LabVIEW

PRENTICE HALL

LabVIEW For Everyone

Graphical Programming Made Easy and Fun

JEFFREY TRAVIS
JIM KRING

THIRD EDITION

LabVIEW for Everyone

Third Edition

The **Invert** option has a different effect depending on the **Compound Arithmetic** function's mode of operation (as selected from the **Change Mode** pop-up submenu). Table 7.1 shows the effect of the invert option, applied to an *input*, for each of the **Compound Arithmetic** function's modes of operation.

Table 7.1 *The Effect of an Inverted Input for Each Compound Arithmetic Mode*

Mode	Non-Inverted Input	Inverted Input
Add	add input	add (0–input)
Multiply	multiply input	multiply (1/input)
AND	AND input	AND (NOT input)
OR	OR input	OR (NOT input)
XOR	XOR input	XOR (NOT input)

When the **Invert** option is applied to an *output*, the function is similar. Table 7.2 shows the effect of the invert option, applied to an *output*, for each of the **Compound Arithmetic** function's modes of operation.

Table 7.2 *The Effect of an Inverted Output for Each Compound Arithmetic Mode*

Mode	Non-Inverted Output	Inverted Output
Add	result	0–result
Multiply	result	1/result
AND	result	NOT result (NAND)
OR	result	NOT result (NOR)
XOR	result	NOT result (NOT XOR)

In order to make this as clear as possible, we have created a collection of equivalent operations, shown in Figure 7.40. This is meant only to help you see the underlying principles. (Don't worry; this won't be on the quiz.)

For a simpler example of some practical ways to use the compound arithmetic node, see Figure 7.41. The two equivalent operations on the left show how you can use the compound arithmetic node for combined addition and subtraction. The two equivalent operations on the right show how you can use the compound arithmetic node for combined multiplication and division.

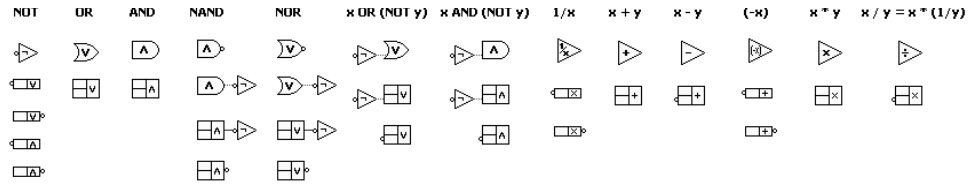

Figure 7.40
Equivalent standard and compound arithmetic operations (in columns)

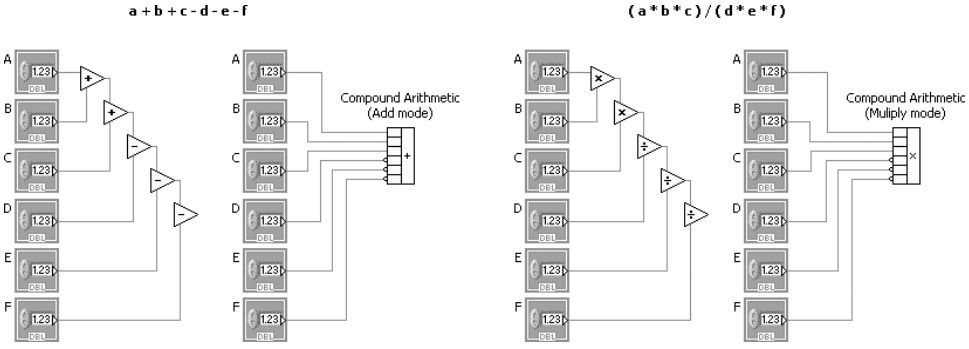

Figure 7.41
Equivalent standard and compound arithmetic operations (in columns)

A Word About Boolean Arithmetic

LabVIEW's Boolean arithmetic functions—**And, Or, Not, Exclusive Or, Not Exclusive Or, Not And,** and **Not Or**—can be very powerful. You will see occasional Boolean arithmetic throughout this book. If you've never seen it before, we recommend reading about it in a good logic or digital design book. But just as a refresher, we'll mention a few basics. If you can't remember which function does what, use the Help window!

Not is probably the easiest to describe, because it simply inverts the input value. If the input value is TRUE, **Not** will output FALSE; if the input is FALSE, **Not** returns TRUE.

The **And** function outputs a TRUE only if all inputs are TRUE.

The **Or** function outputs a TRUE if at least one input is TRUE.

The **And** and **Or** functions have the following outputs, given the inputs shown:

FALSE **And** FALSE = FALSE	FALSE **Or** FALSE = FALSE
TRUE **And** FALSE = FALSE	TRUE **Or** FALSE = TRUE
FALSE **And** TRUE = FALSE	FALSE **Or** TRUE = TRUE
TRUE **And** TRUE = TRUE	TRUE **Or** TRUE = TRUE

All About Clusters

Now that you've got the concept of arrays under your belt, clusters should be easy. Like an array, a *cluster* is a data structure that groups data. However, unlike an array, a cluster can group data of different types (i.e., numeric, Boolean, etc.); it is analogous to a *struct* in C or the data members of a *class* in C++ or Java. A cluster may be thought of as a *bundle* of wires, much like a telephone cable. Each wire in the cable represents a different element of the cluster. Because a cluster has only one "wire" in the block diagram (even though it carries multiple values of different data types), clusters reduce wire clutter and the number of connector terminals that subVIs need. You will find that the cluster data type appears frequently when you plot your data on graphs and charts. Figures 7.42 and 7.43 illustrate the concepts of bundling and unbundling clusters of element data.

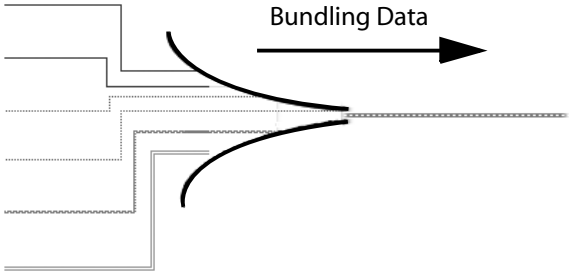

Figure 7.42
A conceptual illustration showing the bundling of data elements to create a cluster

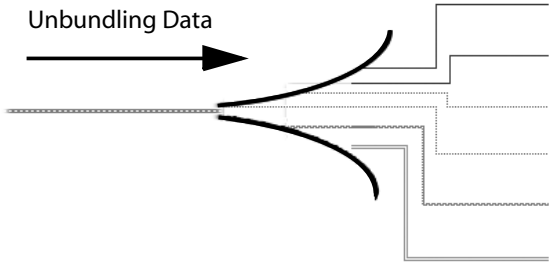

Figure 7.43
A conceptual illustration showing the unbundling of data elements from a cluster

You can access cluster elements by *unbundling* them all at once or by indexing one at a time, depending on the function you choose; each method has its place (see Figure 7.43). Think of unbundling a cluster as unwrapping a telephone cable and having access to the different-colored wires. Unlike arrays, which can change size dynamically, clusters have a fixed size, or a fixed number of wires in them.

You can connect cluster terminals with a wire *only* if they have exactly the same type; in other words, both clusters must have the same number of elements, and corresponding elements must match in both data type and order. The principle of polymorphism applies to clusters as well as arrays, as long as the data types match.

You will often see clusters used in error handling. Figure 7.44 shows the error clusters, **Error In.ctl** and **Error Out.ctl,** used by LabVIEW VIs to pass a record of errors among multiple VIs in a block diagram (for example, many of the data acquisition and file I/O VIs have error clusters built into them). These error clusters are so frequently used that they appear in the **Modern>>Array, Matrix & Cluster** subpalette of the **Controls** palette for easy access.

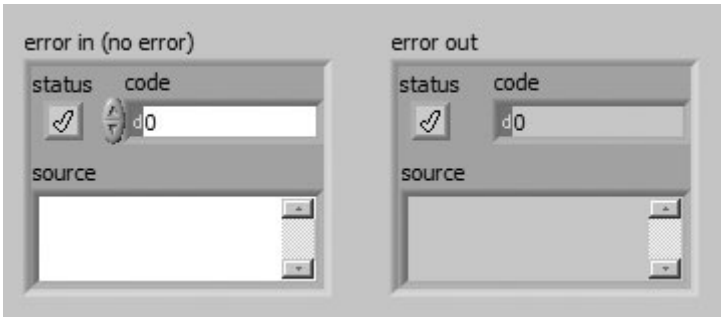

Figure 7.44
Front panel containing Error In and Error Out clusters, which are used for error handling

Creating Cluster Controls and Indicators

Create a cluster by placing a **Cluster** shell (**Modern>>Array, Matrix & Cluster** subpalette of the **Controls** palette) on the front panel. You can then place any front panel objects inside the cluster. Like arrays, you can deposit objects directly inside when you pull them off of the **Controls** palette, or you can drag an existing object into a cluster. *Objects inside a cluster must be all controls or all indicators*. You cannot combine both controls and indicators inside the same cluster, because the cluster itself must be one or the other. The cluster will be a control or indicator based on the status of the first object you place inside. Resize the cluster with the Positioning tool if necessary. Figure 7.45 shows a cluster with four controls.

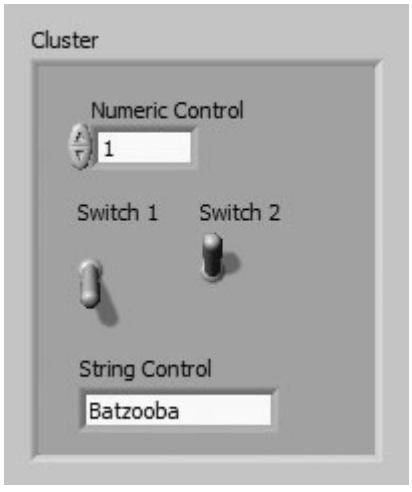

Figure 7.45
Front panel with a cluster containing four controls

You can create block diagram cluster constants in a similar two-step manner.

If you want your cluster to conform exactly to the size of the objects inside it, pop up on the *border* (NOT inside the cluster) and choose an option in the **Autosizing** menu.

Cluster Order

Cluster elements have a logical order unrelated to their position within the shell. The first object placed in the cluster is element zero, the second is element one, and so on. If you delete an element, the order adjusts automatically. *You must keep track of your cluster order if you want to connect your cluster to another cluster—the order and data types must be identical*. Also, if you choose to unbundle the cluster all at once, you'll want to know which value corresponds to which output on the cluster function (more about unbundling will be discussed later in this chapter in the section, "Unbundling Your Clusters").

Change the order within the cluster by popping up on the cluster *border* and choosing **Reorder Controls in Cluster . . .** from the pop-up menu. A new set of buttons appears in the Toolbar, and the cluster appearance changes, as shown in Figure 7.46.

The white boxes on the elements show their current places in the cluster order. The black boxes show the new places. Clicking on an element with the cluster order cursor sets the element's place in the cluster order to the number displayed on the Toolbar. You can type a new number into that field before you click on an object.


Revert Button

If you don't like the changes you've made, revert to the old order by clicking on the Revert button. When you have the order the way you want, you can set it and

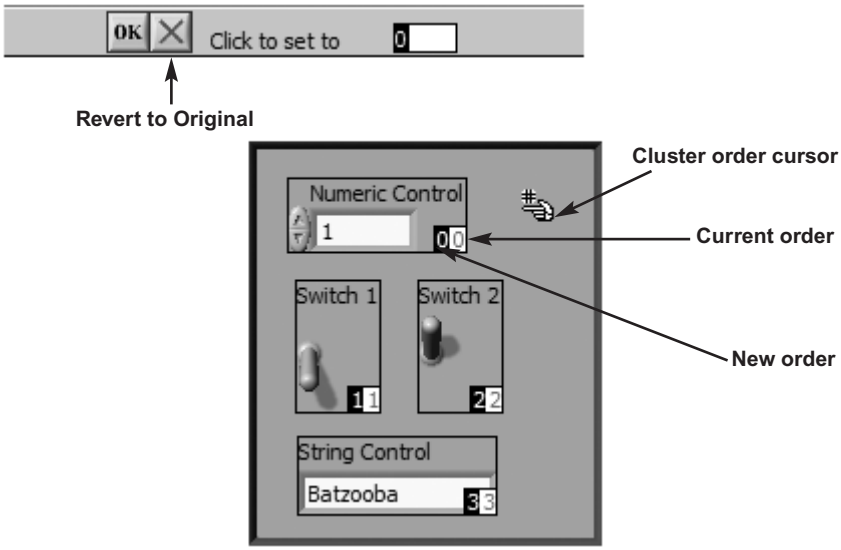

Figure 7.46
Setting the order of cluster elements


OK Button

return to your regular front panel by clicking on the OK button and exiting the cluster order edit mode.

You can quickly inspect the order of any cluster using the Context Help window. Simply move the Wiring tool over the wire, and the Context Help window will display the data type of the wire—the cluster elements are displayed in order, from top to bottom, as shown in Figure 7.47. (Show the Context Help window by selecting **Help>>Show Context Help** from the menu or by pressing the shortcut key <ctrl-H> [Windows], <command-H> [Mac OS X], or <meta-H> [Linux].)

Using Clusters to Pass Data to and from SubVIs

The connector pane of a VI can have a maximum of 28 terminals. You probably don't want to pass information to all 28 terminals when calling a subVI anyway—the wiring can be very tedious and you can easily make a mistake. By bundling a number of controls or indicators into a cluster, you can use a single terminal and wire to pass several values into or out of the subVI. You can use clusters to work around the 28-terminal limit for the connector or just enjoy simplified wiring with fewer (and therefore larger and easier to see) terminals.


Bundle Function

The **Bundle** function (**Programming>>Cluster & Variant** palette) assembles individual components into a new cluster or allows you to replace elements in an