



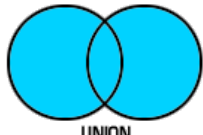
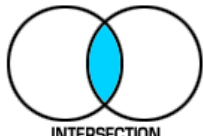
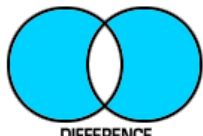
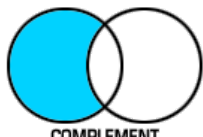
Products Services Developer Resources Contact STS About STS



## Creating 'Union', 'Intersection', 'Difference' and 'Complement' Arrays



There are times when you have two arrays and you need to manipulate them to get a single array that contains the results that you are looking for. If you think about it, manipulating two arrays is nearly identical to operations performed using Venn Diagrams. If you remember back to your Venn Diagram days, you'll remember that these four types of set theory methods are defined as follows:

Method	Description	Logical Equivalent
 UNION	Contains all data from both arrays, eliminating any duplicates	A OR B
 INTERSECTION	Contains only the data which is common to both arrays"	A AND B
 DIFFERENCE	Contains only the data which is <i>not</i> common to either array (basically the opposite of Intersection)	A OR B AND NOT(A AND B)
 COMPLEMENT	Contains only data that is in one array, but <i>not</i> in the other array	A AND NOT(A AND B)

For the purposes of this tip, we are going to work with two arrays, **FruitA** and **ColorsA**. (Why do both names end with "A"? Well two reasons: (1) it is a style convention I'm used to where all array variables end in "A" so I can recognize them when scanning a script, and (2) "Colors" is a keyword in Revolution, so you can't use it as an array name.)

Assume that the array **FruitA** contains the following elements (keys):

- apple
- orange
- pear

... and the array **ColorsA** contains the following elements:

- red
- orange
- green

Assume that for the purposes of this tip we have set up FruitA and ColorsA as follows:

```
put "" into FruitA["apple"]
put "" into FruitA["orange"]
put "" into FruitA["pear"]
put "" into ColorsA["red"]
put "" into ColorsA["orange"]
put "" into ColorsA["green"]
```

(Note: Since we only care about the *keys* of an array, not the *contents* of each element, it doesn't matter what you put into each array element, so I'm using empty.)



Union

Revolution provides the union command in order to get the union of two arrays, with the syntax:

```
union Array1 with Array2
```

Keep in mind that this *changes* Array1 to now contain the union of both arrays, so if you need to keep Array1 unchanged, make sure you copy Array1 into another variable before you do the union. Here's how I would do that with our sample arrays:

```

put FruitA into FruitUnionColorsA
union FruitUnionColorsA with ColorsA -- changes FruitUnionColorsA to be the union of FruitA and ColorsA
put the keys of FruitUnionColorsA

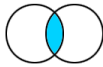
```

Which puts the following into the message box:

```

apple
orange
pear
red
green

```



### Intersection

Like union, Revolution provides the `intersect` command in order to get the intersection of two arrays, with the syntax:

```
intersect Array1 with Array2
```

Also like `unionArray1` will be changed to contain the intersection of both arrays. Here's how to use `intersect`, maintaining `FruitA` for use later:

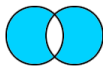
```

put FruitA into FruitIntersectColorsA
intersect FruitIntersectColorsA with ColorsA -- changes FruitIntersectColorsA to be the intersection of FruitA and ColorsA
put the keys of FruitIntersectColorsA

```

Which puts the following into the message box:

```
orange
```



### Difference

Unfortunately Revolution does not have a simple command for this, so a handler is necessary. What this function does is to get the intersection of both arrays, and then remove those elements from each array, finally joining them together with union and returning the combined array as the returned value.

```

function difference pArray1,pArray2
  put pArray1 into tempA
  intersect tempA with pArray2
  repeat for each line tKey in (the keys of tempA)
    delete local pArray1[tKey]
    delete local pArray2[tKey]
  end repeat
  union pArray1 with pArray2
  return pArray1
end difference

```

And here's how we'd use it:

```

put difference(FruitA,ColorsA) into FruitDiffColorsA
put the keys of FruitDiffColorsA

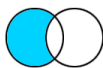
```

Which puts the following into the message box:

```

pear
green
apple
red

```



### Complement

Revolution also does not have a simple command for this, but the approach is nearly identical to *Difference*, above, with two exceptions: (1) we don't care about the second array's data, and (2) we need to identify from which array we want the complement. Here's the code:

```

function complement pArray1,pArray2 -- the first param is the array we want back; the second we don't care about
  put pArray1 into tempA
  intersect tempA with pArray2
  repeat for each line tKey in (the keys of tempA)
    delete local pArray1[tKey]
  end repeat
  return pArray1
end complement

```

And here's how we'd use it:

```

put complement(FruitA,ColorsA) into FruitWithoutColorsA
put the keys of FruitWithoutColorsA

```

Which puts the following into the message box:

```

pear
apple

```

Well, there you have it... have fun working with arrays!

Posted 2/7/06 by Ken Ray

 [Print this tip](#)

---

[News and Rumors](#)

[Products](#)

[Services](#)

[Developer Resources](#)

[Contact STS](#)

[About STS](#)

Copyright ©1997-2013 Sons of Thunder Software, Inc. All rights reserved.  
Send all comments to [webmaster@sonsothunder.com](mailto:webmaster@sonsothunder.com).

---