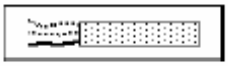
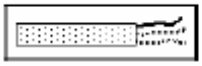
A. Clusters

A cluster is a data structure that combines one or more data components into a new data type. The components that form a cluster may even have different data types. For example, you can combine Boolean, string, and integer data types into a new data type. A cluster is analogous to a *record* in Pascal or a *struct* in C.

On the block diagram, a wire that carries the data stored in a cluster may be thought of as a bundle of smaller wires, much like a telephone cable. Each wire in the cable represents a different component of the cluster. Because a cluster constitutes only one "wire" in the block diagram, clusters reduce wire clutter and the number of connector terminals that subVIs need.

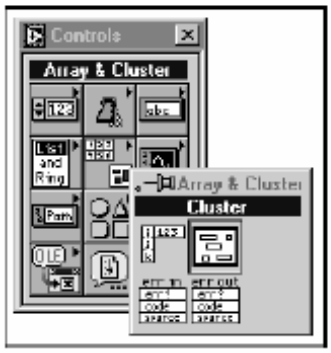


You "unbundle" the cluster in the diagram to access its components. You may think of unbundling a cluster as unwrapping a telephone cable and accessing the individual wires within the cable.



Creating Cluster Controls and Indicators On the Front Panel

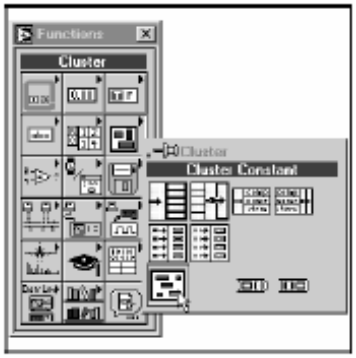
You create cluster controls and indicators on a VI's front panel by placing a *cluster shell* in the Panel window. To place an empty cluster shell on the panel, choose **Array & Cluster** » **Cluster** from the **Controls** palette. Then, click in the Panel window to place the cluster. You can adjust the size of the cluster shell by holding down the mouse button and dragging the cursor when placing the cluster shell on the panel.



You can place any objects inside the cluster that you normally place in the Panel window. You can deposit objects directly inside the cluster by popping up inside the cluster, or you can drag an 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. For example, if you drop an indicator into a cluster containing controls, the indicator changes to a control. The cluster assumes the data direction (control or indicator) of the first object you place inside the cluster. A cluster with four controls is shown below.

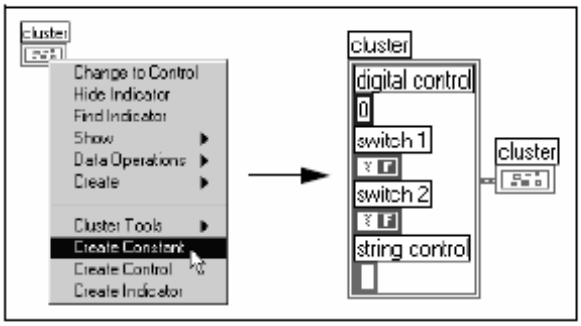
Creating Cluster Constants on the Block Diagram

To create a cluster constant on the block diagram, you can use the same technique as you used on the front panel. From the Diagram window, choose **Cluster** » **Cluster Constant** from the **Functions** palette to create the cluster shell. Then, place other constants of the appropriate data type within the cluster shell.



If you have a cluster control or indicator on the front panel, and would like to create a cluster constant containing the same components in the diagram,

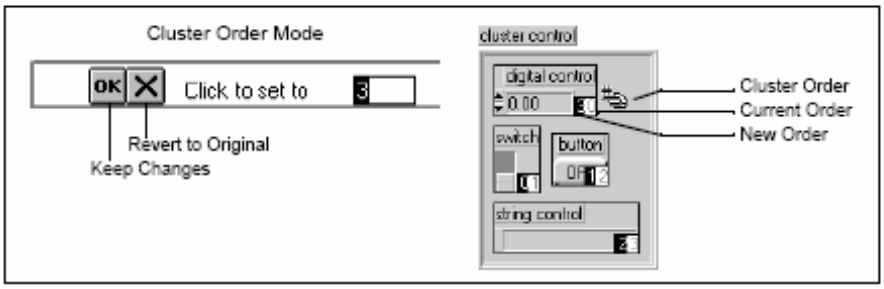
you can pop up on its terminal and select **Create Constant** from the pop-up menu. This technique is a great time-saver. If you use this method on a cluster indicator, the constant is wired to the indicator automatically.



Cluster Order

When LabVIEW manipulates clusters of data, not only are the data types of the individual components within the cluster important, but also the order of the components in the cluster. Cluster components have a logical order unrelated to their position within the shell. The first object placed in the cluster shell is component 0, the second is component 1, and so on. If you delete a component, the order adjusts automatically.

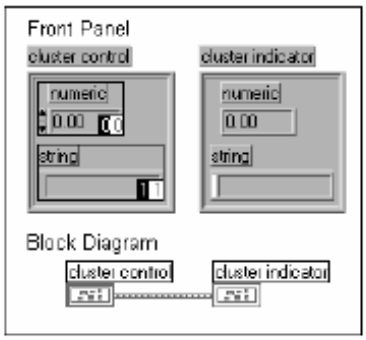
You can change the order of the objects within the cluster by popping up on the cluster *border* and choosing **Cluster Order...** from the pop-up menu. A new set of buttons replaces the toolbar, and the cluster appearance changes as shown below. The white box on each component shows its current place in the cluster order. The black box shows a component's new place in the order.



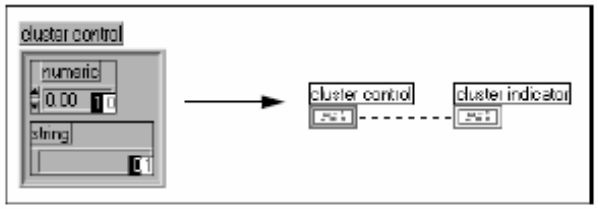
Revert to Original button

To set a cluster component to a particular index in the cluster order, first type the desired order number into the "Click to set to" field. Then click on the desired component. You will notice that the component's cluster order index changes. You will also notice that the cluster order indices of the other components adjust automatically. To save your changes, click on the OK button in the palette. To revert to the original settings, click on the Revert to Original button. You use this technique to set the cluster order for constants on both the front panel and the block diagram.

The example shown below illustrates the importance of cluster order. The front panel contains two simple clusters. In the first cluster, component 0 is a numeric control, and component 1 is a string control. In the second cluster, component 0 is a numeric indicator, and component 1 is a string indicator. The cluster control wires to the cluster indicator on the block diagram.

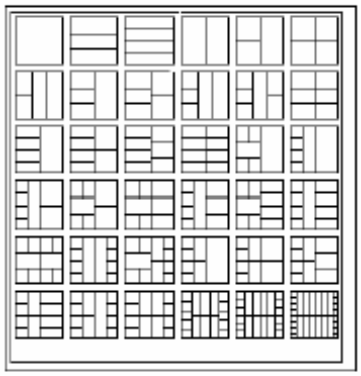


However, if you change the cluster order of the indicator so the string indicator is component 0 and the numeric is component 1, the wire connecting the control to the indicator is broken. If you try to run the VI, you get an error message stating that there is a type conflict because the data types do not match.

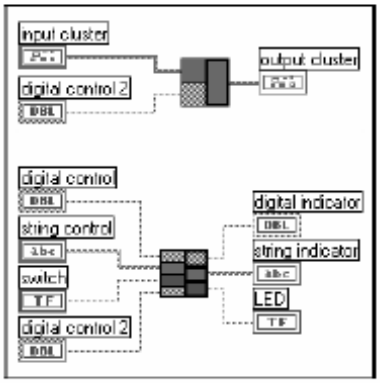


Using Clusters to Pass Data to and from SubVIs

As shown in the figure below, the connector pane of a VI can have a maximum of 28 terminals. When you use a connector pane that has a large number of terminals, the terminals are very small. With such small terminals, wiring errors are more likely.



By bundling a number of controls into a cluster, and passing the cluster to the subVI, you can overcome the 28-terminal limit of the connector pane. One cluster control uses one terminal on the connector pane, but that cluster can contain several controls. Similarly, one terminal assigned to a cluster indicator can pass several outputs from the subVI. Because your subVI uses clusters containing several items each, you can use fewer, and therefore larger, terminals on the connector pane. This will make for cleaner wiring on the diagram.



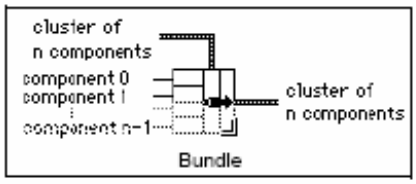
B. Cluster Functions

LabVIEW uses several functions to manipulate clusters. We will study the **Bundle** and **Bundle by Name** functions, which assemble and modify clusters, and the **Unbundle** and **Unbundle by Name** functions, which disassemble clusters.

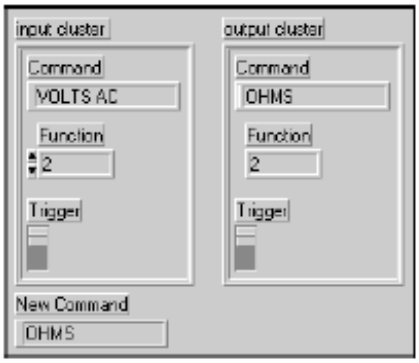
Assembling Clusters


Positioning tool

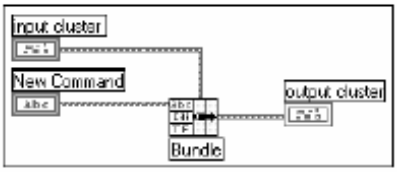
The **Bundle** function (**Cluster** palette) assembles individual components into a single cluster or replaces components within an existing cluster. The topmost component wired to the **Bundle** function is component 0 in the cluster; the component beneath it is component 1; and so on. You can increase the number of inputs by resizing the function with the Positioning tool or by popping up on the icon and selecting **Add Input** from the pop-up menu. If the **cluster of n components** terminal is wired, the number of input terminals to the **Bundle** function must match the number of items in the input cluster.



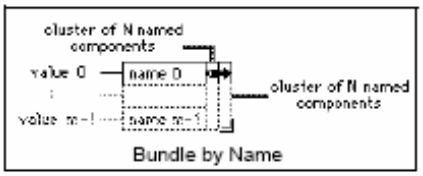
When you use the **cluster of n components** terminal, you do not need to wire data to every input terminal of the function. Instead, you can wire data only to the items you want to change. For example, consider the cluster shown below, which contains three controls—a string labeled "Command," a numeric labeled "Function," and a Boolean labeled "Trigger."



You can use the **Bundle** function to change the string value by wiring the components as shown below. You must know the cluster order to do this correctly.



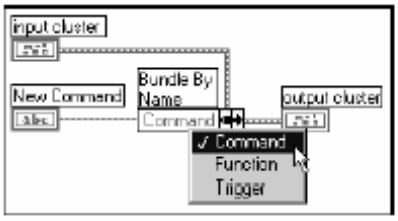
The **Bundle by Name** function replaces components in an existing cluster. **Bundle by Name** works similarly to the **Bundle** function, but instead of referencing cluster components by their cluster order, it references them by their owned labels. You cannot access components in the input cluster (connected to the **cluster of N named components** input terminal) that do not have owned labels.



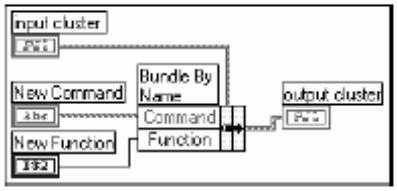

Operating tool

You select a component by clicking on an input terminal using the Operating tool and selecting a name from the list of components in the cluster. Also, you can pop up on the input terminal and select the component from the **Select Item** menu. Note that you *must* wire an input cluster to the **cluster of N named components** input of this function, and at least one item in the input cluster must have a name. The number of terminals on the **Bundle by Name** icon does not need to match the number of components in the input cluster.

For example, consider again the cluster control containing the "Command," "Function," and "Trigger" controls. You can use the **Bundle by Name** function to change the string value by wiring the components as shown below. To select the name "Command," you click on the left terminal of the **Bundle by Name** function using the Operating tool, and select "Command" from the list of names.



If you need to modify both "Command" and "Function," you can resize the **Bundle by Name** function as shown below.

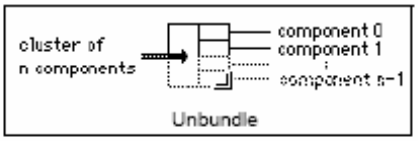


The **Bundle by Name** function is useful when working with data structures that may change during the development process. If you add a new component to the cluster or modify its order, you do not need to rewire the **Bundle by Name** function on the diagram because the names are still valid.

Disassembling Clusters


Positioning tool

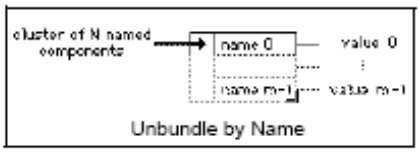
The **Unbundle** function (**Cluster** palette) splits a cluster into each of its individual components. The components are arranged from top to bottom according to the cluster order of the input cluster. You can increase the number of outputs by resizing the function with the Positioning tool or by using the pop-up menu. The number of output terminals for this function must match the number of components in the input cluster.




Operating tool

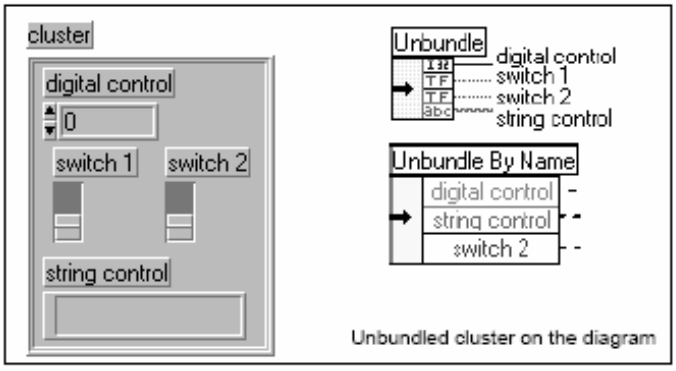
The **Unbundle by Name** function (**Cluster** palette) returns the cluster components that you reference by name. You select a component by clicking on the output terminal using the Operating tool and selecting a name from the list of components in the cluster. Also, you can pop up on an output terminal and select the component from the **Select Item** menu. Because the cluster components are referenced by name, you can access only the cluster components that have owned labels. The number of output terminals of the

Unbundle by Name function does not depend on how many components are in the input cluster.



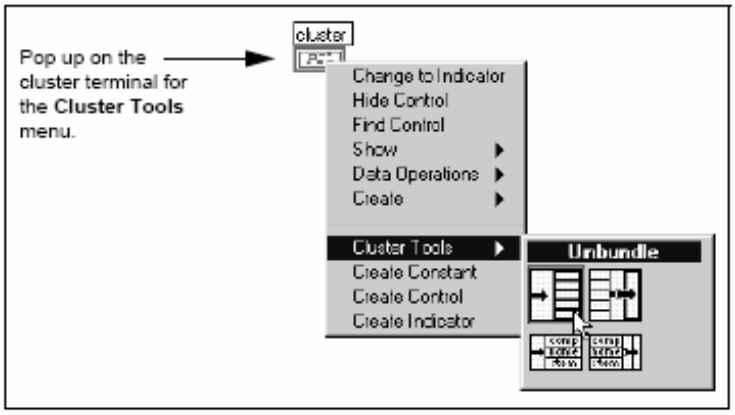
Unbundle by Name

For example, if you used the **Unbundle** function with the cluster shown below, it would have four output terminals. These four terminals correspond to the four controls inside the cluster. Notice that you need to know the cluster order so that you can associate the correct Boolean (TF) terminal of the unbundled cluster with the corresponding switch inside the cluster. In this example, the components are ordered from top to bottom starting with component 0. Notice that when you use the **Unbundle by Name** function, you can have an arbitrary number of terminals and access specific components by name in any order.



Unbundled cluster on the diagram

As shown below, you can also create the **Bundle**, **Bundle by Name**, **Unbundle**, and **Unbundle by Name** functions by popping up on a cluster terminal in the block diagram and choosing **Cluster Tools** from the pop-up menu. The **Bundle** and **Unbundle** functions will automatically contain the correct number of terminals. The **Bundle by Name** and **Unbundle by Name** functions will appear with the first component in the cluster.



C. Error Clusters

When designing applications, you should try to anticipate any errors that may occur. For example, if you acquire signals and save them to a file over a long period of time, the operation might fail at some point due to a lack of disk space. If your application requires reliable operation, you should check for errors and decide what to do when an error occurs.

LabVIEW I/O operations check for errors and return error codes. Because error handling is different for every application, LabVIEW does not perform automatic handling of all errors. For example, you may want to terminate an application if an error occurs, or you may just want to notify the user so that he or she can correct the problem.

Nearly all of the low-level I/O operations in LabVIEW incorporate an error cluster that makes error handling easier. Under this scheme, I/O VIs have both an error input and an error output. If the input contains an error, the I/O VI either does nothing or shuts down the I/O operation it manages. One advantage of this approach is that you can connect several I/O operations together so that, if an error occurs, LabVIEW will not perform subsequent I/O operations.

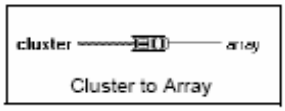
The error cluster used by most of the I/O operations—including data acquisition (DAQ), DDE, TCP, UDP, VISA, GPIB, and the File I/O functions—contains a Boolean indicating whether an error has been detected, a numeric error code, and a string used to identify the source of the error.

To make error handling easier for you, LabVIEW features several error handling functions. The **Simple Error Handler** VI takes care of the error handling that most applications need. This handler contains error information about all I/O operations in LabVIEW.

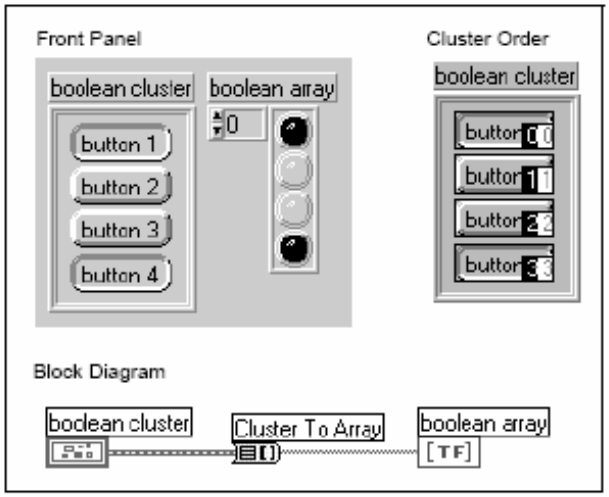
D. Cluster Conversion

You can convert a cluster to an array if *all* cluster components have the *same* data type (for example, all are Boolean or all are numeric). With this conversion, you can use array functions to process components within the cluster.

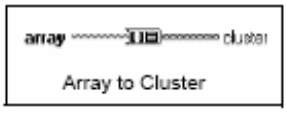
The **Cluster to Array** function (**Cluster** and **Array** palettes) converts a cluster of identically typed components to a 1D array of the same data type.



The example below shows a four-component Boolean cluster converted to a four-element Boolean array. The index of each element in the array corresponds to the logical order of the component in the cluster. For example, Button 1 (component 0) corresponds to the first element (index 0) in the array, Button 2 (component 1) to the second element (index 1), and so on.



The **Array to Cluster** function (**Cluster** and **Array** palettes) converts a 1D array to a cluster in which each component in the cluster is the same type as the array element.

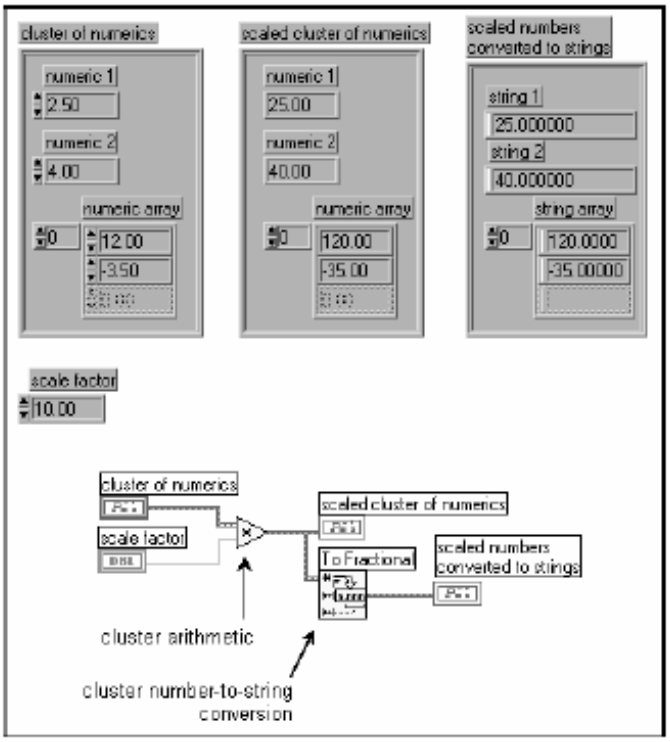


Note *You must pop up on the function icon to set the number of components in the cluster. The default number of components in the output cluster is nine.*

E. Using Polymorphism with Clusters

Most LabVIEW functions are polymorphic. As discussed in the LabVIEW Basics I course, polymorphic functions can have inputs of different types. You can use the **Online Reference** (**Help** menu) to search for a list of polymorphic functions.

Because the arithmetic functions are polymorphic, you can use them to perform computations on clusters of numerics. The string-to-number functions are also polymorphic. As shown in the following example, you use the arithmetic functions with clusters in the same way you use them with arrays of numerics. You can also use the string-to-number functions to convert a cluster of numerics to a cluster of strings.



Cluster Summary

- A cluster is a data structure that groups data, even data of different types.
- If a VI has many front panel controls and indicators that you need to associate with terminals, you may want to group them into one or more clusters and use fewer terminals.
- Objects inside a cluster must be all controls or all indicators.
- Creating a cluster on the front panel or in the diagram is a two-step process. First, create a cluster shell and then place the components inside the cluster shell.
- When working with clusters, the order of the components is critical. This is especially true if you have similar objects inside a cluster.
- The Unbundle function (Cluster palette) splits a cluster into each of its individual components.
- The Bundle function (Cluster palette) assembles individual components into a single cluster.
- The Unbundle by Name function (Cluster palette) extracts components from a cluster by name.
- The Bundle by Name function replaces components in a cluster by name.
- Error clusters are a powerful method of error handling used with nearly all of the I/O VIs in LabVIEW. These clusters pass error information from one VI to the next.
- An error handler at the end of the data flow can receive the error cluster and display error information in a dialog box.
- You can convert a cluster containing components of the same data type to an array and then use the array functions to process the cluster components. The Cluster to Array function (Cluster or Array palette) converts a cluster to a 1D array.
- The Array to Cluster function (Cluster or Array palette) converts a 1D array to a cluster. You must specify the number of components in the output cluster.
- State machine programming is very useful for user interface VIs and generates clean, simple code.
- You can use the polymorphic capabilities of LabVIEW functions with clusters. The Online Reference (Help menu) contains a list of all polymorphic functions.
- Try to place items in a cluster that logically or conceptually belong together.