals.

Self-Review: Quiz

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 Figure 5-17. What is the value of the Iterations indicator after running this VI?

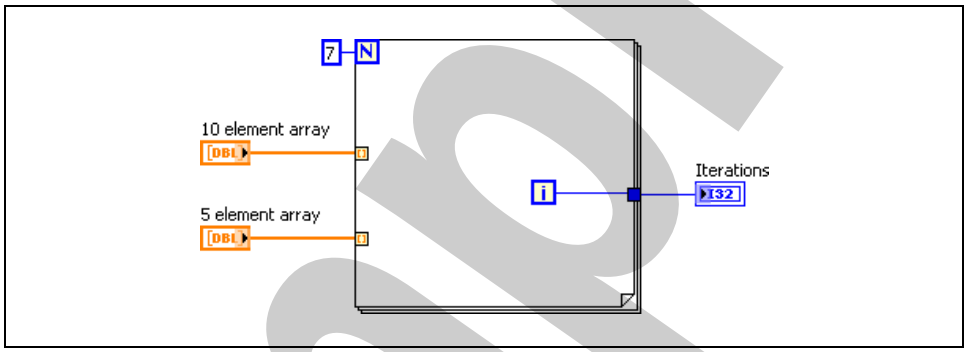


Figure 5-17. What is the Value of the Iteration Indicator?

3. You customize a control, select **Control** from the **Control Type** pull-down menu, and save the control as a `.ctl` file. You then use an instance of the custom control on your front panel window. If you open the `.ctl` file and modify the control, does the control on the front panel window change?
   a. Yes
   b. No

4. You are inputting data that represents a circle. The circle data includes three double precision numerics: x position, y position and radius. In the future, you might need to expand all instances of the circle data to include the color of the circle, represented as an integer. How should you represent the circle on your front panel window?
   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.

Self-Review: Quiz 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?

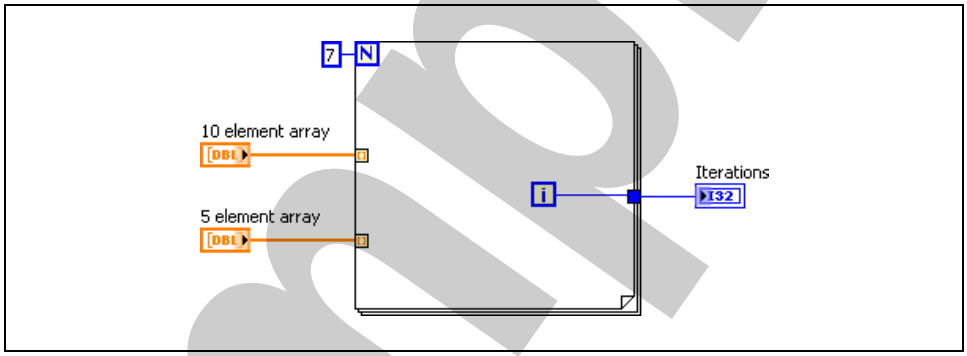


Figure 5-18. What is the value of the Iteration indicator?

   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. You customize a control, select **Control** from the **Control Type** pull-down menu, and save the control as a `.ctl` file. You then use an instance of the custom control on your front panel window. If you open the `.ctl` file and modify the control, does the control on the front panel window change?

   a. Yes

   b. No

4. You are inputting data that represents a circle. The circle data includes three double precision numerics: x position, y position and radius. In the future, you might need to expand all instances of the circle data to include the color of the circle, represented as an integer. How should you represent the circle on your front panel window?
   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.

Notes

Notes