A. Arrays

An array is a collection of data elements that are all the same type. An array has one or more dimensions and up to 2^{31} elements per dimension, memory permitting. Arrays in LabVIEW can be of any type. You cannot, however, have an array of arrays, charts, or graphs. You access each array element by its index. The index is in the range 0 to N-1, where N is the number of elements in the array. The *one-dimensional* (1D) array shown below illustrates this structure. Notice that the *first* element has index 0, the *second* element has index 1, and so on.

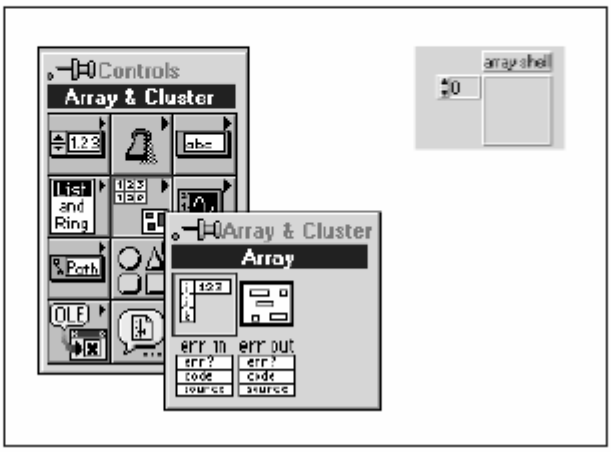
index	0	1	2	3	4	5	6	7	8	9
10-element array	1.2	3.2	8.2	8.0	4.8	5.1	6.0	1.0	2.5	1.7

Creating Array Controls and Indicators

You create the array control or indicator by combining an *array shell* with a *data object*, which can be numeric, Boolean, or string.

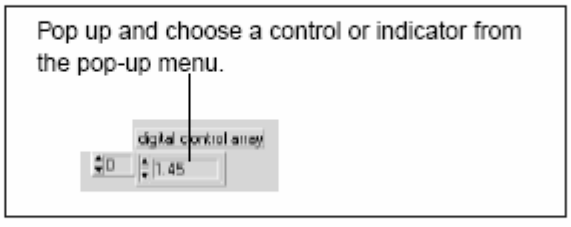
Step 1:

Select an empty array shell from the **Array & Cluster** subpalette of the **Controls** palette.



Step 2:

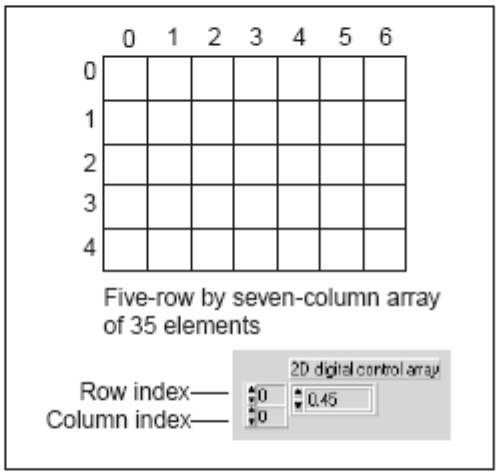
To create an array, you drag a *data object* into the array shell.



☞ **Note** ***Remember that you must assign a data object to the empty array shell before using the array on the block diagram. If you do not assign a data object, the array terminal will appear black with an empty bracket.***

Two-Dimensional Arrays

A two-dimensional (2D) array requires two indices—a row index and a column index, both of which are zero based—to locate an element. The example to the right is an N-row by M-column array, where N=5 and M=7.



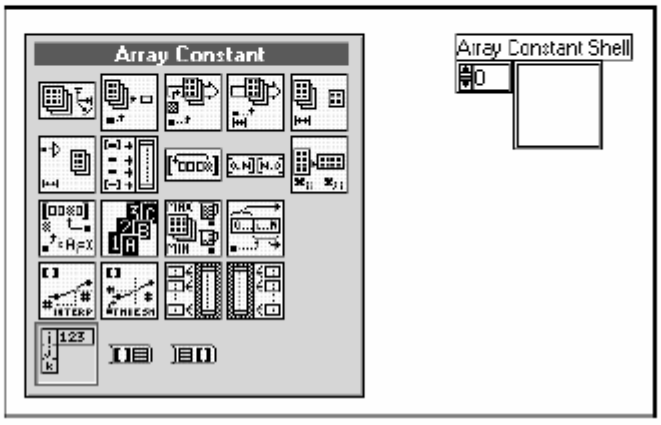
You add dimensions to the array control or indicator by popping up on the array *index display* and choosing **Add Dimension** from the pop-up menu. The example to the right shows a 2D digital control array.

Creating Array Constants

You can create array constants in the block diagram by combining an array shell with a data object as you would on the front panel. Array constants are a combination of an **Array Constant** shell found in the **Array** subpalette of the **Functions** palette and a data constant. The following example demonstrates how to create a Boolean array constant.

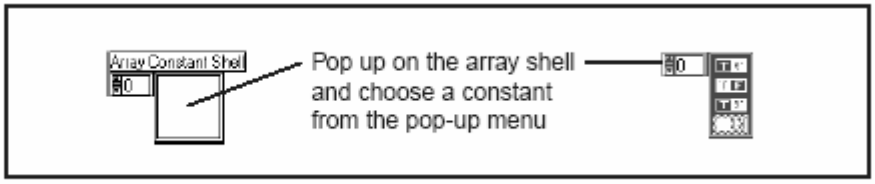
Step 1:

Select an empty **Array Constant** shell from the **Array** subpalette of the **Functions** palette.



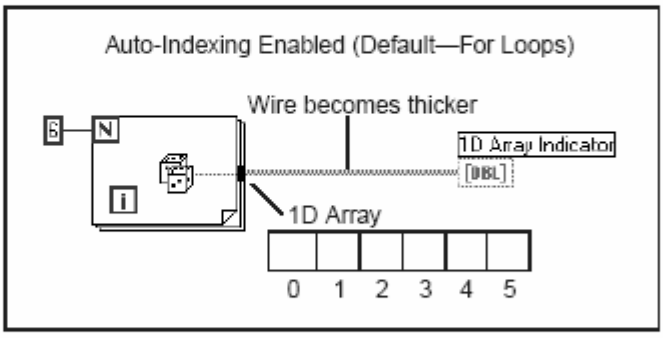
Step 2:

To create an array, you drag a *data object* into the array shell. Different data objects include numeric, Boolean, or string constants from the **Functions** palette.

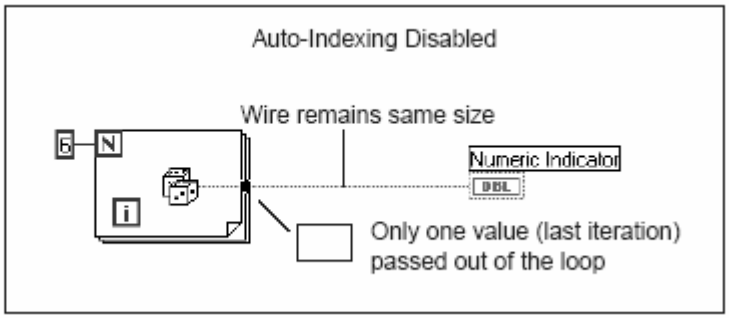


B. Creating Arrays with Loops

The For Loop and While Loop can index and accumulate arrays at their boundaries automatically. This capability is called *auto-indexing*. The illustration below shows a For Loop auto-indexing an array at its boundary. Each iteration creates the next array element. After the loop completes, the array passes to the indicator. Notice that the wire becomes thicker as it changes to an array at the loop border.

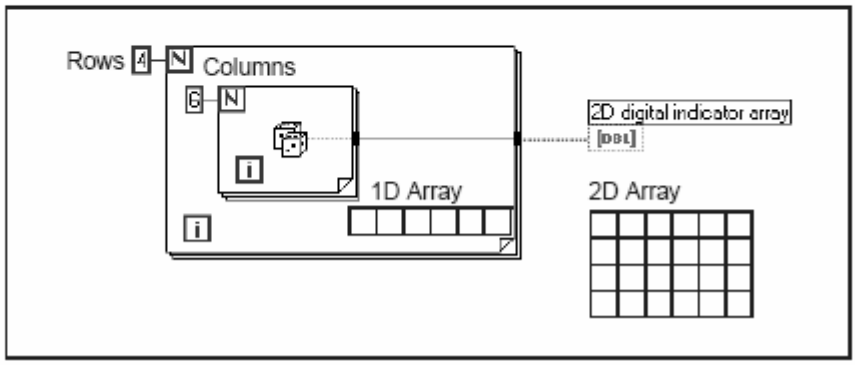


If you need the last array value passed to the tunnel out of a loop without creating an array, you must disable auto-indexing by popping up on the tunnel (the black square on the border) and choosing **Disable Indexing** from the pop-up menu. In the illustration below, auto-indexing is disabled, and only the last value returned from the **Random Number (0-1)** function passes out of the loop. Notice that the wire remains the same size after it leaves the loop.



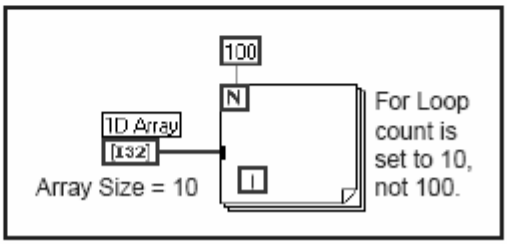
Creating Two-Dimensional Arrays

You can use two For Loops, 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. The example below shows two For Loops auto-indexing a 2D array containing random numbers.



Using Auto-Indexing to Set the For Loop Count

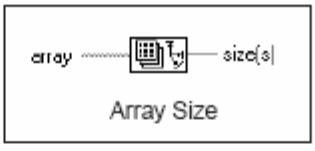
When you enable auto-indexing on an array *entering* a For Loop, LabVIEW automatically sets the loop iteration count to the array size, thus eliminating the need to wire a value to the count terminal, **N**. If you enable auto-indexing for more than one array, or if you set the count, the count becomes the smaller of the two choices. In the example below, the array size, and not **N**, sets the For Loop count because the array size is the smaller of the two.



C. Array Functions

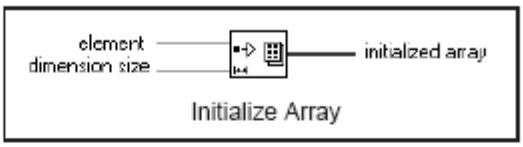
LabVIEW has many functions to manipulate arrays in the **Array** subpalette of the **Functions** palette. Some common functions are discussed below.

Array Size returns the number of elements in the input array.

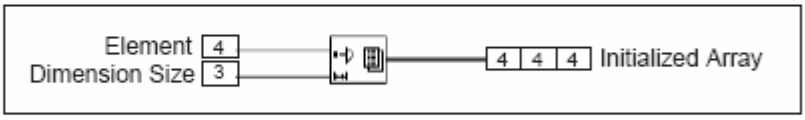


If the input array is N-dimensional, the output **size** is an array of N elements. Each element records the number of elements in each dimension.

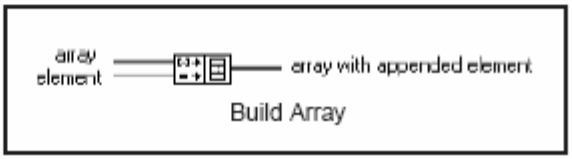
Initialize Array creates an **array** of **dimension size** elements containing the **element** value.



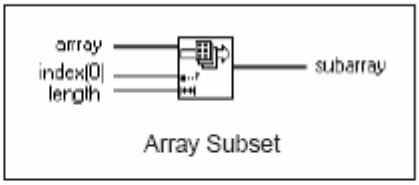
The function can be resized to correspond to the number of dimensions of the output array. The example below depicts a 1D array of three elements initialized with the value of 4.



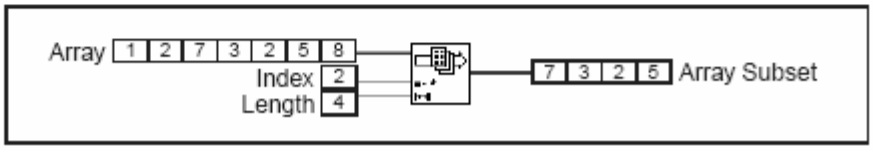
Build Array concatenates multiple arrays or appends elements to an array.



Array Subset returns a portion of an array starting at **index** and containing **length** elements.

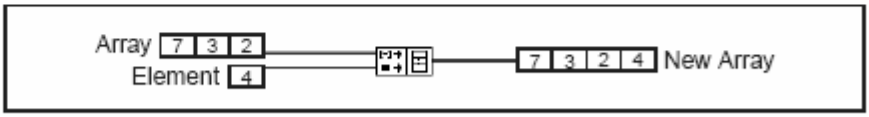


An example is shown below.

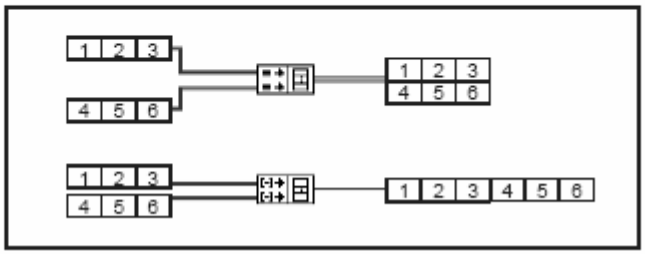


Build Array function when placed in the Diagram window

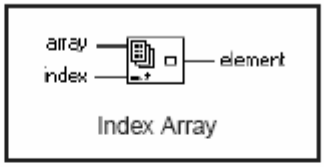
The function looks as shown at left when placed in the Diagram window. You can resize this function to increase the number of inputs. You can change the input type by popping up on the input and selecting *Change to Array* or *Change to Element*. For example, the **Build Array** function shown below is configured to concatenate an array and one element into a new array.



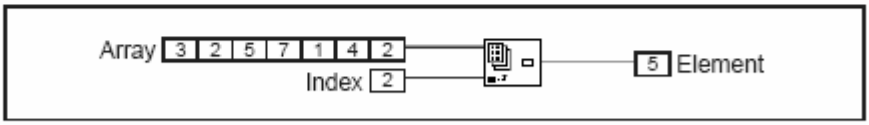
Note ***Keep in mind that if two 1D arrays are wired as element inputs, the output will be a 2D array where the top array will be the top row and the bottom input will be the bottom row. If the two 1D arrays are wired as array inputs, the output will append the two arrays to form a 1D array. See the examples below.***



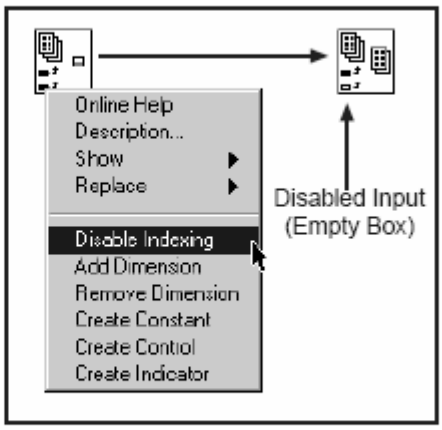
Index Array accesses an element of an array.



An example of an **Index Array** function accessing the third element of an array is shown below. Notice that the third element's index is two because the index starts at zero; that is, the first element has index zero.

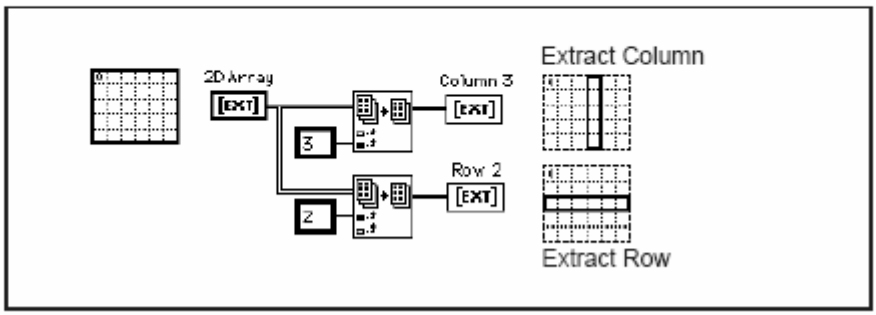


The previous example showed the **Index Array** function being used to extract a scalar element from an array. You also can use this function to *slice off* a row or column of a 2D array to create a subarray of the original. To do this, stretch the **Index Array** function to include two index inputs and select the **Disable Indexing** command on the pop-up menu of the index terminal as shown below.



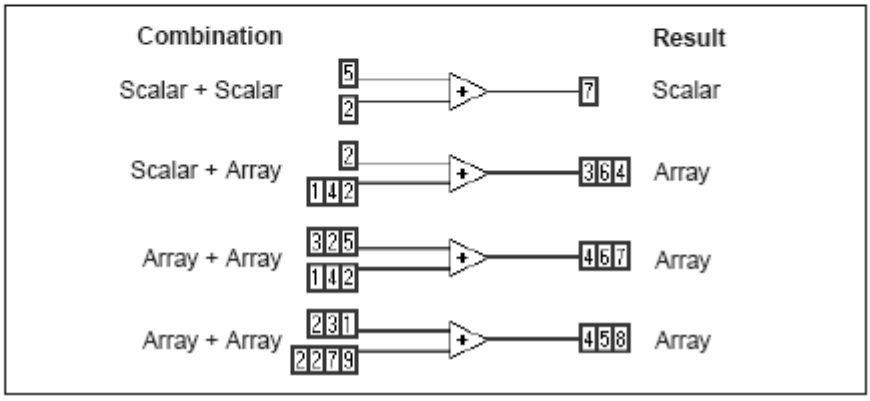
Notice that the index terminal symbol changes from a solid to an empty box when you disable indexing. You can restore a disabled index with the **Enable Indexing** command from the same menu.

You can extract subarrays using any combination of dimensions. The example below shows how to extract 1D row or column arrays from a 2D array.



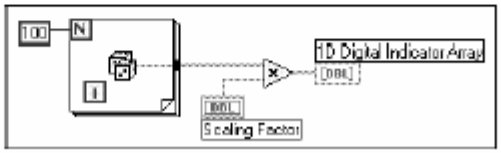
D. Polymorphism

The LabVIEW arithmetic functions, **Add**, **Multiply**, **Divide**, and so on, are *polymorphic*. This means that the inputs to these functions can be different data structures—scalars and arrays. For example, you can add a scalar to an array or add two arrays together. The example below shows some of the polymorphic combinations of the **Add** function.



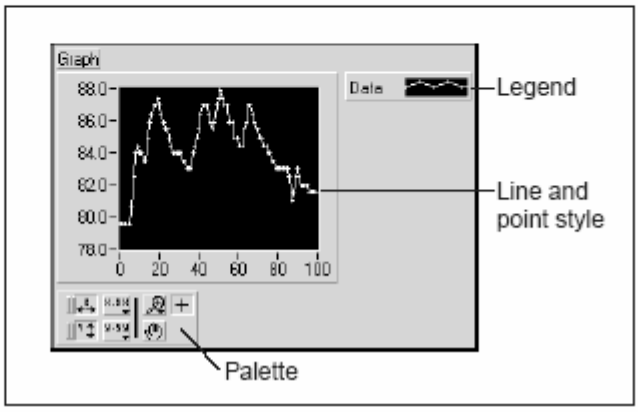
In the first combination, the result is a scalar. In the second combination, the scalar is added to *each* element of the array. In the third combination, each element of one array is added to the corresponding element of the other array. In the fourth combination, the result is calculated like the third combination, but because one array is smaller than the other, the resulting array is the same size as the smaller input array.

In the following example, each iteration of the For Loop generates one random number stored in the array created at the border of the loop. After the loop finishes execution, the **Multiply** function multiplies each element in the array by the scaling factor. The front panel indicator then displays the array.



E. Graphs

A graph indicator is a 2D display of one or more data arrays called *plots*. LabVIEW features two types of graphs: XY graphs and waveform graphs. Both types look identical on the front panel of your VI. An example of a graph is shown below.

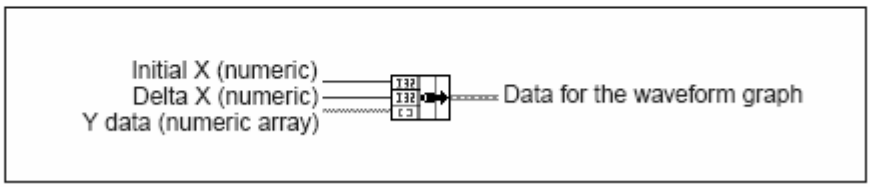


You obtain the waveform graph indicator from the **Graph** subpalette of the **Control**s palette. The waveform graph plots only single-valued functions with uniformly spaced points, such as acquired time-varying waveforms. The waveform graph is ideal for plotting arrays of data in which the points are evenly distributed.

Clusters

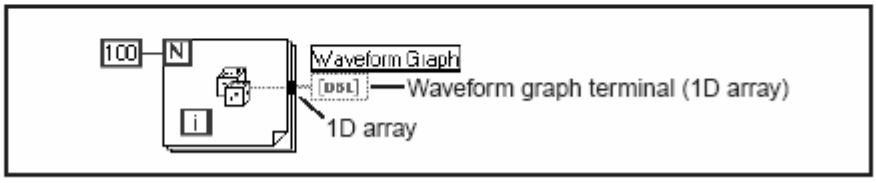
To use graphs, it is important to have a rudimentary understanding of another LabVIEW structure, the cluster. (Clusters are discussed in detail in the LabVIEW Basics II course.) A cluster is a data structure that groups data, even data of different types. You can think of a cluster as a bundle of wires, much like a telephone cable. Each wire in the cable represents a different element of the cluster.

The **Bundle** function (**Cluster** subpalette) assembles the plot components into a single cluster. For the waveform graph, the components include the initial X value, the delta X value, and the Y array.

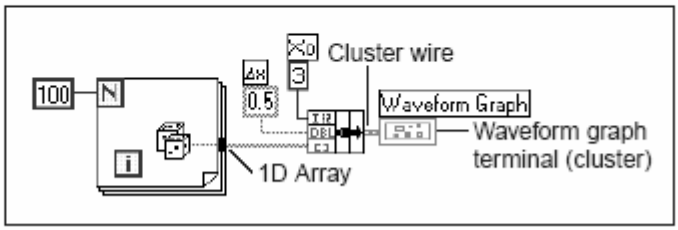


Single-Plot Graphs

For basic single-plot graphs, an array of Y values can pass directly to a waveform graph. This method assumes the initial X value and the delta X value are 0 and 1, respectively. The graph icon now appears as an array indicator.

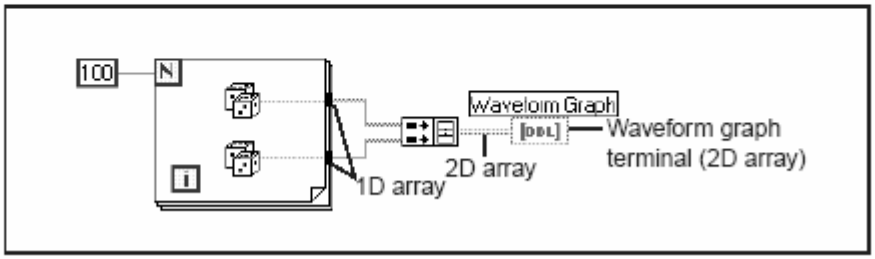


You can wire a cluster of data consisting of the initial X value, the delta X value, and a data array to the waveform graph. With this feature, you have the flexibility to change the timebase for the array. Notice that the graph icon appears as a cluster indicator.

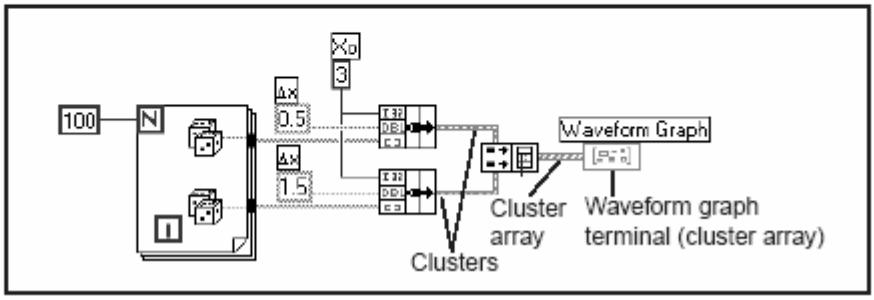


Multiple-Plot Graphs

You can pass data to a multiple-plot waveform graph by creating an array of the data types used in the single-plot examples above. The examples shown below detail two methods for wiring multiple-plot waveform graphs. As in previous examples, the graph icon assumes the data type to which it is wired.

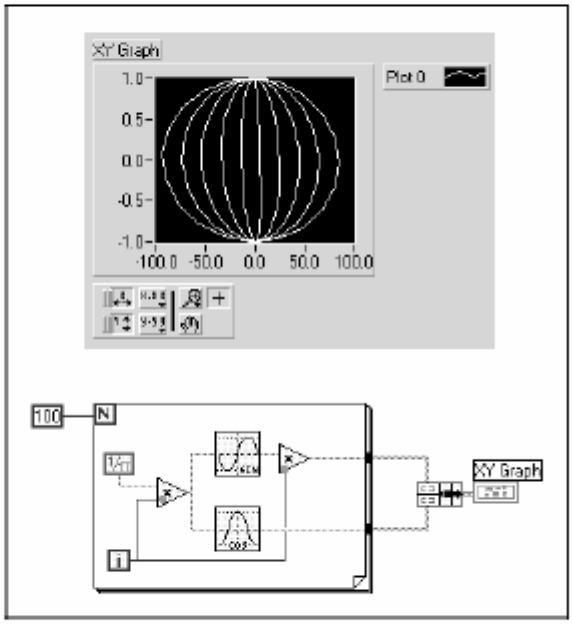


The example above assumes the initial X value is 0 and the delta X value is 1 for both arrays. In the following multiple-plot graph example, the initial X value and a delta X value for each array is specified. These X parameters do not need to be the same for both sets of data.



XY Graphs

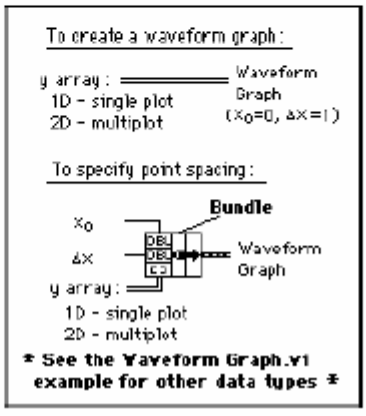
You obtain the **XY Graph** indicator from the **Graph** subpalette of the pop-up Controls palette. The XY Graph is a general-purpose Cartesian graphing object ideal for plotting multivalued functions such as circular shapes or waveforms with a varying timebase.



The **Bundle** function (**Array** subpalette) combines the X and Y arrays into a cluster wired to the XY graph. For the XY graph, the components are, from top to bottom, an X array and a Y array. The XY graph now appears as a cluster indicator.

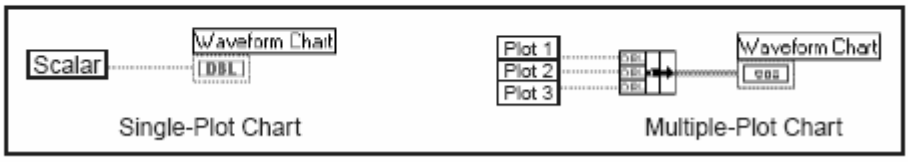
Chart and Graph Use Summary

When you first use the charts and graphs in LabVIEW, it can often be confusing when you try to wire data to them. Do you use a Build Array function, a Bundle function, or both? What order do the input terminals use? Remember that the Help window in LabVIEW contains valuable information—especially when you are using charts and graphs. For example, if you select **Show Help** from the **Help** menu and put your cursor over a Waveform Graph terminal in the diagram, you will see the following information:

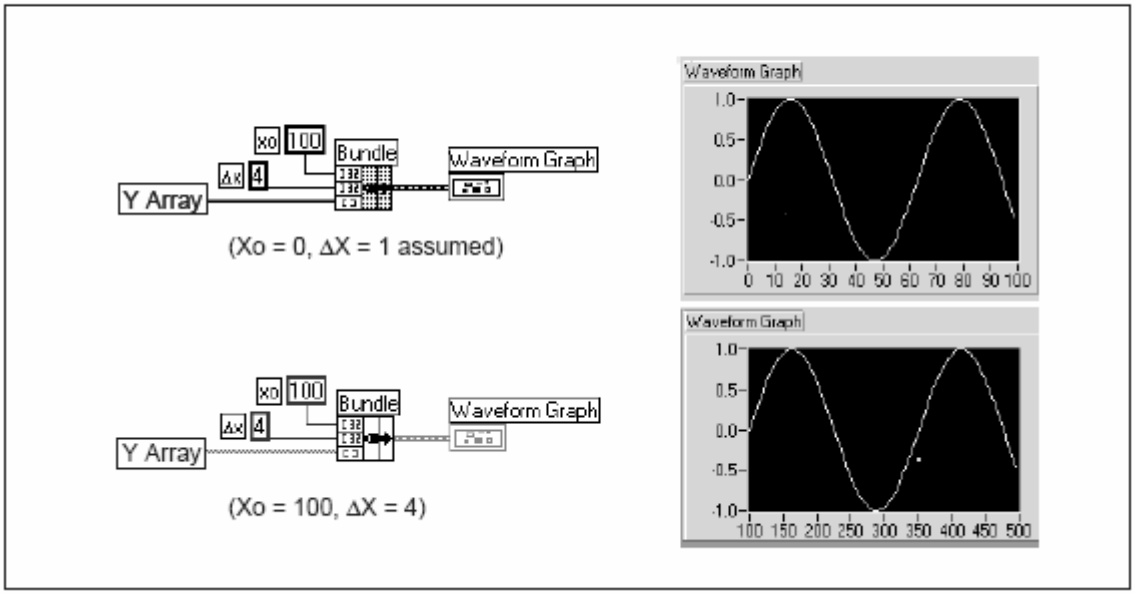


The Help window shows you what data types to wire to the Waveform Graph, how to specify point spacing with the Bundle function, and which example to use when you want to see the different ways you can use a Waveform Graph. These examples are located in the **Examples » General » Graphs » gengraph.llb** library. (The charts examples are in **Examples » General » Graphs » Charts.llb**.) The Help window shows similar information for XY Graphs and Waveform Charts. Also, you can use the next few pages in this course as a reference for wiring data to the various charts and graphs.

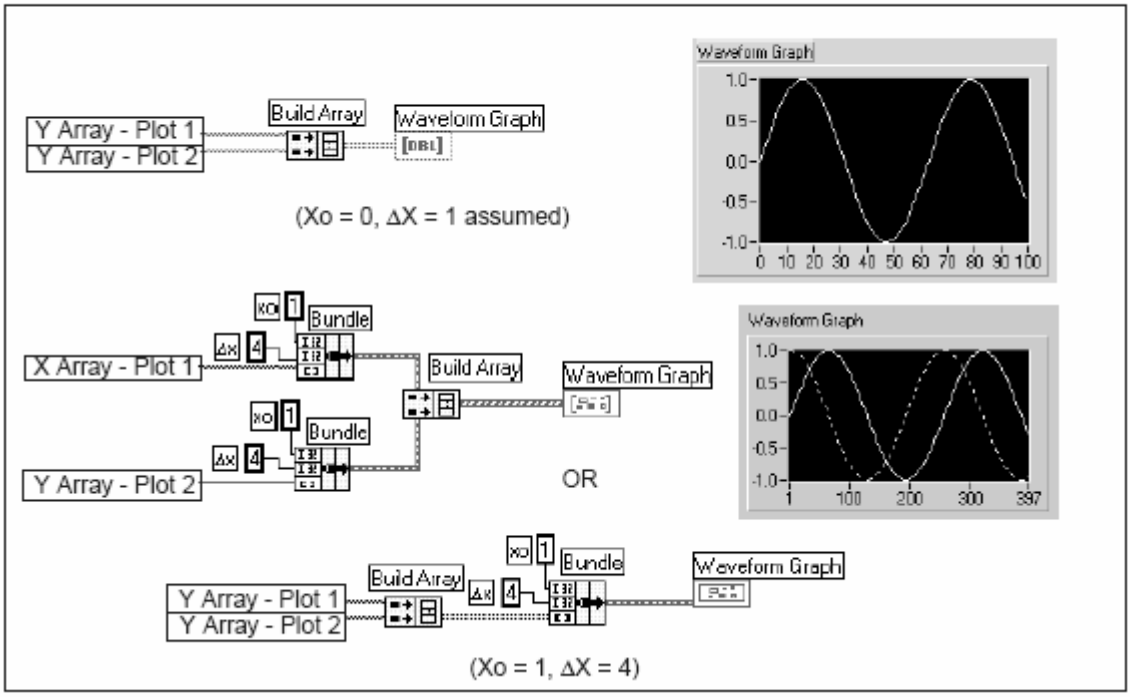
Waveform Chart



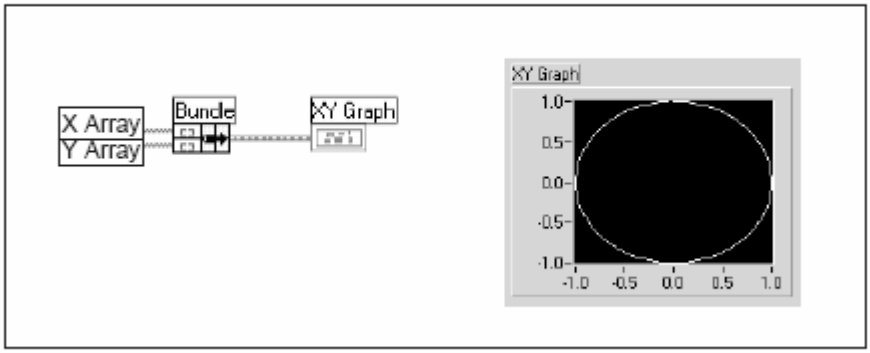
Single-Plot Waveform Graph



Multiple-Plot Waveform Graph



Single-Plot XY Graph



Multiple-Plot XY Graph