LabVIEW Day 3: Arrays and Clusters

Vern Lindberg

By now you should be getting used to LabVIEW. You should know how to Create a Constant, Control, or Indicator. I will assume you know how to create a new blank vi, and how to create functioning While Loops and For Loops, and how to create tunnels, either indexed or nonindexed. And you should know that the palette on the block diagram is called Functions and the palette on the front page is called Controls.

Data in LabVIEW can be single element—a Boolean, a floating point number, or a string—but often we want to deal with groups of data. Arrays and Clusters are two ways to group data. In an array all elements are of the same type, all integer or all string, etc. Arrays can be one dimensional such as (1, 1, 2, 3, 5, 8), two dimensional like a spreadsheet with rows and columns, or higher dimensional.

In a 2D array all rows must have the same length.

A cluster groups data of different types. A spreadsheet having one column of numbers and another of Booleans would need to be a cluster.

Elements of a cluster can be different lengths. For example the Graphical Data Type that you used last class with the Waveform Graph consisted of two scalars for start time and time interval, and a 1D array of data.

There can be Clusters of Arrays, and Arrays of clusters also.

1 Creating Arrays

1.1 Using loops to create arrays or clusters

One way to create an array is to use a loop, bringing data out through indexed tunnels.

(a) Place a While Loop on the block diagram. Inside the loop place a random number generator (Numeric→Random Number (0-1)) wire this to a tunnel at the right boundary, ensure it is an indexed tunnel, right click on the tunnel and create an indicator.
On the front panel expand the array indicator, and run the program. When you Stop the program, data is transferred to the indicator. You can use the Index Indicator to look at different elements. There will be a lot of them!

Look at the wire connected to the Array Indicator. It has the same color as the data type, but is thicker.

(b) You may want to know how many elements are in the array. Place an \textit{Array} \rightarrow \textit{Array Size} and wire the array to its input. Create an indicator and run the programming. There are a very large number of elements.

(c) Arrays can be of other types. Put a \textit{Comparison} \rightarrow \textit{Greater Than?} in the loop, and connect the top input to the random number and create a constant 0.5 to the lower input. Wire the output to a tunnel, create an indicator and run the program.

(d) How can we get these two arrays, each of the same length, linked together. Since they are different types of data we need a cluster. Place \textit{Cluster, Class \\& Variant} \rightarrow \textit{Bundle} in the loop and wire the the random number and the Boolean to the two inputs, wire to the boundary, make it indexed, and create an indicator. Run the program and see what the output looks like.

(e) We can create an array from two numbers, even if if one is integer and the other is floating point. Place an \textit{Array} \rightarrow \textit{Build Array} in the loop. Expand the Build Array to have two inputs. Wire the random number to one input and the index to the other. Inside the loop an array element of two numbers is made.

Notice that a coercion dot appears where the index is connected indicating a conversion from integer to floating point. We could use a \textit{Numeric} \rightarrow \textit{Conversion} \rightarrow \textit{To Double Precision Float} to explicitly do the conversion, thereby improving the programs use of storage and speed. It makes little difference here.

Wire through an indexed tunnel to an indicator. Notice that the line outside the loop indicates a 2D array. You can expand the indicator to see more elements of the 2D array.

1.2 Creating array constants, controls, and indicators

Sometimes we just want to have an array constant or control. The same general procedure is used, whether you get a constant or a control depends on whether you are the block diagram or the front panel.

(a) Here is how to create a constant on a block diagram. Place an \textit{Array} \rightarrow \textit{Array Constant} on the block diagram. Next tell what the content of the array is, for example to have
an array of strings, place a `String→String Constant` in the array an you have an array constant of String type.

(b) Initially the array is empty, but you can click on the cells and enter the desired values.

(c) To create an array control or indicator, do something similar, starting from the front panel. Place an `Array, Matrix, & Cluster→Array` on the front panel, then place a control or indicator of the desired type in the array. See Figure 1.

Fig. 1: Front Panel Arrays created by placing an Array shell on the panel, then placing the desired type of control or indicator in the shell.

(d) Finally, if you have an Array Control on the Front Panel, you can drag it to the block diagram. The indicator is unchanged, but a new Array Constant is formed on the block diagram.

### 1.3 Using Initialize Array

(a) Try `Array→Initialize Array`. When you first place it, it has two inputs and is configured to produce a 1D array of length `Dimension Size 0` with all identical `Elements`. Create controls for the inputs and try this.

(b) If you want a 2D array, expand the icon and add a constant or control for `Dimension Size 1`. This produces an array with `Dimension Size 0` rows and `Dimension Size 1` columns.

### 1.4 Build Array with Concatenate Inputs

The `Array→Build Array` operates in two modes. Try this.
(a) Create two 1D arrays using Initialize Array. Make the elements of one all 1 and the elements of the other all 2, and make them both of length 4.

Next place two Build Arrays and expand them to have two inputs. Wire the 1D arrays to the two inputs. Right click on the second Build Array and check Concatenate Inputs. Create indicators and run the program. In one case you should have a 2D array with one row of 1 and one row of 2. In the second case you should have a 1D array with the first few elements being 1 and the last few elements being 2.

(b) If the two input arrays are of different length, what happens for each mode? Make two array constants, (1,2,3,4,5) and (10,20) and look at the outputs for the two modes.

(c) Answer the following.

If two 1D arrays of lengths 2 and 4 are combined using a Build Array in default mode, the result is (a) A 1D array of length 6 (b) A 1D array of length 4 (c) a 2D array, 2 rows and 2 columns, with some data being discarded (d) a 2D array, 2 rows and 4 columns, with one row padded with 0 at the end, (e) a 2D array, 2 rows and 4 columns, with one row having empty (grayed out) elements at the end.

If two 1D arrays of lengths 2 and 4 are combined using a Build Array in Concatenate Inputs mode, the result is (a) A 1D array of length 6 (b) A 1D array of length 4 (c) a 2D array, 2 rows and 2 columns, with some data being discarded (d) a 2D array, 2 rows and 4 columns, with one row padded with 0 at the end, (e) a 2D array, 2 rows and 4 columns, with one row having empty (grayed out) elements at the end.

1.5 Extracting data from a 1D array

Use a loop to create a 1D array having elements \((2*i)\), \(i\) being the index, and length 6. Make an indicator for the values of this array.

(a) Now use Array→Index Array, wire the array to the Array input, and create a control for the Index 0, and an indicator for the output. The result will look something like Figure 2.

Run the program for different choices of the Index control and see what output you get.

(b) If you had a 2D array connected to the Index Array input with \(m\) rows and \(n\) columns, you would see a third input after Index 0, Index n-1. If you only connect to Index 0 with a value \(s\) the output would be row \(s\) of the 2D array.

If you want to pick out a column of a 2D array connect nothing to Index 0 and a control to Index n-1.
Fig. 2: Index Array allows extraction of parts of an array. For a 1D array it extracts a single element. For a 2D array it can extract a row, a column, or a single element.

(c) If you connect controls to both Index inputs you can pick out a single element by row and column.

### 1.6 Other array actions

You can look at the other icons in Array and get context help or detailed help on any of them.

Some interesting ones are Array Max and Min, Sort 1D Array, Transpose 2D Array, Array Subset and two on the Numeric menu, Add Array Elements, Multiply Array Elements.

If you multiply arrays, it is done element by element, so $C_{ij} = A_{ij} \times B_{ij}$. Often we want to do matrix multiplication instead. To do so, convert arrays into matrices using Array→Array To Matrix vi and a Numeric→Multiply.

### 1.7 Clusters

I have not said much about clusters. The only things from Cluster, Class & Variant that we will use at the beginning level are Bundle, Unbundle By Name and Unbundle. We have seen a cluster formed by a bundle earlier.
A useful applications of clusters is to group some inputs. Suppose we have the following controls: a string variable called Name, a switch (Boolean) called Increasing?, and an array called Sale Price. On the front panel these are 3 controls, and on the block diagram these would be 3 icons and 3 data lines. With three lines running all around our block diagram, it can start to look very confusing.

It can be convenient to make a cluster out of these. On the front panel deposit a Array, Matrix, & Cluster→Cluster. Either drag or deposit the various controls in the cluster. Switch to the block diagram and you will see a single icon for the cluster. The data can be transferred via a single line.

In order to extract the data, use a Cluster, Class, & Variant→Unbundle by name. The appearance on a block diagram and front panel are shown in Figure 3.

![Diagram](image)

Fig. 3: Making a cluster of related controls can clean up a block diagram. It is useful when all elements in a cluster belong together.

## 2 Updating a graph in real time

A Waveform Chart adds one new data point to those already plotted, and remembers all the data in a buffer. On Day 2 we had a Waveform Graph outside a loop, data were in an array, and were plotted only upon completion of the loop.

It would be nice to have a graph inside the loop that would update. The graph will be inside the loop, and each time through the loop we append a new datum to the previous array, and do the plot. To do this we will use a Shift Register.
2.1 Shift Registers

Shift registers are the means to access previous values of a variable in a loop. The previous value from the input shift register, say $A_{n-1}$, is used to calculate the next value, $A_n$, and the result is passed to the output shift register. This value becomes the input for the next time through the loop.

Sometimes we want access to more than one previous value, $A_{n-1}, A_{n-2}, \ldots$. For example, suppose we want to produce the Fibonacci series, $F_n = F_{n-1} + F_{n-2}$.

Start with a For Loop with a control to tell how many terms to compute. Right click on the boundary of the loop and *Add Shift Register*. Shift Register terminals appear on either side of the loop. This gives us access to $F_{n-1}$ however we also need $F_{n-2}$.

Hover over the input shift register, right click, and *Add Element* and you will have two inputs associated with one output.

In the loop, add the values from the two input shift registers to produce the new Fibonacci number, and wire to the output shift register arrow. This produces a recursion process to compute the series.

What remains is to set the initial conditions, and a means to see the result. The output of the shift register is a single number. We want an array that shows all the elements of the series that we have computed. You should know how to get a 1D array out of the loop.

I’ll let you experiment with what initial conditions are required to produce the series (1, 1, 2, 3, 5, 8, 13, …). My program is in Figure 4.

![Diagram](image.png)

This finds the first 15 members of the Fibonacci Series. It uses two shift registers to remember previous entries and adds them. The Series is taken out of the FOR loop using an indexed tunnel to produce a 1D array.

Fig. 4: To produce a Fibonacci Series, use a shift register with two elements to remember the last two values of the series.
2.2 Updating a Graph Using Shift Registers

Returning to the original task, we will compute the sum of the index, \( i \), and a random number then plot the sum versus the elapsed time, and expect to get a straight line with some noise.

(a) Create a While Loop and add a delay of 0.500 seconds to the loop and add a Shift Registe. Produce the new data point by adding the index to a random number.

(b) Outside the loop make an empty array constant of floating point numbers and pass into the input shift register. Inside the loop place a Build Array icon, expand to two inputs, wire the top input to the shift register, the bottom input to the new data point, and the output to the output shift register. Since the inputs were an array and a single number, the mode of operation automatically is Concatenate Inputs.

(c) Put a Waveform Graph on the front panel, and on the block diagram move it into the loop, and wire the Build Array output to the Waveform Graph input.

Fig. 5: Updating a graph inside a loop using shift registers. The Build Array is in Concatenate Inputs mode

Finally place a Waveform Graph on the front panel. On the block diagram make sure the icon is inside the loop. Bundle graph information (Start time of 0, wait time interval in seconds from the loop, and data from the build array), and connect to the graph.

My block diagram is shown in Figure 5. Run your program to see how it works.