EasyLanguage : Understanding Arrays

This page last changed on Feb 02, 2010 by 9275.

Array Concept

The Array concept it is actually very simple.
The attempt of this page is to demystify the concept with simple examples. Please feel free to edit as well as to add alternate methods to clarify the content

Think of an array as an Excel spreadsheet, where you have rows and columns.
It is from this approach that the following examples are designed.

Back to top

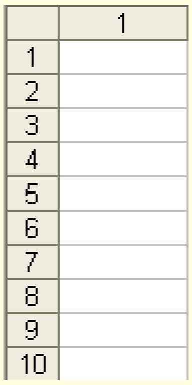
Incrementing : For Counter = 1 to Array_Size begin... end;
Decrementing : For Counter = Array_Size ***DownTo*** 1 begin... end;

One Dimensional ARRAYS

With one dimensional arrays you get one column with X number of rows to query.

Array: myArray **[10] (0)**;

- the **[10]** defines one column with 10 rows (not to confuse but a 0 row is actually available too to make 11)
- **(0) =** Numeric Data
- **(Text) =** would = String Data could be entered.



	1
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Back to top

Multi-Dimensional Arrays

Multi-Dimensional arrays enable you to store vast amounts of data more similar to a spreadsheet, thus you could add a specific series of data per bar or per condition to back reference for compare.

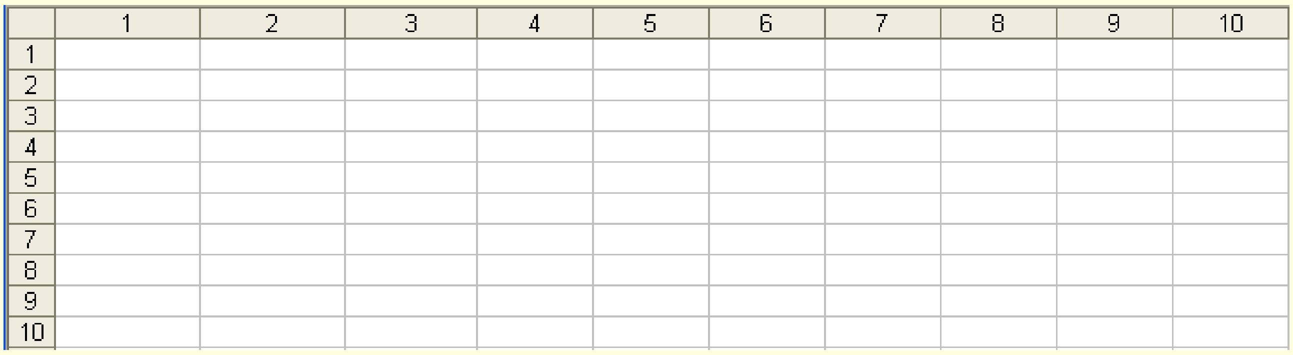
Warning
More standard is the *two dimensioned array*, but since you can do more a warning must be stated here in using a *three dimensional array*
For example 3 dimensioned arrray would be myArray[10,10,10]; You should have your reasons well in order in advanced as improper dimensioning could easy take up too much memory as well as calculations time could be increased exponentially . The conceptualization of a three dimensional Array would be a spreadsheet cell hyperlinking to another completely new spreadsheet. A better example may be a pallet of boxs 10 across, 10 high and 10 deep. It starts getting very complicated at that level.

Two Dimensional Array

Array: myArray **[10,10] (0)**;

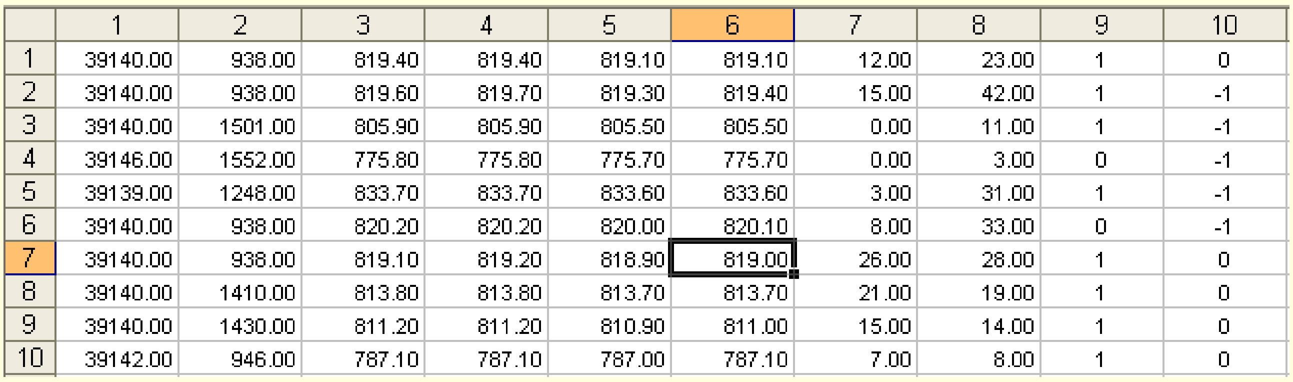
the **[10,10]** defines 10 columns with 10 rows (or not counting the 0 row, 100 field to populate with data)

- **(0) =** Numeric Data
- **(Text) =** would = String Data could be entered.



	1	2	3	4	5	6	7	8	9	10
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										

So if you ask for
Value1 = myArray*[6,7]*; then you are looking for column 6, row 7's data.



	1	2	3	4	5	6	7	8	9	10
1	39140.00	938.00	819.40	819.40	819.10	819.10	12.00	23.00	1	0
2	39140.00	938.00	819.60	819.70	819.30	819.40	15.00	42.00	1	-1
3	39140.00	1501.00	805.90	805.90	805.50	805.50	0.00	11.00	1	-1
4	39146.00	1552.00	775.80	775.80	775.70	775.70	0.00	3.00	0	-1
5	39139.00	1248.00	833.70	833.70	833.60	833.60	3.00	31.00	1	-1
6	39140.00	938.00	820.20	820.20	820.00	820.10	8.00	33.00	0	-1
7	39140.00	938.00	819.10	819.20	818.90	819.00	26.00	28.00	1	0
8	39140.00	1410.00	813.80	813.80	813.70	813.70	21.00	19.00	1	0
9	39140.00	1430.00	811.20	811.20	810.90	811.00	15.00	14.00	1	0
10	39142.00	946.00	787.10	787.10	787.00	787.10	7.00	8.00	1	0

Back to top

Populating Arrays

Arrays values hold from bar to bar.
An array can be populated by a hard number as in:
myArray[**4**] = High;

or by a Counter method that you predefine variable name is arbitrary as long as the value is not higher than the declared dimensional value.

Var: Counter(0);

Counter = Counter +1;
myArray[**Counter**] = High;

Back to top

Shuffling Array Data

Shuffling simply moves the first row of an array to the second and so forth leaving the Arrays first row open for a new entry.
Normally this is done in one step or procedure.

Single Dimension Array Shuffle

```
Array: myArray [10] (0);
Var: myArraySize(10);
Var: Counter1(0);
Var: HoldRow(0), SaveValue(0);
SaveValue = Value1; {your value or caluclation you want arrayed}
For Counter1 = 1 to myArraySize begin
```

```
        holdRow = myArray[Counter1];  {save the current Value of this row for the next row }
        myArray[Counter1] = SaveValue; {pass the newer Value to this row}
        SaveValue = holdRow;       {pass the old Value of this row for the next loop}


end;
```

Multi-Dimensioned Array Shuffle

The code below will shuffle a multi-dimensional array. A new row will be inserted in position 1 and roll the remaining rows back one row each.

```
Array: myArray [100,10] (0);
Var: myArraySize(100);
Array: Save[10](0);   {used as holders for the shuffle}
Array: Hold[10](0);   {used as holders for the shuffle}

Var: Z(0), X(0);      {Using Single Letters as counters}


{Pass your newest set of Values to a one dimensional Array}

If condition1 then begin
        {Sample Data}
        Hold[1]= Date;
        Hold[2]= Time;
        Hold[3]= Open;
        Hold[4]= High;
        Hold[5]= Low;
        Hold[6]= Close;
        Hold[7]= UpTicks;
        Hold[8]= Downticks;
        Hold[9]= CurrentBar;
        Hold[10]= close -close[1]0;

        for Z = 1 to myArraySize begin

                For X = 1 to 10 begin Save[X] = myArray [Z,X];  end;  {pass Array Row to Save
Array variable LOOP}

                For X = 1 to 10 begin myArray [Z,X] = Hold[X];  end;  {pass Hold Array
variables row  LOOP}

                For X = 1 to 10 begin Hold[X] = Save[X];          end;  {pass the SaveArray to
Hold Array LOOP}

                if Hold[1]= 0 then BREAK;      {Get out of main LOOP if passed row /Hold Array
is Empty }

        end;
end;
```

Circular Arrays

This is an example of a First-In-First-Out (FIFO) circular array. Using arrays in this manner is much faster then 'shuffling' because you are not constantly moving all the array data around. The disadvantage of this approach is that references to previous values (the equivalent of data[1], data[2] etc) is more difficult.

```
//Circular Array - First-In-First-Out (FIFO)
```

```
//Code works by ReadIndex always being one bar behind WriteIndex

Inputs: myArraySize(10);

Vars:   ReadIndex(0),
        WriteIndex(1),

        ArrayHasData(false),

        MyValue(0);

//Must declare our array size bigger than the myArraySize input
Arrays:  myArray[100](0);

//this is where you are putting your data into the array
//we test 'condition' to see if we should be putting data in this bar
if {condition} then begin
        myArray[WriteIndex] = close;

        WriteIndex = WriteIndex + 1;
        ReadIndex = ReadIndex + 1;

        ArrayHasData = true;
end;


//this is where you get your data out of the array
if ArrayHasData then begin
        MyValue = myArray[ReadIndex];
end
else begin
        //your array is currently empty
 //MyValue = -999999;  //or something similar
end;


//check for 'rollover' of our read and write index counters
if WriteIndex > myArraySize then WriteIndex = 1;
if ReadIndex > myArraySize then ReadIndex = 1;
```

Back to top

Related Pages

- Arrays reserved word
- Comparing the performance of variables vs. arrays vs. ELC
- Programming Reference - Arrays - Static
- Programming Reference - Arrays - Dynamic
- Passing Arrays to DLLs