

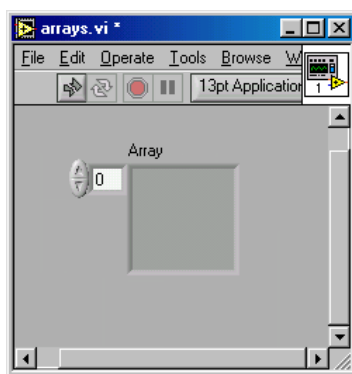
## Introduction to Use of Arrays

Arrays in LabVIEW require a special introduction: once you get the basic idea they are easy to apply, but the first few steps can be frustrating unless you have been introduced to some tips.

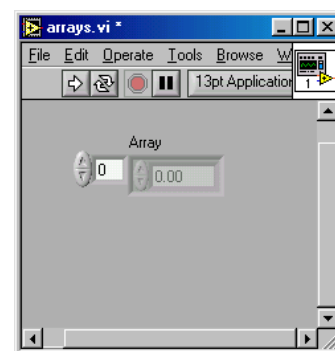
LabVIEW arrays are indexed with the first element having an index value of zero. Thus the last element in a 10 element array has index 9.

There are many built-in functions to make life easy. In my experience, when starting, it is best to build the simplest possible test VI to learn how each one works by trial and error. We will follow the same approach until we reach the main exercises. Then I would like you to try designing the VIs first before you produce the code.

Start with a blank VI.

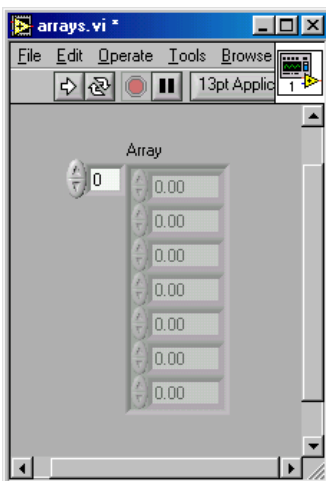


- 1) Place an array on the panel. (Select "All controls" and go to the left end of the second row....*Arrays and Clusters*)



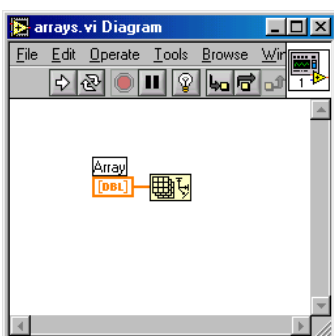
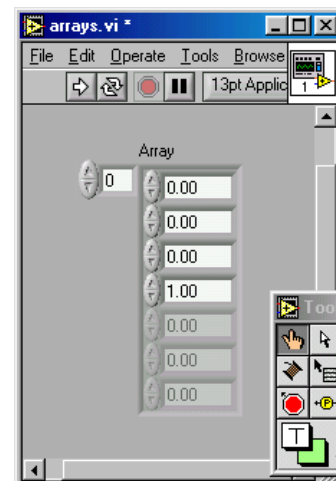
- 2) Look at the diagram: the terminal is black, meaning that it does not yet have a variable type (colour).

- 3) Now, back on the panel, right-click *inside* the array box and select a digital control from the numeric palette. Place this control inside the array box. The array re-sizes itself to accommodate the control.



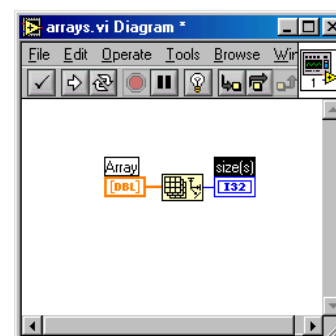
- 4) Now select the bottom right corner of the array border and drag it down to expose more of the array elements – see picture to the left.

- 5) At the moment, all the elements are empty: they are greyed out. However, select the *finger tool* and click the fourth element down, and then make it 1.0. Now the other elements (0..2) will be made zero. The elements that are defined will be changed from their original grey colour – see right picture.



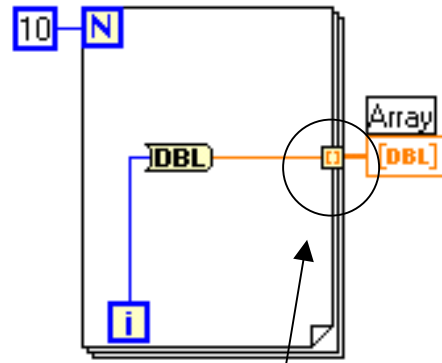
- 6) Now, on the diagram, right-click to obtain the functions palette, and choose the array size function from the 'arrays' sub-palette. The diagram now looks like this:


- 7) With the pointer (arrow) tool, right-click the right hand end



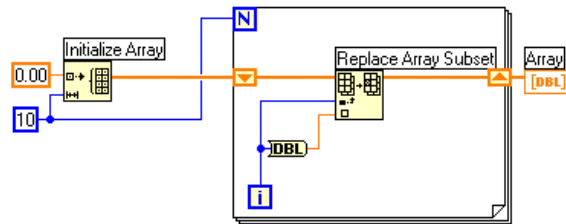
of the array size block and select the create..., then select 'indicator'.

- 8) Now run the VI from the front panel. The correct array size will be reported.
- 9) Next, we are going to see how an array can be easily created using a simple program: a for loop.
- 10) Create a new VI, and build it like this diagram.

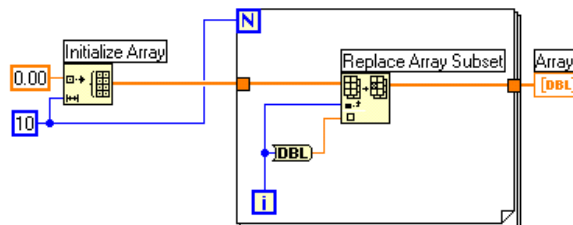


- 11) Run the VI. The result is rather unspectacular. However, using the finger tool, press the array index buttons (to the left of the array index) and examine each of the array elements. Confirm the extent of the array by using the arrow tool to drag the lower right corner of the digital display of the first element.
- 12) Notice how the point where the 'value' wire leaves the 'for' loop has a little  symbol. This means "auto indexing" is on. The single variable (thin wire) becomes an array value (thick wire). If you right click on this box you can turn auto-indexing on or off. We can do the same operation in a rather more complex way:

- 13) Here we create the array first with 'Initialize Array' (notice that I have made the label for each function block visible). All the elements are initially zero. Then we replace each element in the array with the index value.



- 14) Notice how the shift register is essential here. We are replacing each element one by one, and we want to retain the partially built array each time we go round the loop. If we simply wired the array through the loop border like this we get quite a different result.



- 15) Work out why we get this result. If you are not sure, try changing the value with which each array element is initialized to, say, 25.0. Also, slow the loop down with a millisecond wait function – use a wait time of 500 msec. Finally, add array indicators to display the array value inside the loop. Then the answer might be more obvious.

- 16) Notice how 'array indexing' has been disabled where the array enters and leaves the loop. To do this, right click on the small square where the array wire enters or leaves the loop and select 'indexing disabled'.

Now you have finished the basic exercises.

### A Word of Warning

Try to avoid using the 'Build Array' function, and also the 'Insert into Array' or 'Delete Array Subset' functions, or replacing an array subset with a sub-array which is different in size to the section being replaced. Excessive use of all these operations can result in fragmentation of the memory space your program uses. Your program will also run slowly and will be inefficient.

## Array Exercise 1

*Do these exercises when you have completed the first part of this tutorial. Work with pencil and paper first and design the code you intend to use. Use an eraser when you need to correct minor mistakes, and write notes as you develop your ideas. Don't be afraid to re-start the design again: if you do don't erase what you have done – start a new set of drawings.*

A useful tool in many control applications is a circular buffer. This is an array into which we write values one at a time. However, when we have filled the array, we continue writing from the beginning of the array again. This way, we always have a given number of *the most recent* values of the input signal.

In this exercise you will develop a simple VI that does the following:

- 1) Initialize a 100 element array to the value NaN (not a number – see numeric constants palette, or just type 'nan' into a numeric constant box).
- 2) A continuously repeating loop, executing once every 50 msec, that uses the array as a circular buffer. It writes a value  $y$  into the buffer:

$$y = a \sin(bt) \quad (a = \text{amplitude}, b = \text{frequency, both from user-settable controls, } t \text{ is time value incremented by } 0.05 \text{ each time the loop repeats})$$

Each time the loop repeats it plots the circular buffer array on a waveform graph (note: make sure you select 'waveform graph' not the 'waveform chart' display module).

Confirm that the program runs correctly: the sine wave should appear on the graph, and the new values will overwrite the old values progressively.

## Array Exercise 2

- 3) Now modify the program to extract *the last  $n$  data points* which have been written into the circular buffer. This is not so easy. You have to think really carefully what this means.....

The value of  $n$  should be a variable set by the user. It will be less than or equal to 100, the size of the circular buffer.

If the *latest* value added to the buffer has an index value in the buffer which is less than  $n$ , then it follows that some of the most recent data points will be *at the other end of the buffer array*. This means that you have to retrieve some of the data from one part of the buffer and the rest from another part, and join the two blocks together to form a single array containing the most recent data.

Confirm that this works by displaying the last  $n$  points on the graph. Now the graph behaves more like a chart, moving backwards with time.

## Array Exercise 3

This is for those who want to develop good array skills in LabVIEW.

Re-work both exercises above, but this time, use a two dimensional array for the circular buffer so that each data point can consist of multiple values. Demonstrate this plotting three different functions (e.g. sine waves with different phase, amplitude and frequency).