SYNTHIAM

synthiam.com

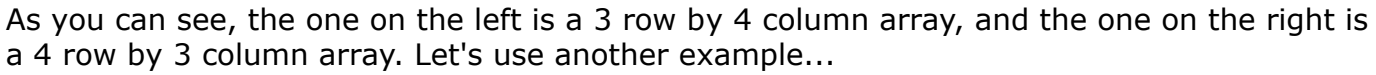
Explanation and Uses of Arrays.

This Tutorial will explain what an "Array" is, and give you some examples of how they are, or can be used within EZ-Builder and your robotics projects. This is not a "How to" tutorial, but more of a guide explaining how how arrays are used in various applications.

Last Updated: 10/16/2015

Step 1. Expaining an Array.

Arrays are found in different subject matter such as mathematics, electronics, computer programming, and are formed as an arrangement of similar objects, and sometimes have varying meanings.
One way of describing what an array is, is that an array is basically a fixed size grid made up of rows and columns, and these rows and columns contains an arrangement of a same kind of object or values which can be numbers, colours, data, or other objects. As a real world example, the seats in a cinema are arranged in rows and columns. It's the arrangement of these objects (seats) that is defined as an array. In short, an array is a object or data structure.
An array is not classed as an object as such, more of an index that contains objects in a fixed size grid of rows and columns.
ROW: A horizontal line of objects.
C:
O:
L:
U:
M:
N: A vertical line of objects or an upright support.
As a visual example, take a quick look at the following image...



As you can see, the one on the left is a 3 row by 4 column array, and the one on the right is a 4 row by 3 column array. Let's use another example...

Revolution Six has six legs made up of servos. Each leg has two servos which makes up 1 row and 2 columns (or vise versa), so you could say that "Six" has a collection of 6, 2x1 servo arrays, or in total, a 6x2 servo array (6 rows of 2 columns of servos).
With the basics of explaining what an array is done, the following few steps will show you some simple examples of using arrays with your robot and EZ-Builder.

Step 2. LED Array and Matrix

This step will give a short example of an LED array. An LED array is a series of LED's on a flat panel that form rows and columns. These can be made to display patterns or images (depending on how many LED's make up an array) either by using switching circuits and/or software like the RGB Animator found in EZ-Builder.

With an RGB (red, green, blue) LED panel, each bulb (diode) has the same characteristics (they produce light and use the same power), but have different values (each RGB LED is made up of 3 different colour LED's), but as the characteristics and function are the same, it is still classed as an array. Now, if you were to chain multiple LED arrays together, this would form a matrix. In simple terms, a matrix is made up of multiple arrays. Let's look more closely at the RGB LED array seen above...

The RGB LED Block is made up of 18 RGB LED's which adds up to 54 LED's in total (18 red, 18 green, 18 blue). The two sets of 3x3 RGB's you can see in the picture would make up a matrix, right? Not really, as both equal groups of bulbs are arranged in equal rows and on the same board which forms an array.
To use another type of LED LED array, below you will see on 8x8 LED array. Four of them chained together becomes an array of arrays. This is now a matrix.

An array, is a fixed object or data structure.
A matrix, is made from multiple arrays. (Matrix has different meanings in different subjects such as maths, science and biology).
The next step will look at another hardware array.
RGB Animator

Step 3. Sensor Arrays.

Sensor array, is probably a term you may have heard mentioned in your favourite science fact or science fiction TV show or movie. Simply put, a sensor array is a group of individual sensors working in tandem to monitor what the signals that they are designed to measure or monitor. One such example could be an array of ultrasonic distance sensors.

One sensor running will monitor or measure an object and report the information back to its controller, such as an EZ-B which then sends the information back to EZ-Builder. But if this sensor is on a wide base of a robot, an object may not be detected, as demonstrated below...

The advantage of having multiple sensors in an array, is that having them located across the entire width of the base, if one sensor does not see an object slightly off to one side of it's detection range, another one in the array will...

As you can see, the five sensors are sending out an ultrasonic signal to detect the object. Four of them do not see the object, but one does as it is in it's line of sight and therefore sends back an ultrasonic echo to confirm that an object has been seen.
To make the sensor array detect objects, we need to make a script array, which is explained next.

Step 4. Script Array.

So we have our ultrasonic sensor array set up on our robot, but we need a script to make the sensor array work. If we had one sensor fitted to our robot, we could have the following script running...
code# Monitor 1 ultrasonic sensor for detection

mode. When a value is less than 30,

it will execute the "Detection" part of the script.

:Loop
Forward()
$ping1 = GetPing(d1, d2)

This "IF" script command will be referred to as

an object for an array.

if ($ping1 < 30)
Goto(Detection)
endif
Sleep(1000)
Goto(Loop)
:Detection
Sleep(1000)
Say("Object detected.")
Left(70,2000)
sleep(2000)
Goto(Loop)[/code]
A simple script monitoring one sensor that takes action by saying that an object has been seen, turns left at a low speed for two seconds, then starts monitoring again. This is not really an array yet, as the script only has one "object" to detect one sensor.
We need to set up an array script, so we start with the following...
Forward()
$ping1 = GetPing(d1, d2)
$ping2 = GetPing(d3, d4)
$ping3 = GetPing(d5, d6)
$ping4 = GetPing(d7, d8)
$ping5 = GetPing(d9, d10)[/code]
This is not an array yet, just the start of one. So now we need to add some "Objects" to make it an array...
codeif ($ping1 < 30)
Goto(Detection)
endif
if ($ping2 < 30)
Goto(Detection)
endif
if ($ping3 < 30)

Goto(Detection)
endif
if ($ping4 < 30)
Goto(Detection)
endif
if ($ping5 < 30)
Goto(Detection)
endif
Sleep(1000)
Goto(Loop)[/code]
Now we have five objects that have the same characteristics (they all monitor sensors), but have different values (they have different names and one will take action when the others don't). We have now formed an array script. We just need to finish off be making the robot do something when an object is detected...
Sleep(1000)
Say("Object detected.")
Left(70,2000)
sleep(2000)
Goto(Loop)[/code]
So, to put it all together, you will have a fully operational sensor and script array...
code# Monitor the 5 ultrasonic sensors for detection

mode. When a value is less than 30,

it will execute the "Detected2 part of the script.

:Loop
Forward()
$ping1 = GetPing(d1, d2)
$ping2 = GetPing(d3, d4)
$ping3 = GetPing(d5, d6)
$ping4 = GetPing(d7, d8)
$ping5 = GetPing(d9, d10)
if ($ping1 < 30)
Goto(Detection)
endif
if ($ping2 < 30)
Goto(Detection)
endif
if ($ping3 < 30)
Goto(Detection)
endif
if ($ping4 < 30)
Goto(Detection)
endif
if ($ping5 < 30)
Goto(Detection)
endif
Sleep(1000)
Goto(Loop)
:Detection
Sleep(1000)
Say("Object detected.")

Left(70,2000)
sleep(2000)
Goto(Loop)[/code]
In the next step, we will go over the basics of variable arrays.

Step 5. Variable Arrays.

Before we go over variable arrays, we'll quickly go over what a variable is.
The easiest way to describe a variable, is to compare it to a container. A container stores an item. You can open the container and change the item, but it will only hold one item. A variable is a memory location that stores a data value. This data value can change, but the variable can still only contain one data value.
So where does an array come in to it? As we have seen above, a single variable is a memory location, but putting a group or series of these memory locations together will create an array. Let's look back at part of our ultrasonic sensor script again...

```
$ping1 = GetPing(d1, d2)
$ping2 = GetPing(d3, d4)
$ping3 = GetPing(d5, d6)
$ping4 = GetPing(d7, d8)
$ping5 = GetPing(d9, d10)
```

Each "**GetPing()**" line is the memory location for each variable, and each one has their own unique name "Ping1, 2, 3, 4, and 5".

```
if ($ping1 < 30)
Goto(Detection)
endif
if ($ping2 < 30)
Goto(Detection)
endif
if ($ping3 < 30)
Goto(Detection)
endif
if ($ping4 < 30)
Goto(Detection)
endif
if ($ping5 < 30)
Goto(Detection)
endif
```

We now have the memory locations put together, which now makes our variable array. When the script is run, the value of each variable is can also be displayed, including the values within arrays, in the variable watcher control seen below...

Step 6. Array Script Commands.

This step will explain what the "Array" script commands do. We will use the "Append Array" project example that can be found in EZ-Builder, and give some quick demonstrations of each array command.
Append Array: This creates an array of the variable to the specified size. (This will add a new line to the variable watcher as seen in this step).
Define Array: Describes the nature of the array with variable name.
Fill Array: Fill an existing array with the default value.
Get Array Size: Returns the size of the specified array variable.
Append Array and Define Array.
The "Append Array" example consists of two control which are a "Script Control", and a "Variable Watcher" control. Clicking on the script configuration tab, you will see the following script. Have a look through it and read the comments prefixed with a **#** tag which will explain what each part of the script does...

```
ClearVariables()
# This will clear the list on the "Variable Watcher" .
DefineArray($x, 1, "orange")
# "DefineArray" Describes the nature of the array with variable.
# "$x" is the variable.
# "1" is the size of the array, which here is one line
# And "Orange" is the and a value name,
repeat ($z, 0, 5, 1)
# The "$z" variable is added to make two variables...
# two variables = 1 array.
#$z is part of the array will repeat.
# "0" is the first array line on the variable watcher.
```

"5" will repeat the array this many times.

"1" multiplies the previous array, 5 times in this case.

AppendArray($x, $z)

When this runs, it will add a new line to the variable watcher.

endrepeat

The repeat will now end.

The following message will display in the script window.

print("Now look at the Variable list to see how the array grew from the initial defined size")
```

So you can see the "Append Array" and "Define Array" commands are being used here. Let's take a look at the script when it runs...

We see in green, that the "Z" variable will repeat the "X" array 5 times. In orange, the array runs the called "Orange" has six variables in it. And in yellow, the "X" array runs for the first time, then after that it repeats five more times.

**repeat ($z, 0, 5, 1)**

**!** The "$z" variable is added to make two variables (two variables = 1 array).
**!** $z is part of the array will repeat.
**!** "0" is the first array line on the variable watcher.
**!** "5" will repeat the array five more times.
**!** "1" multiplies the previous array, 5 times in this case.

**Get Array Size**

This command will display how many values there are of the array that is defined. For this example, we add a script control which will contain the array size script. Below is an example of this...

```
We first need to set a variable for the "Get Array Size" command and add the array name...
$y = GetArraySize("$x")

The variable will be called, which will get the size of the array that was just run.

if($y)
Print("Now look in the Variable Watcher, and look for variable Y.")
endif
```
```

So we run the repeat array script (seen on the left) then we run the array size script...

You can see on the image on the right, that there is a new line in the variable watcher. This is telling us the full size of the "X" array which we can see, is "7" (seven lines in the array).

Fill Array

The final array script command will fill an existing array with the default value. So for this final example, we add another script control, and add the following script...

```
FillArray($x,"Hello")
```

When the "X" array has been run, we start the "Fill Array" script and we get the following...

You can see that the values have changed to what we have put in the script. That finishes our short example of variable arrays and almost brings us to the end of this "Arrays" tutorial.

Conclusion.

Before we finish, here is a few definitions of the word "array" in different subjects...
In maths, an array is an arrangement of numeric values or symbols, arranged in rows and columns.
In computing, an array is a series of objects that are the same size and type, or an indexed set of related values.
And in general, the term "array" is used to describe an impressive or vast amount of a particular type of item. *He had a vast array of science fiction movies on his stall.*
Remember, if you have questions or are unsure of anything, you are always welcome to ask in the Community Forum.
Happy building.
Tutorial created on 12th October 2015