

# **Lecture 3-2**

Loops and Charts,  
Arrays, Graphs, Clusters,  
Case, Sequence Structures,  
Local, Global Variables, and  
Attribute Nodes  
in Graphical Programming Language

吳文中

# Clusters

- Data structure that groups data together
- Data may be of different types
- Analogous to *struct* in C or a *record* in Pascal
- Elements must be either all controls or all indicators
- Thought of as wires bundled into a cable



# Arrays

- Collection of data elements that are of same type
- One or more dimensions, <sup>31</sup>up to 2 elements per dimension
- Elements accessed by their index; first element is index 0

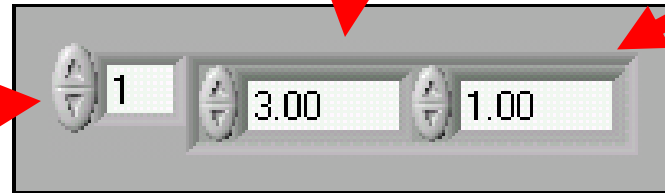
	index	0	1	2	3	4	5	6	7	8	9
10-element array											

		0	1	2	3	4	5	6
2D array	0							
	1							
	2							
	3							
	4							

Five row by seven column array of 35 elements

# Viewing Arrays on the Front Panel

The elements at index 0 are not shown because element 1 is selected in the index display.



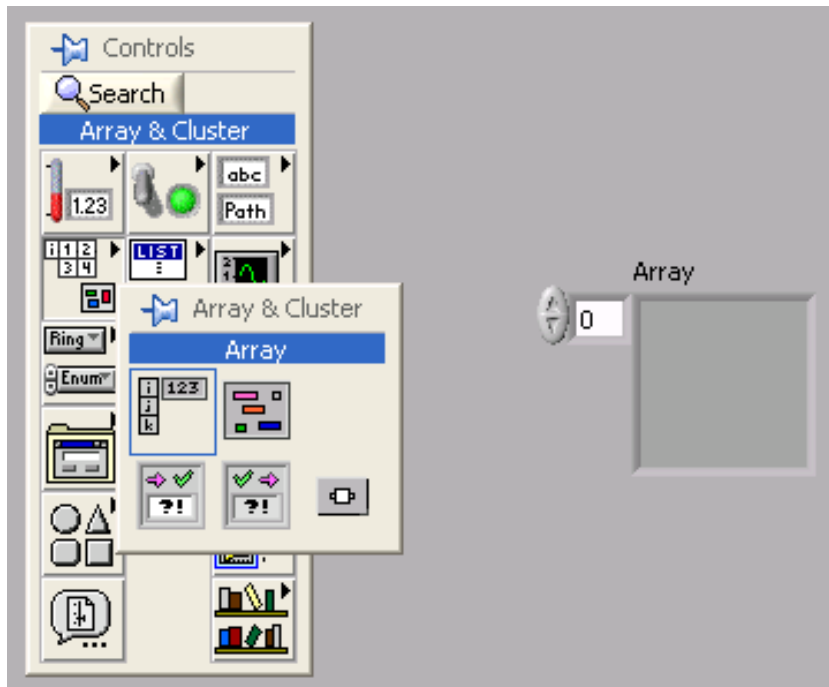
First element at index 1

Second element at index 2

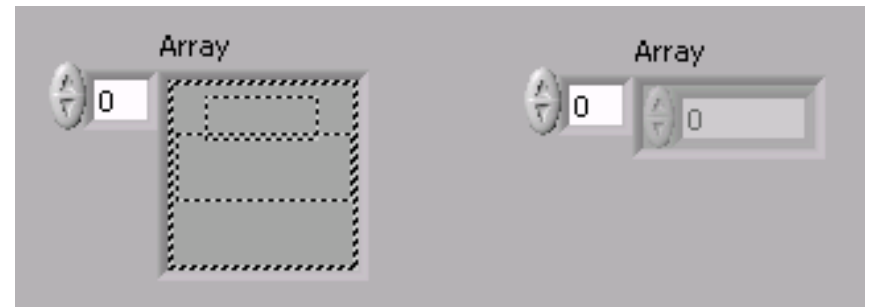
The element selected in the index display always refers to the element shown in the upper-left corner of the element display.

# Array Controls and Indicators

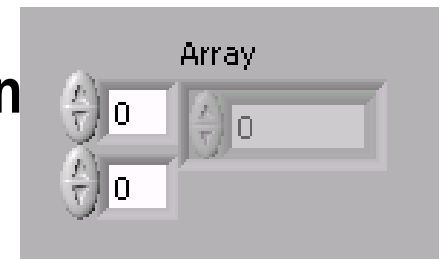
1. Select the **Array** shell from the Controls palette



2. Place data object inside shell

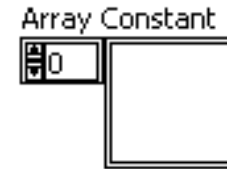
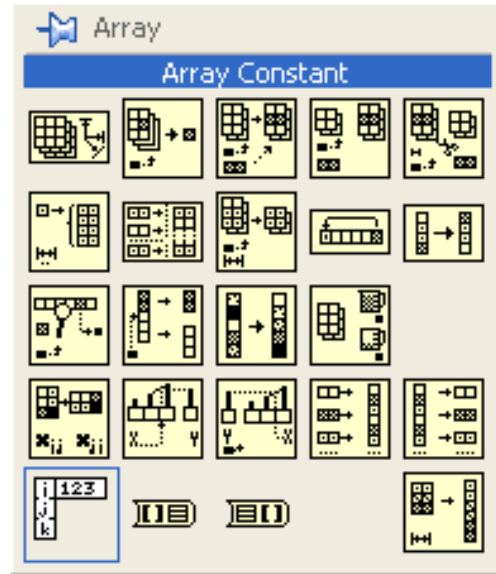


**Add Dimension**  
for 2D arrays

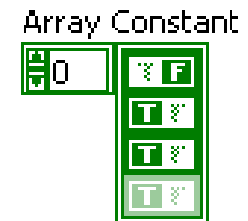


# Creating Array Constants

1. Select **Array Constant** shell from the **Array** subpalette

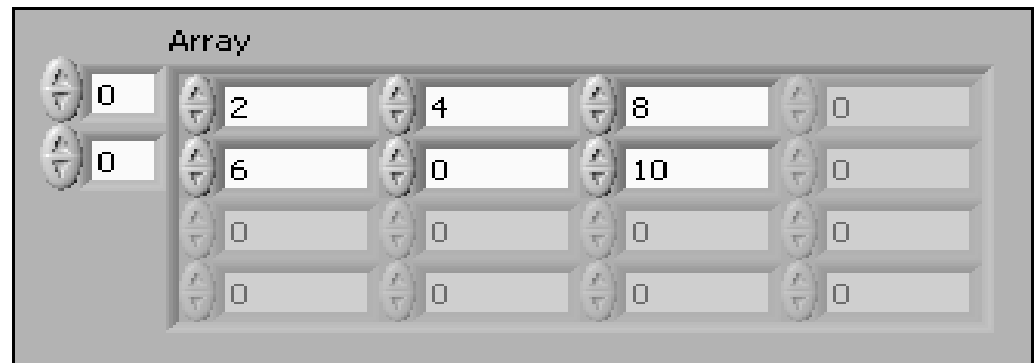
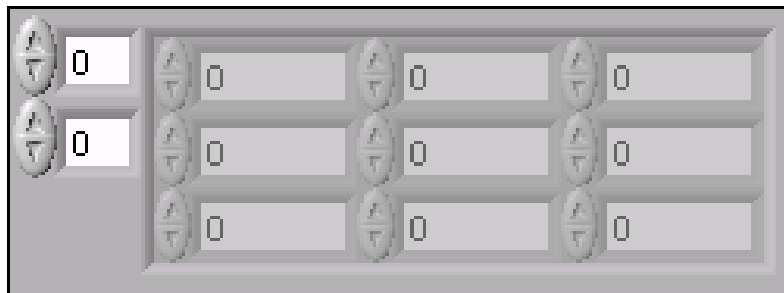


2. Place the data object in the array shell



# Initializing Arrays

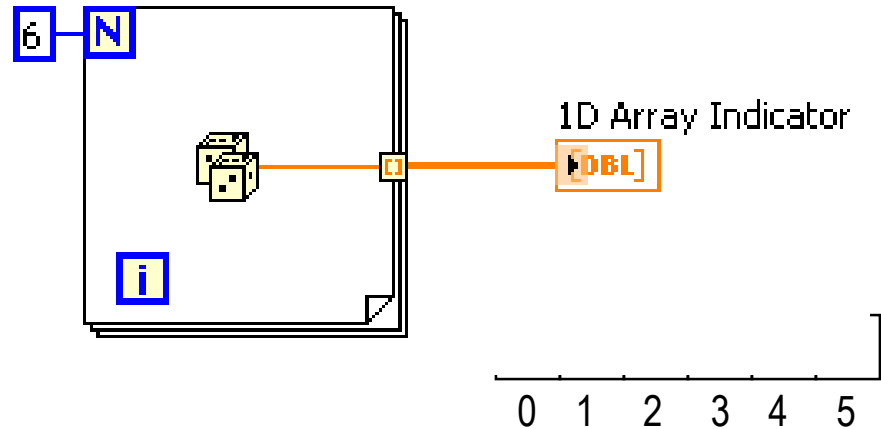
- You can initialize an array or leave it uninitialized.
- For initialized arrays, you define the number of elements in each dimension and the contents of each element.
- Uninitialized arrays have dimension but no elements.



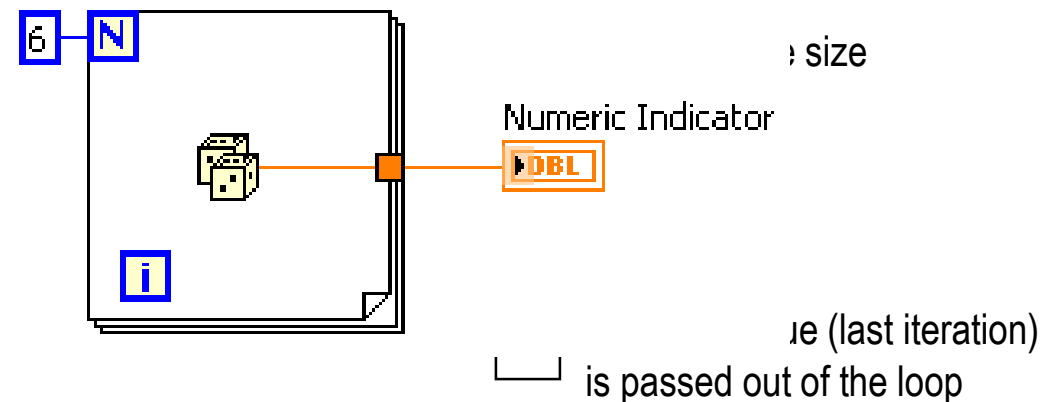
# Auto-Indexing

- Loops can accumulate arrays at their boundaries with auto-indexing
- For Loops auto-index by default
- While Loops output the final value by default
- Right-click on tunnel and enable/disable auto-indexing

## Auto-Indexing Enabled



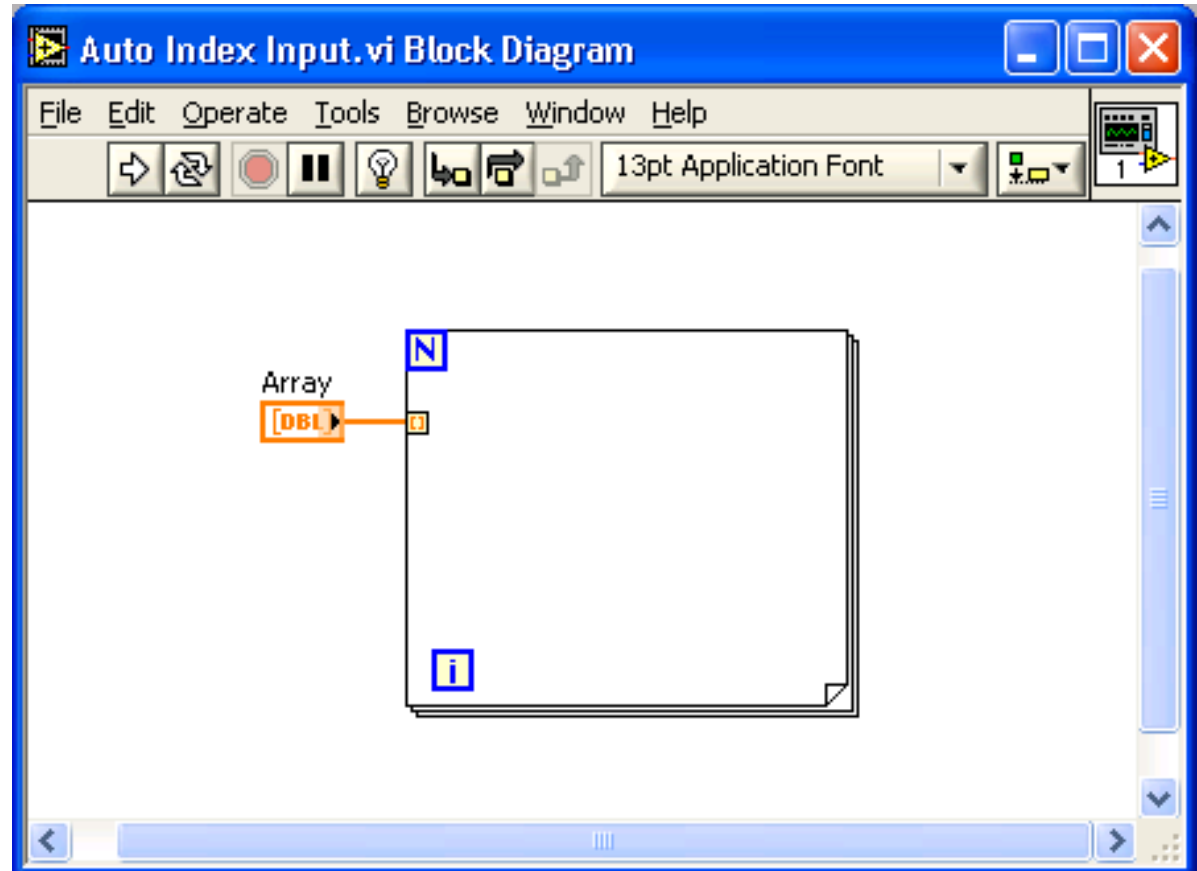
## Auto-Indexing Disabled



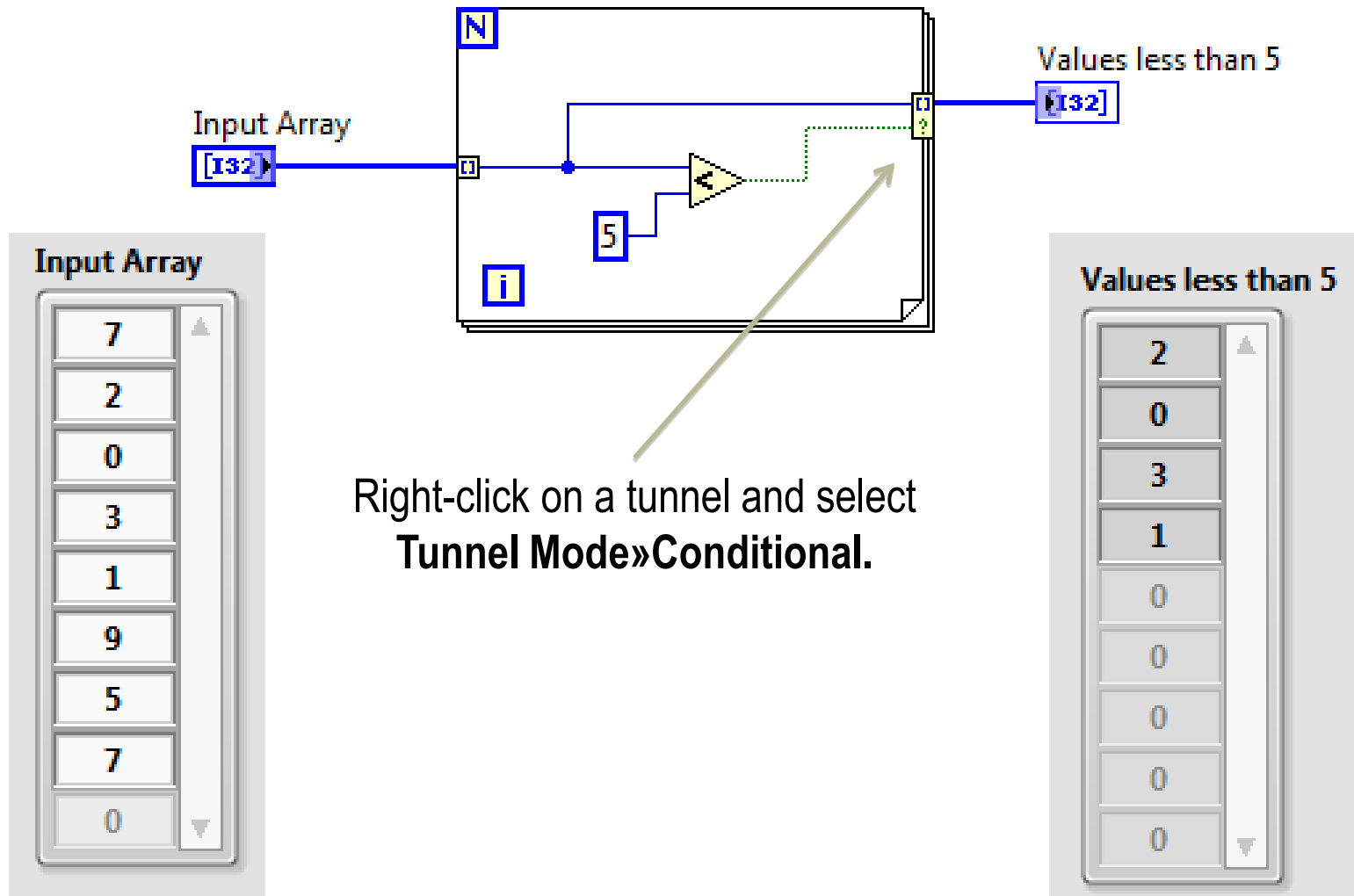


# Auto-Index Input

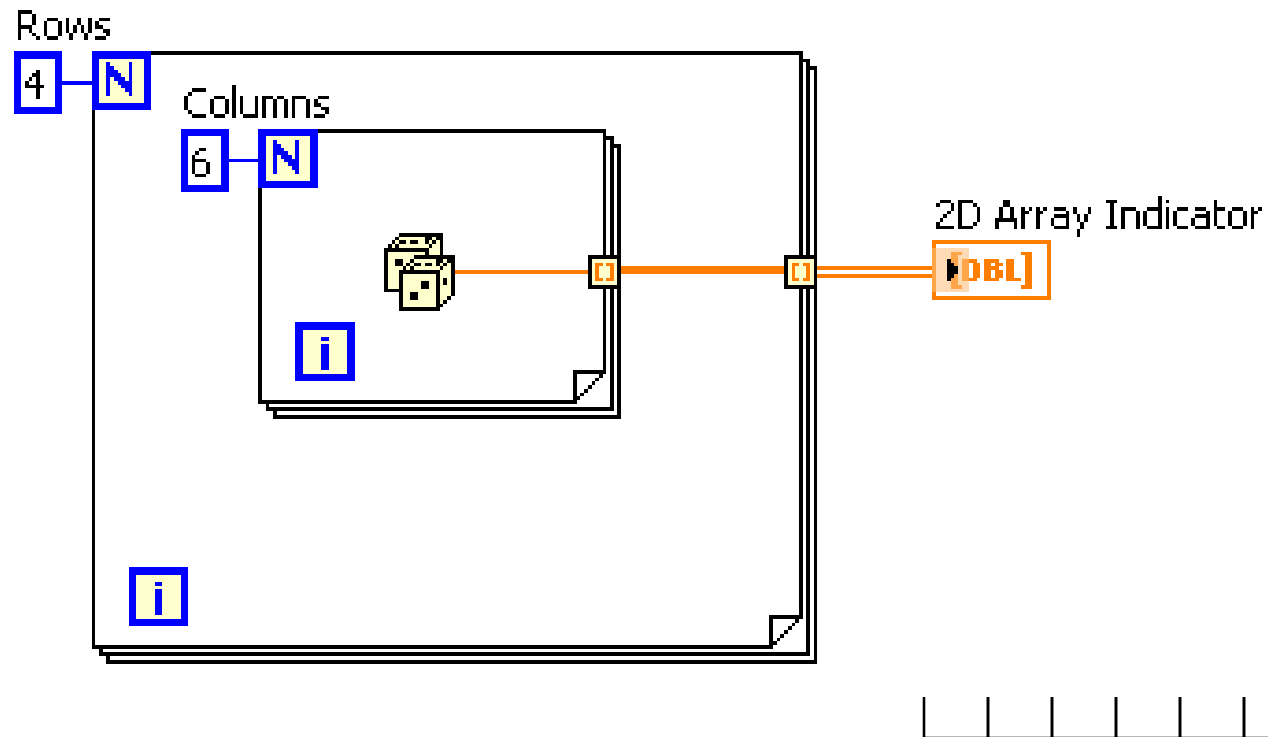
- An array input can be used to set the For Loop count terminal
- Number of elements in the array equals the count terminal input
- Run arrow not broken



# Auto-Indexing with a Conditional Tunnel

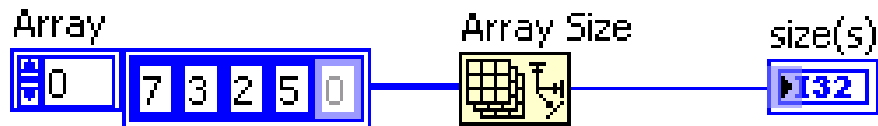


# Creating 2D Arrays

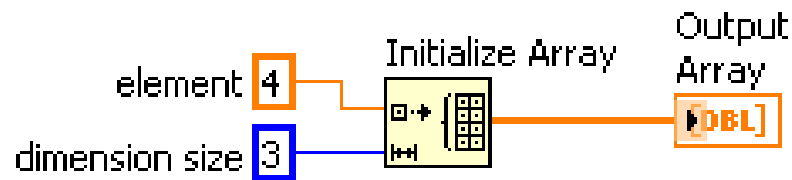


- Inner loop creates column elements
- Outer loop stacks them into rows

# Common Array Functions

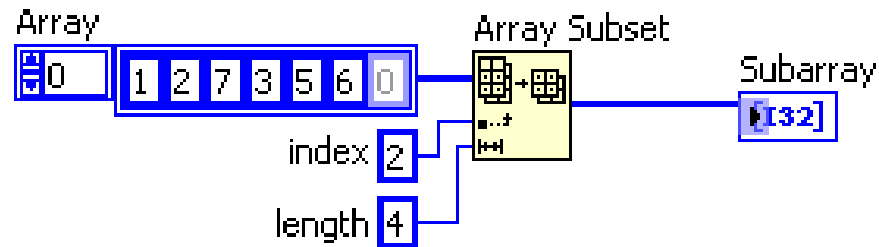


Array Size



Initialize Array

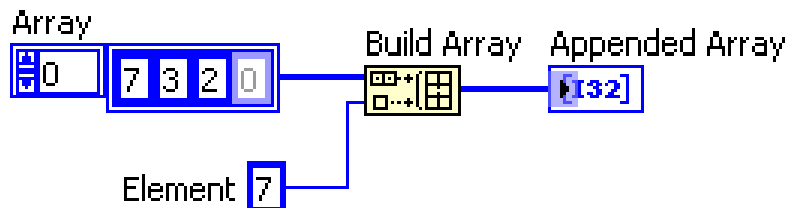
# Common Array Functions



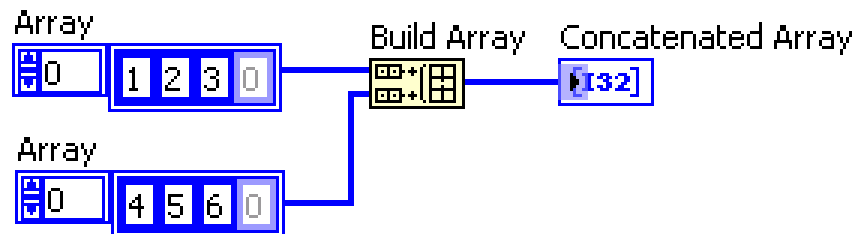
## Array Subset



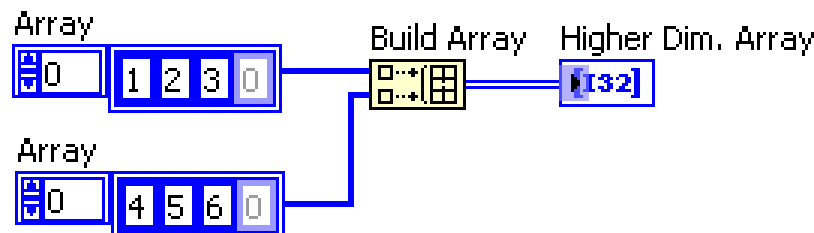
# The Build Array Function



## Appending an element



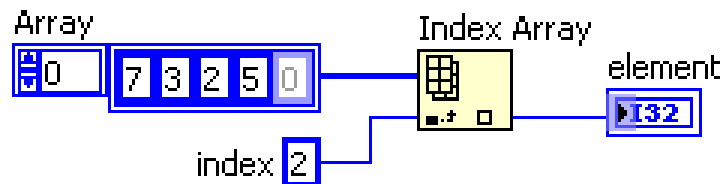
## Concatenate Inputs



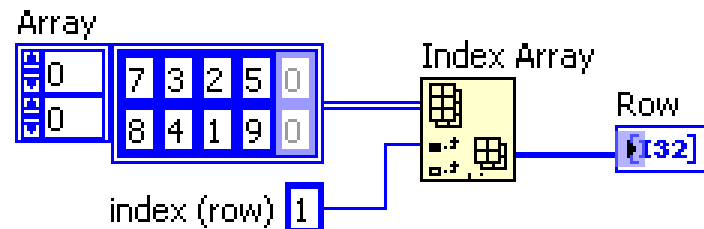
*default*

## Building a higher dimension array

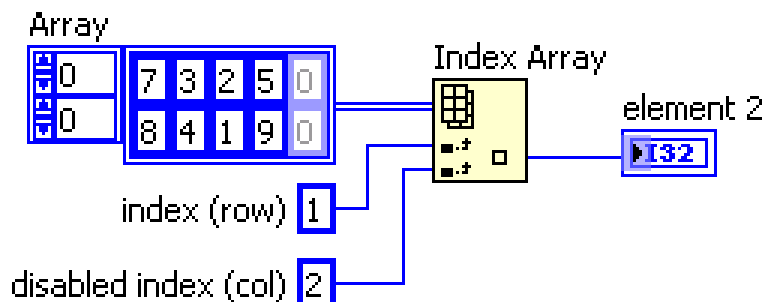
# The Index Array Function



## Extracting an Element

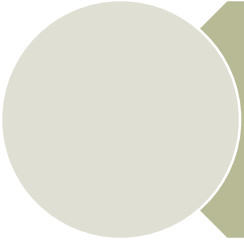


## Extracting a Row



## Extracting an Element of a Row

# Polymorphism



**Polymorphism** - The ability of VIs and functions to automatically adapt to accept input data of different data types

Functions are polymorphic to varying degrees:

- None, some, or all of their inputs can be polymorphic.
- Some accept numeric or Boolean values.
- Some accept numeric or strings.
- Some accept scalars, numeric arrays, or clusters of numerics.



# Polymorphism

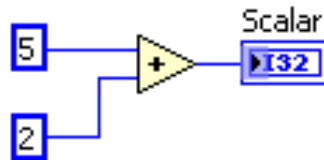
Function inputs can be of different types

All LabVIEW arithmetic functions are polymorphic

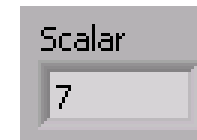
## Combination

## Result

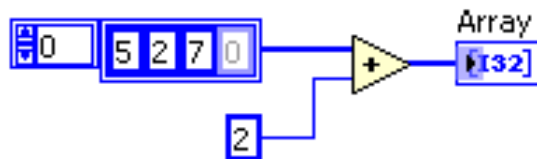
Scalar + Scalar



Scalar



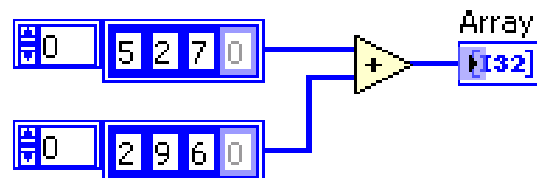
Array + Scalar



Array



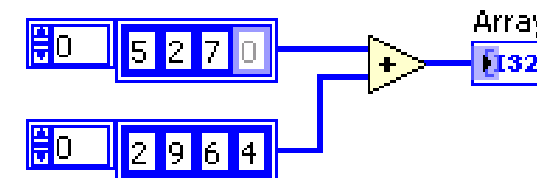
Array + Array



Array



Array + Array



Array

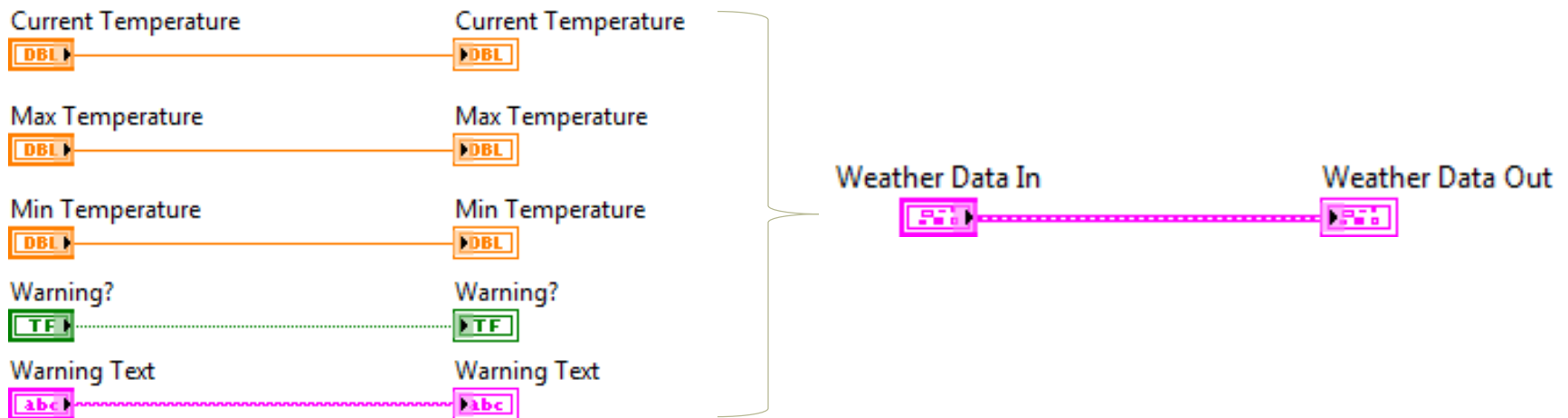


# Summary

- Arrays group data elements of the same type. You can build arrays of numeric, Boolean, path, string, waveform, and cluster data types.
- The array 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.
- To create an array control or indicator, select an Array on the **Controls»Array & Cluster** palette, place it on the front panel, and drag a control or indicator into the array shell.
- If you wire an array to a For Loop or While Loop input tunnel, you can read and process every element in that array by enabling auto-indexing.
- By default, LabVIEW enables auto-indexing in For Loops and disables auto-indexing in While Loops.
- Polymorphism is the ability of a function to adjust to input data of different data structures.

# Why Use Clusters?

- Keep data organized.
  - Logically group related data values together.
  - Improve diagram readability by eliminating wire clutter.
- Reduce the number of connector pane terminals.

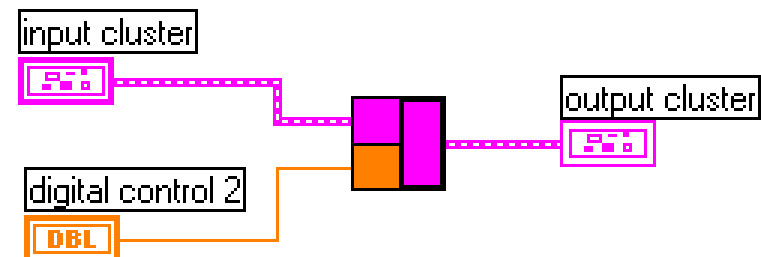
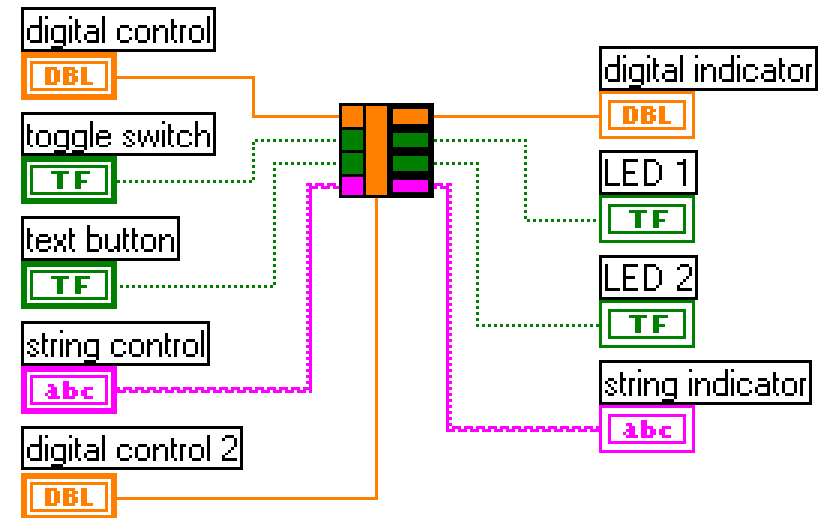


# Using Clusters to Pass Data to SubVIs

Use clusters to pass several values to one terminal

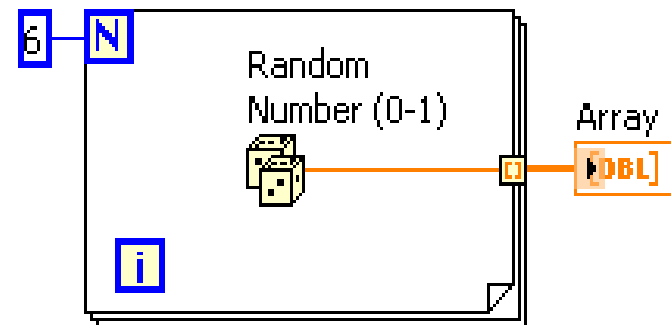
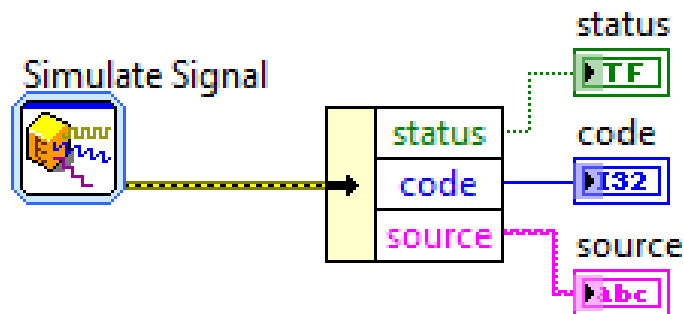
Overcomes 28-terminal limit

Simplifies wiring



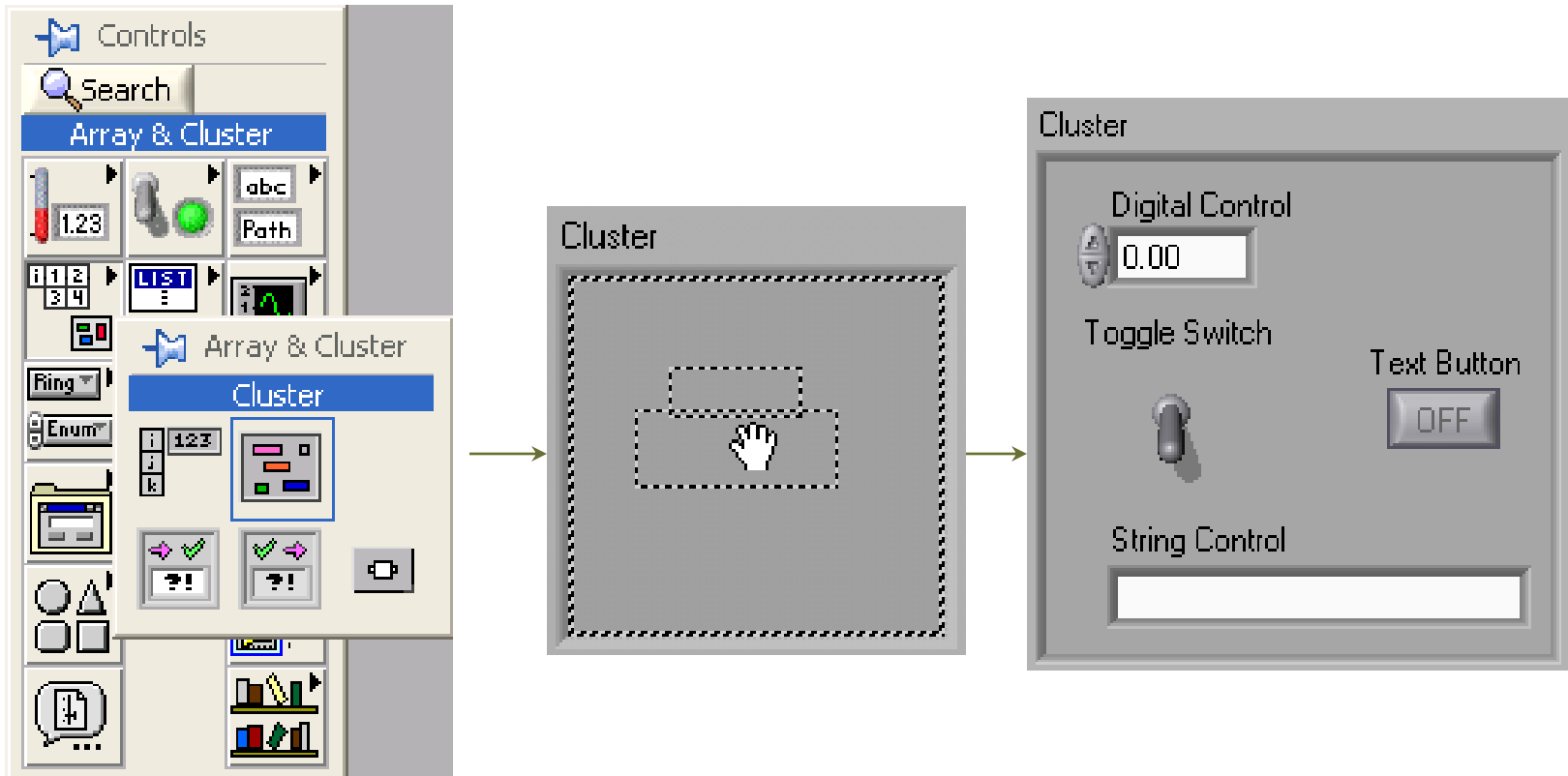
# Clusters vs. Arrays

- Clusters are a fixed size.
  - Clusters can contain mixed data types.
  - Clusters can be a control, an indicator, or a constant.
    - All elements have to be controls, indicators, or
- Arrays vary in size.
  - Arrays contain only one data type.
  - Arrays can be a control, an indicator, or a constant.



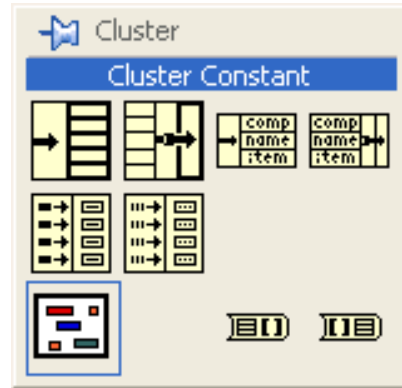
# Cluster Controls and Indicators

1. Select a **Cluster** shell from the **Array & Cluster** subpalette
2. Place objects inside the shell



# Creating Cluster Constants

1. Select **Cluster Constant** shell from the **Cluster** subpalette

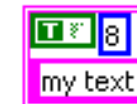


Cluster Shell



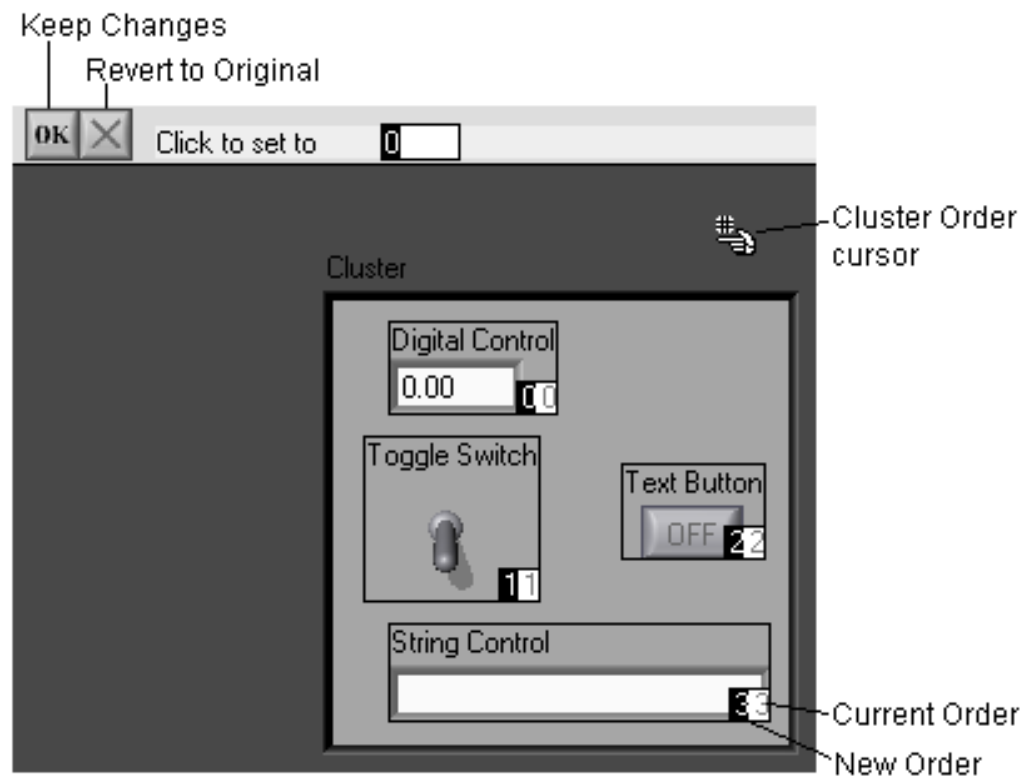
2. Place objects in the Cluster shell

Cluster Constant



# Cluster Order

- Elements have a logical order (start with 0)
- To change order, right-click the border and select **Reorder Controls in Cluster...**



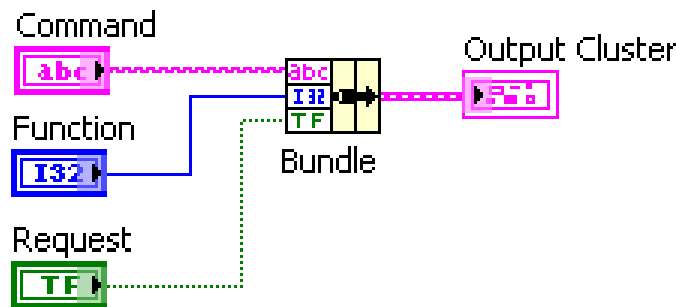


# Cluster Functions - Bundle

## Create new cluster

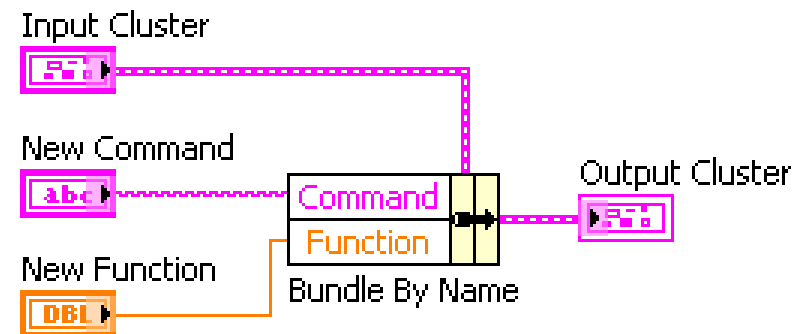
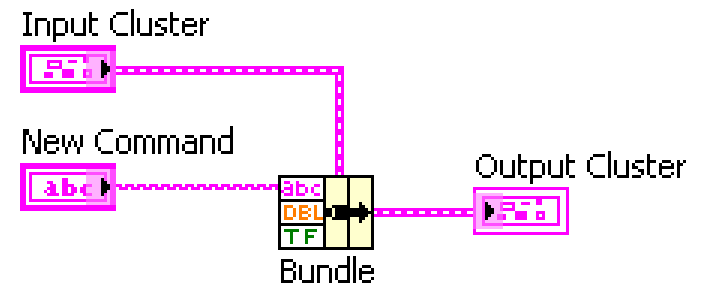
## Modify existing cluster

Bundle



Bundle  
By Name

Must have an existing cluster to use this function.



# Cluster Functions - Unbundle

Applicant Cluster

Name

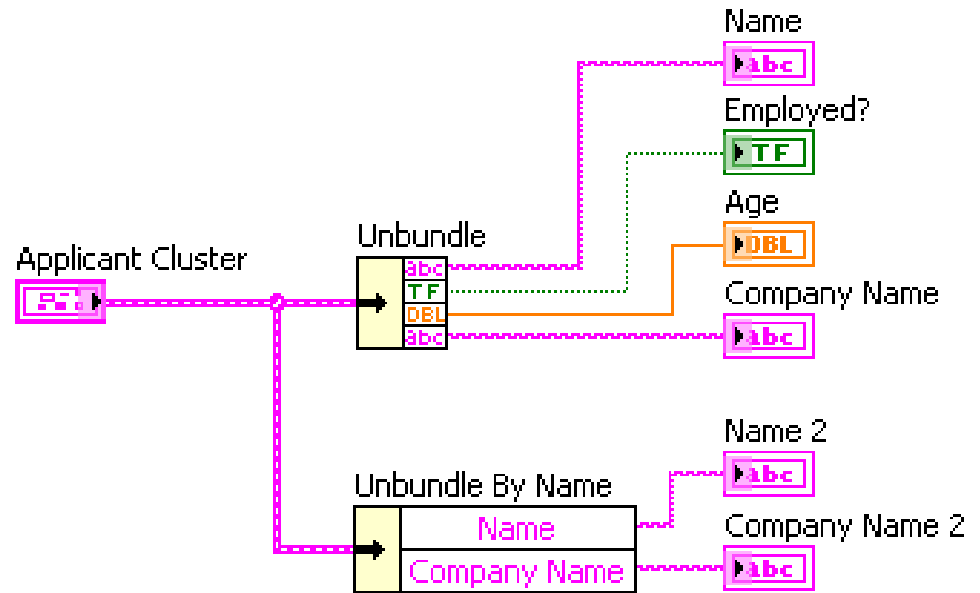
Age

Employed?

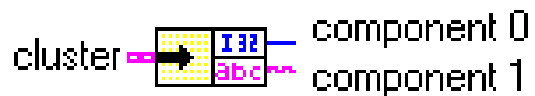
Yes ☐

No ☐

Company Name



## Unbundle



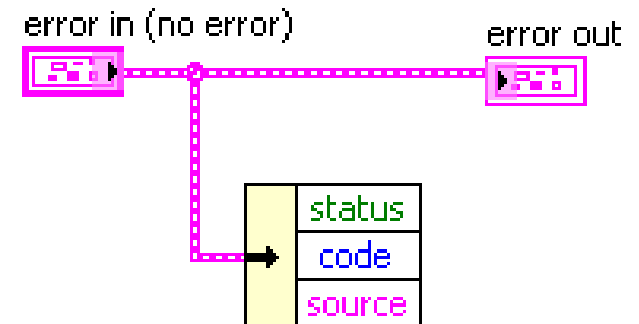
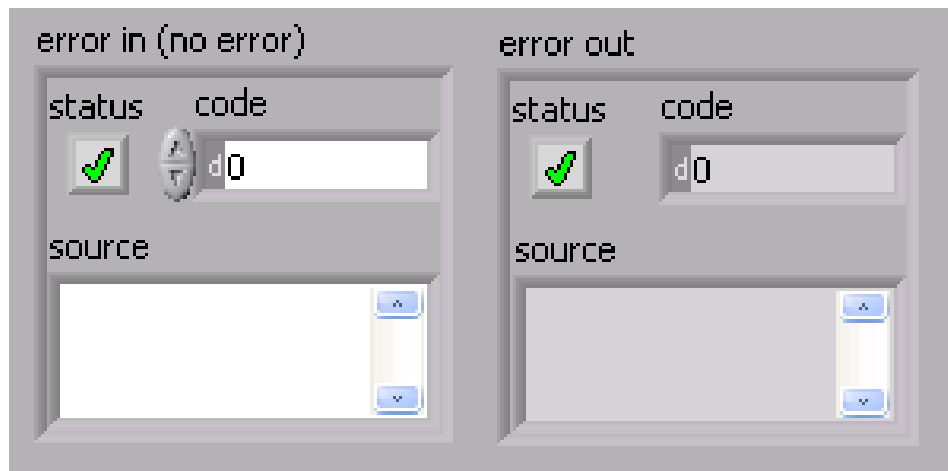
## Unbundle By Name



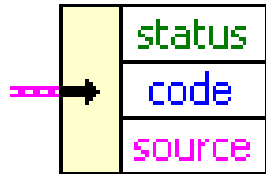
# Error Cluster

Use the error in and error out clusters in each VI you use or build to handle errors in the VI.

The error clusters located on the **Controls»Array & Cluster** palette include the components of information shown



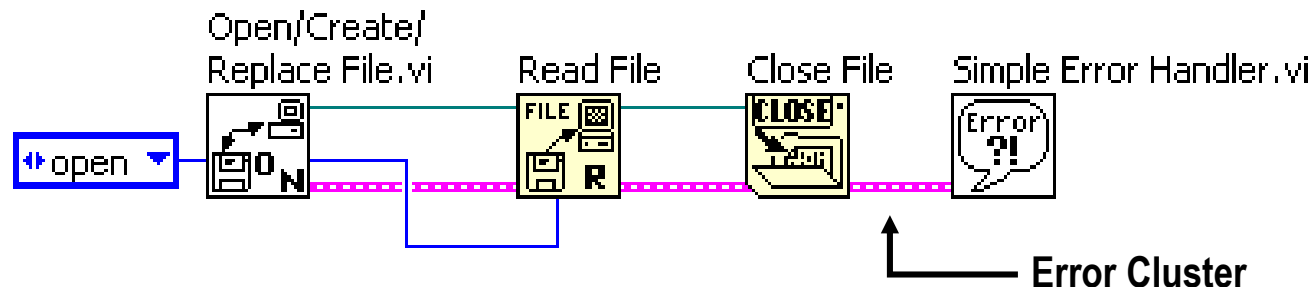
# Error Cluster Details



- **Status** is a Boolean value that reports TRUE if an error occurred. Most VIs, functions, and structures that accept Boolean data also recognize this parameter.
- **Code** is a signed 32-bit integer that identifies the error numerically. A non-zero error code coupled with a status of FALSE signals a warning rather than a fatal error.
- **Source** is a string that identifies where the error occurred.

# Error Handling with Clusters

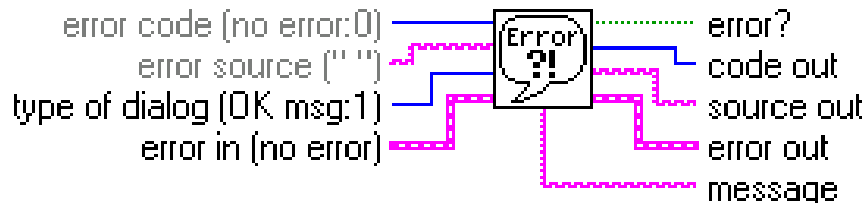
- LabVIEW does not handle errors automatically. In LabVIEW, you can make these error handling decisions on the block diagram of the VI.
- Error handling in LabVIEW follows the dataflow model. Just as data flow through a VI, so can error information.
- Wire the error information from the beginning of the VI to the end.



# Simple Error Handler

Use the Simple Error Handler to handle the error at the end of the execution flow.

The Simple Error Handler is located on the **Functions»All Functions»Time and Dialog** palette. Wire the error cluster to the Error In (no error) input.

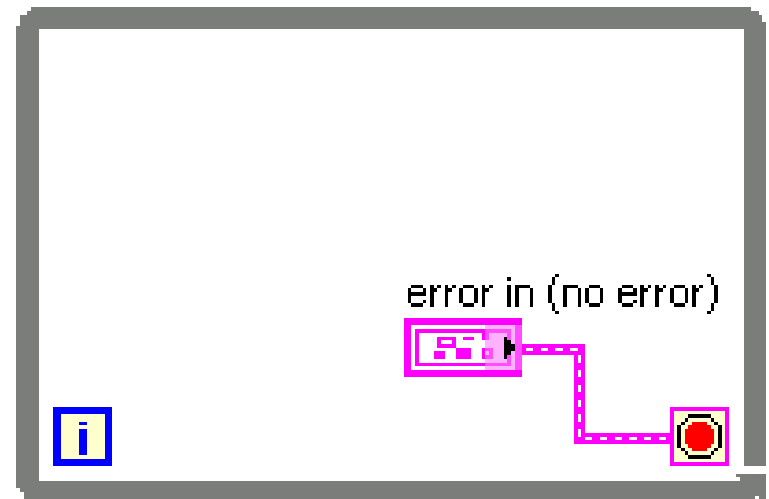


# Using While Loops for Error Handling

You can wire an error cluster to the conditional terminal of a While Loop to stop the iteration of the While Loop.

Only the TRUE or FALSE value of the status parameter of the error cluster is passed to the terminal.

When an error occurs, the While Loop stops.



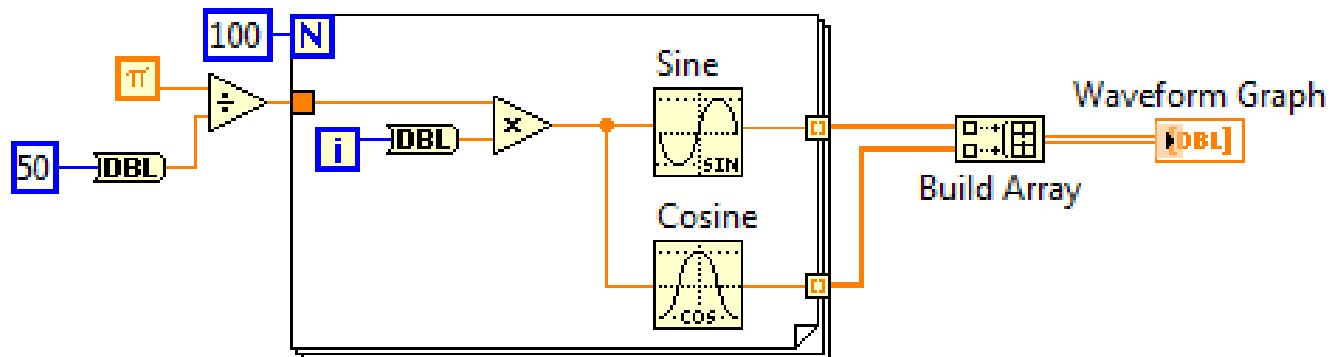
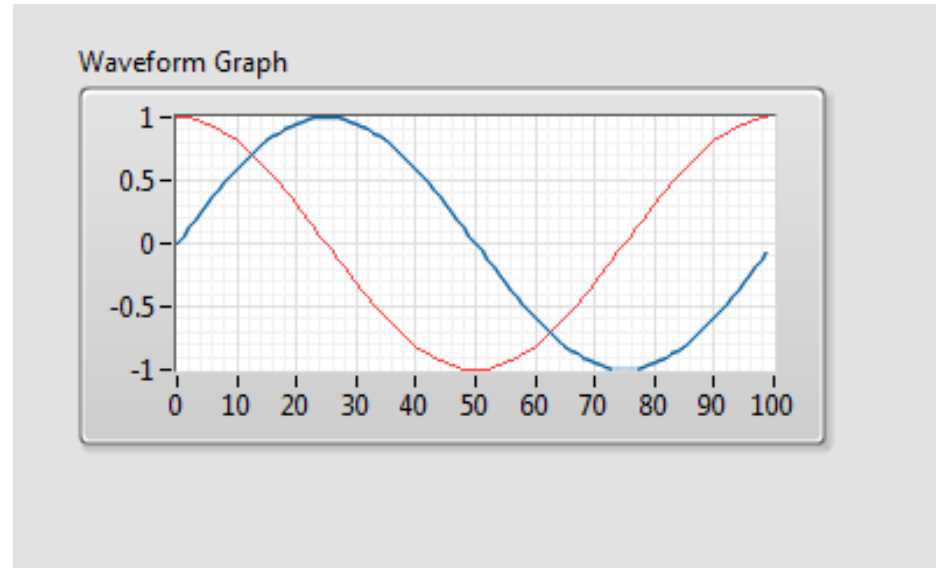
# Summary

- Clusters group data elements of mixed types. A cluster cannot contain a mixture of controls and indicators.
- To create a cluster control or indicator, select a cluster on the **Controls»Array & Cluster** palette, place it on the front panel, and drag controls or indicators into the cluster shell.
- Use the Cluster functions located on the **Functions»All Functions»Cluster** palette to create and manipulate clusters.
- Error checking tells you why and where errors occur.
- The error cluster reports the status, code and source of the error.
- Use the error cluster controls and indicators to create error inputs and outputs in subVIs.



# Waveform Graph

- Is a graphical display of data.
- Displays one or more plots of evenly sampled measurements.
- Is used to plot pre-generated arrays of data.
- Can display plots with any number of data points.

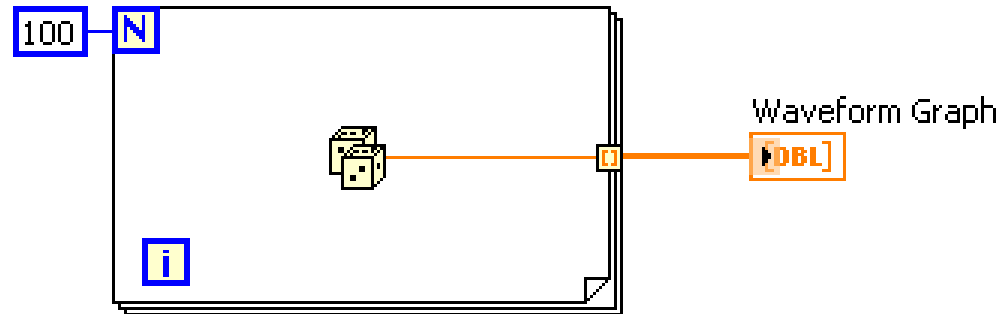


# Single-Plot Waveform Graphs

## Uniform X axis

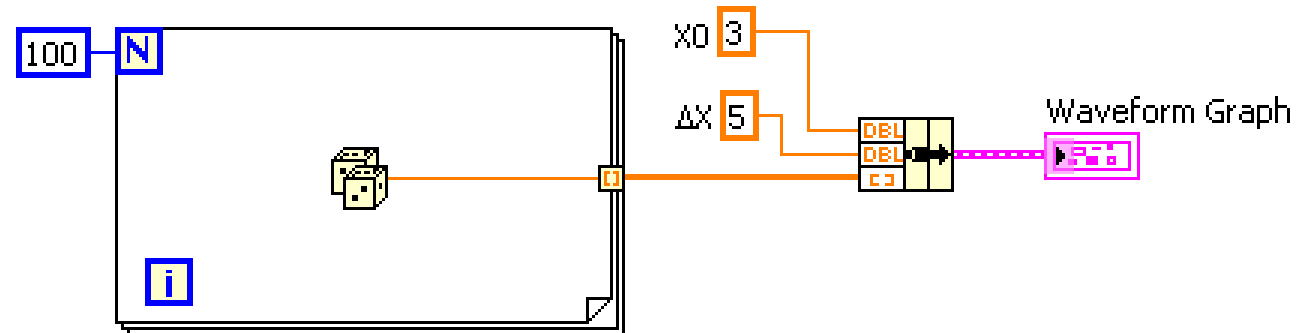
Initial X = 0.0

Delta X = 1.0



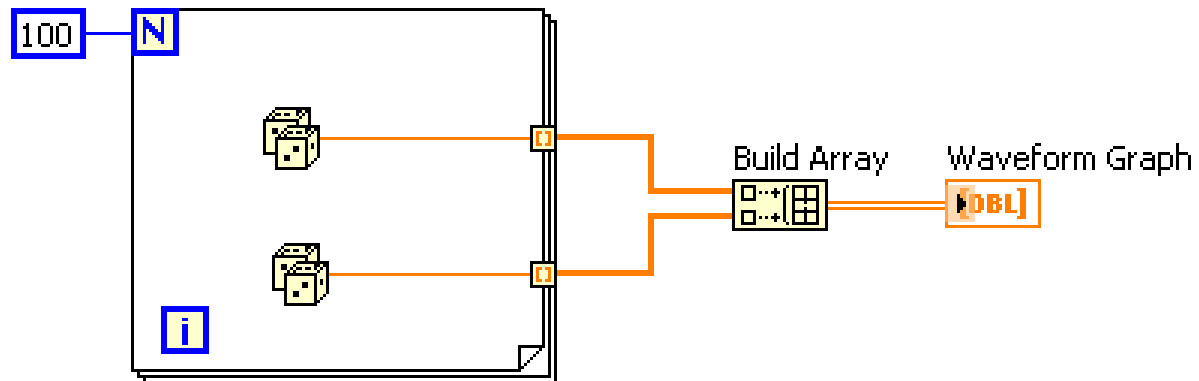
## Uniform X axis

you specify point  
spacing

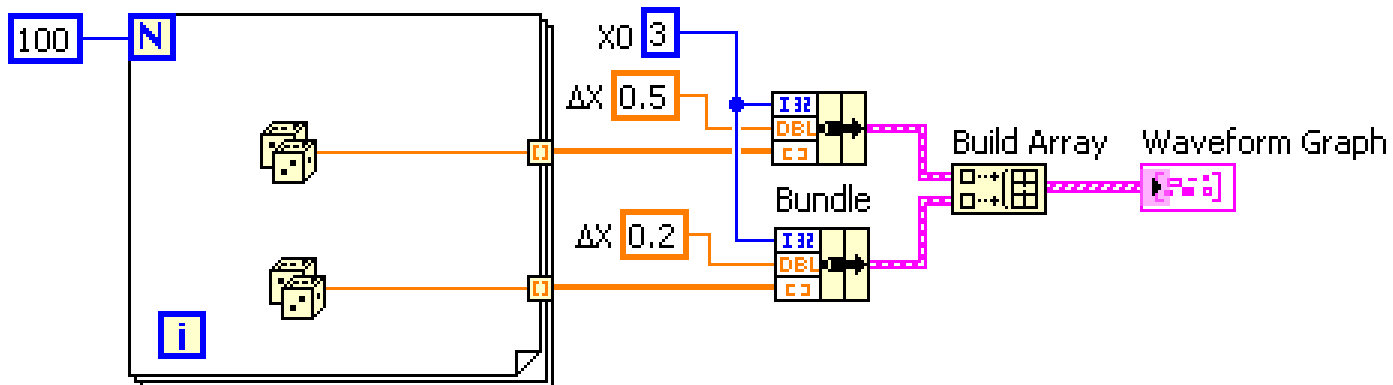


# Multiple-Plot Waveform Graphs

Each row is a  
separate plot:  
Initial X = 0  
Delta X = 1

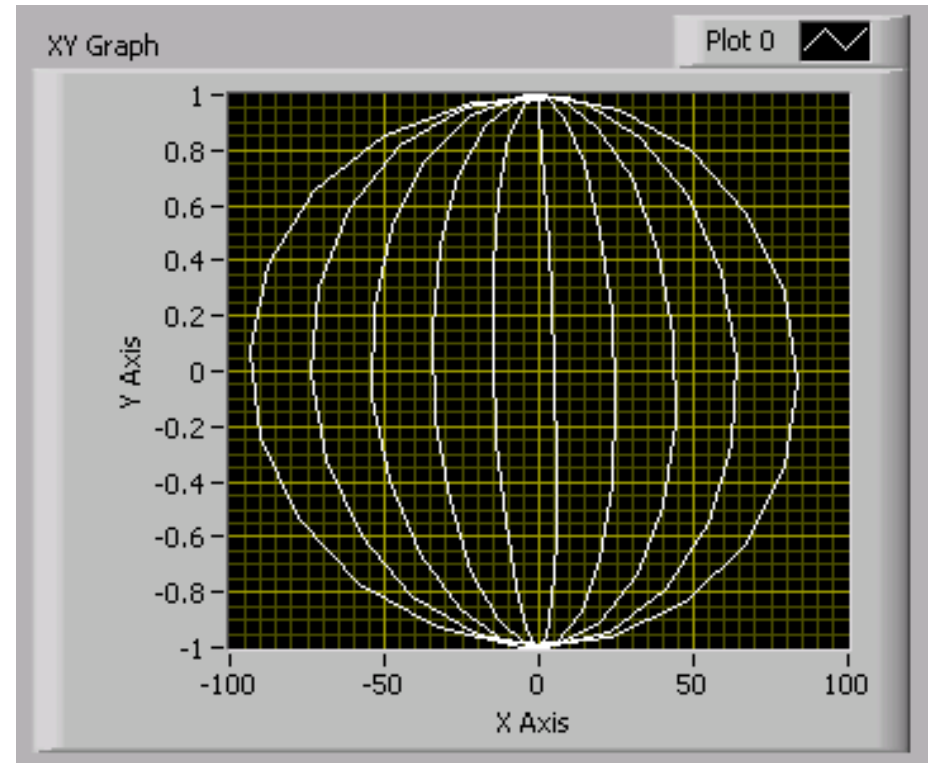
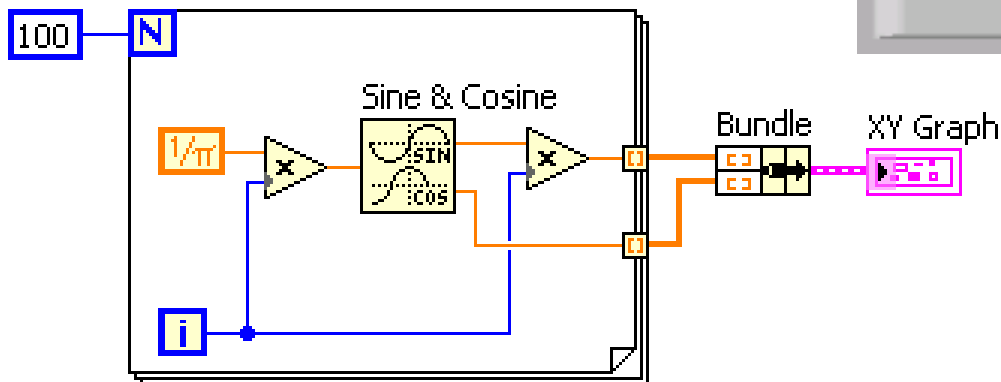


Each row is a  
separate plot:  
Bundle specifies  
point spacing of  
the X axis



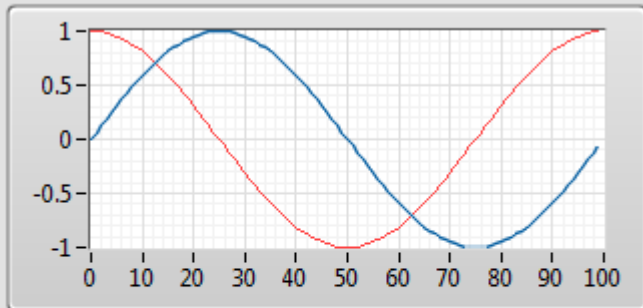
# XY Graphs

- Non-uniform X axis
- Separate X and Y arrays define data points

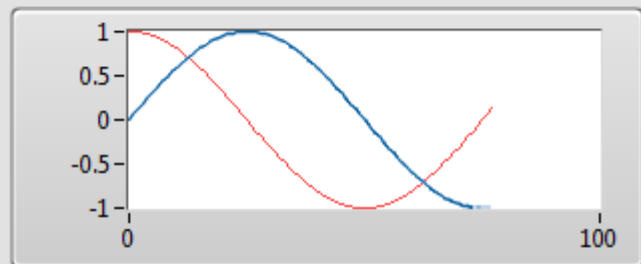


# Charts vs. Graphs – Multi-plot and XY Graph

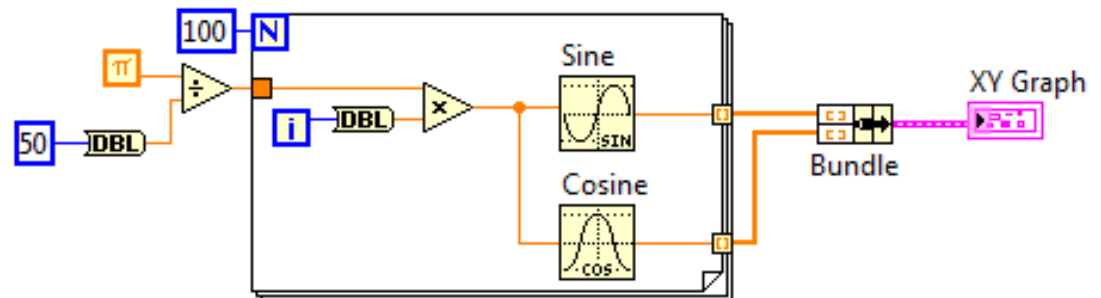
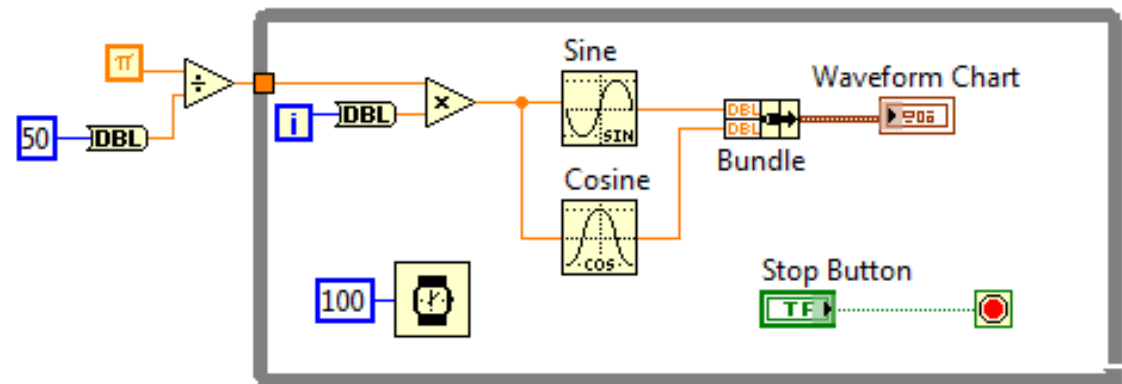
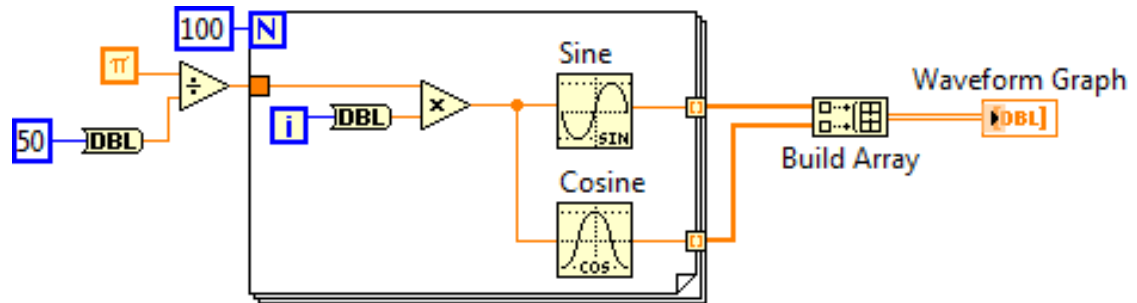
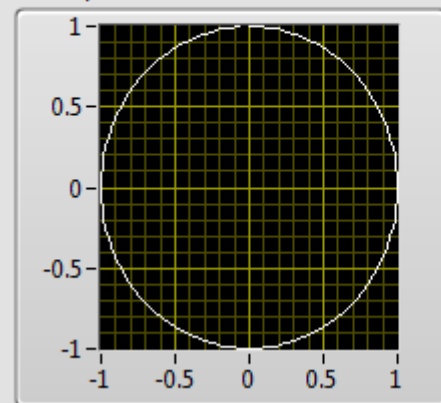
Waveform Graph



Waveform Chart





XY Graph




# Chart and Graph Use Summary

Use the Context Help window with charts and graphs

  
↓

  
↓

  
↓

**Context Help**

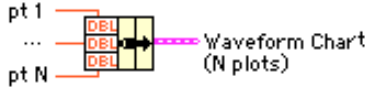
**Waveform Charts:**

Wire data directly to chart:

Data	Resulting Chart
Scalar	Single plot - 1pt
1D	Single Plot - 1 or more pts
WDT	Single Plot - 1 or more pts
2D	Multiplot - 1 or more pts

WDT (Waveform Data Type) includes timing info.

Or combine points with a bundle node:



Or use timing information in WDT.

**See the example: Charts.vi**

**Context Help**

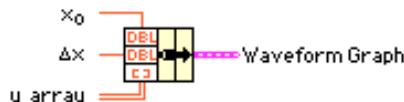
**Waveform Graphs:**

Wire data directly to waveform graph:

Y Array	Resulting Graph
1D	Single Plot
WDT	Single Plot
2D	Multiplot

WDT (Waveform Data Type) includes timing info. Others default to 0 for  $x_0$  and 1 for  $\Delta x$ .

Combine timing information using a bundle node:

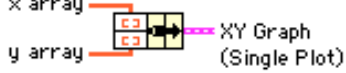


**See the example: Waveform Graph.vi**

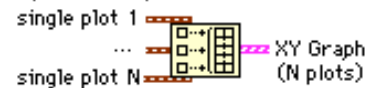
**Context Help**

**XY Graphs:**

Single Plot XY Graph:

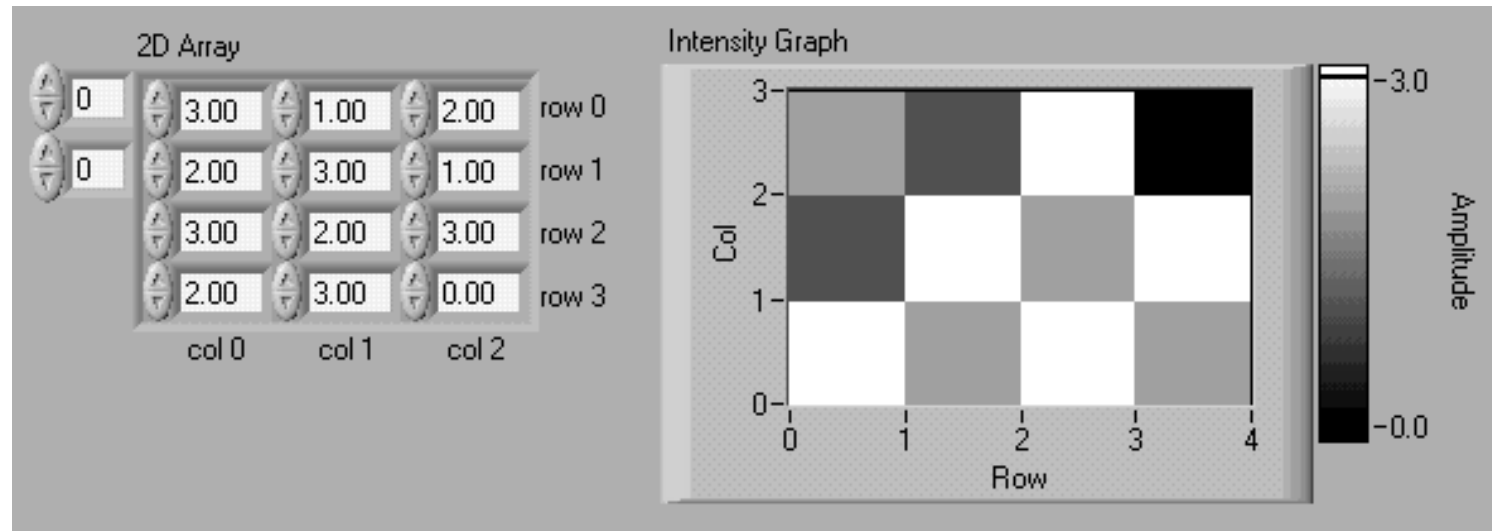


Multiplot XY Graph:



**See the example: XY Graph.vi**

# Intensity Plots and Graphs



- Useful in displaying terrain, temperature patterns, spectrum analysis, and image processing
- Data type is a 2D array of numbers; each number represents a color
- Use these options to set and display color mapping scheme
- Cursor also adds a third dimension

# Summary

- Waveform graphs and XY graphs display data from arrays.
- Right-click a waveform chart or graph or its components to set attributes of the chart and its plots.
- You can display more than one plot on a graph using the Build Array function and the Bundle function for charts and XY graphs. The graph becomes a multiplot graph when you wire the array of outputs to the terminal.
- When you wire data to charts and graphs, use the Context Help window to determine how to wire them.
- You can use intensity charts and graphs to plot three-dimensional data. The third dimension is represented by different colors corresponding to a color mapping that you define. Intensity charts and graphs are commonly used in conjunction with spectrum analysis, temperature display, and image processing.