

# Arrays

## Basics

Arrays in RouterOS can be normal indexed arrays or associative (key/value pair arrays).

### Indexed array

```
# define a variable to play with
:global array1

# a normal array with no keys
:set array1 {1;2}

# referencing an array by index
:put ($array1->0)
:put ($array1->1)

# iterate over each element with for
:for x from=0 to=[:len $array1] step=1 do={
  :put ($array1->$x)
}

# iterate over each element with foreach
:foreach a in=$array1 do={
  :put $a
}
```

### Associative (key/value pair) arrays

```
# define a variable to play with
:global array1

# an associative array, or key/value pairs
:set array1 {a=1;b=2}
```

```
# referencing an array value by key
:put ($array1->"a")
:put ($array1->"b")

# iterate over each pair
:foreach k,v in=$array1 do={
  :put ($k . " -> " . $v)
}
```

## :tostr

Sometimes RouterOS returns a value in a format that you're not expecting, and strange things will happen because you don't understand how RouterOS is handing you data. This primarily happens when you think RouterOS is going to give you a string, but its actually giving you an array.

If you run values through **:tostr** before using them, fewer unexpected things will happen to your script.

The example below shows us getting the neighbor interface for a specific device. If the interface is a member of a bridge, the result we get looks like "[bridge\_port\_name];[bridge\_interface\_name]" when using **:put**. If the interface is not part of a bridge, the result we get looks like "[interface\_name]" when using **:put**. If you were to save that result to a global variable, and then do **:env print** you'll see that what we received both times is actually being stored as an array {"ether5";"bridge-lan"} and {"ether5"} respectively.

When you use **:put** with a variable that is storing an array, interesting things can happen, as you can see in the example OUTPUT.

```
# find the neighbor we want some information about
:global neighbor [/ip neighbor get [find where address=192.168.91.173]]

# get the local interface we're connected via
:global neighborLocalInterface [/ip neighbor get $neighbor interface]

# check the result... using put, it looks like its a normal string
:put $neighborLocalInterface
# OUTPUT: ether5;bridge-lan

# check the type - it's actually an array
:put [:typeof $neighborLocalInterface]
```

```

# OUTPUT: array

# verify that through environment print
:environment print
# OUTPUT: str={"ether5"; "bridge-lan"}

# here's something strange that happens if you print without tostr
:put ("interface: $neighborLocalInterface")
# OUTPUT: interface: ether5;interface: bridge-lan

# here's what we were expecting to see, thanks to the use of :tostr
:put ("interface: $[:tosr $neighborLocalInterface]")
# OUTPUT: interface: ether5;bridge-lan

```

Notice the output of the `:put` on line 20 has some unexpected content.

## Playing with arrays

```

# MSHARP 20180827
# Playing with arrays
# Lesson learned: You cannot mix using keys and not using keys in the same array, even a multi-dimensional
one.

{
:local a1 {0}
:local count 0

:put "=====
:put "Testing the handling of a single dimmension array with no keys"
:set a1 {1;2}
:put "Putting the full array:"
:put $a1
:put "Putting array elements in a foreach loop:"
:set count 0
:foreach i in=$a1 do={
  :put $i
  :set count ($count + 1)
}
:put "count:$count"

```

```

:put "====="
:put "Putting array elements in a foreach loop using keys:"
:set count 0
:foreach k,v in=$a1 do={
  :put ($k . " - " . $v)
  :set count ($count + 1)
}
:put "count:$count"
:put "====="

:put "Testing the handling of a single dimension array with keys"
:set a1 {a=1;b=2}
:put "Putting the full array:"
:put $a1
:put "Putting array elements in a foreach loop without using keys:"
:set count 0
:foreach i in=$a1 do={
  :put $i
  :set count ($count + 1)
}
:put "count:$count"
:put "====="
:put "Putting array elements in a foreach loop using keys:"
:set count 0
:foreach k,v in=$a1 do={
  :put ($k . " - " . $v)
  :set count ($count + 1)
}
:put "count:$count"
:put "====="

:put "Testing the handling of a multi-dimensional array using no keys"
:set a1 {{1;2};{3;4}}
:put "Putting the full multi-dimensional array:"
:put $a1
:foreach i in=$a1 do={
  :put "Putting one of the sub-arrays:"
  :put $i
  :put "Putting the items in the array individually:"
  :set count 0

```

```

:foreach k in=$i do={
  :put $k
  :set count ($count + 1)
}
:put "count:$count"
:put "Putting the key - value pairs of the array"
:set count 0
:foreach k,v in=$i do={
  :put ($k . " - " . $v)
  :set count ($count + 1)
}
:put "count:$count"
:put "======"
}
:put "======"

:put "Testing the handling of a multi-dimensional array using keys"
:set a1 {{a=1;b=2}},{c=3;d=4}
:put "Putting the full multi-dimensional array:"
:put $a1
:foreach i in=$a1 do={
  :put "Putting one of the sub-arrays:"
  :put $i
  :put "Putting the items in the array individually:"
  :set count 0
  :foreach k in=$i do={
    :put $k
    :set count ($count + 1)
  }
  :put "count:$count"
  :put "Putting the key - value pairs of the array"
  :set count 0
  :foreach k,v in=$i do={
    :put ($k . " - " . $v)
    :set count ($count + 1)
  }
  :put "count:$count"
  :put "======"
}
:put "======"

```

```

:put "Testing the handling of a multi-dimensional array, one subarray not using keys and the using keys."
:put "Notice that mixing keys and no keys does not work properly."
:set a1 {{1;2};{c=3;d=4}}
:put "Putting the full multi-dimensional array:"
:put $a1
:foreach i in=$a1 do={
  :put "Putting one of the sub-arrays:"
  :put $i
  :put "Putting the items in the array individually:"
  :set count 0
  :foreach k in=$i do={
    :put $k
    :set count ($count + 1)
  }
  :put "count:$count"
  :put "Putting the key - value pairs of the array"
  :set count 0
  :foreach k,v in=$i do={
    :put ($k . " - " . $v)
    :set count ($count + 1)
  }
  :put "count:$count"
  :put "====="
}
:put "====="

#END
}

```

#end

Revision #4

Created 10 May 2023 17:09:28 by bluecrow76

Updated 7 June 2023 11:31:15 by bluecrow76