Scripting Tips and Tricks

Do not use console numbers to get parameter values

Lets start with very basics. When you work with console to access parameters, you are used to following syntax:

```
[admin@rack1_b34_CCR1036] /interface> print
Flags: D - dynamic, X - disabled, R - running, S - slave
 #     NAME                                TYPE       ACTUAL-MTU L2MTU  MAX-L2MTU
 0  R  ether1                              ether            1500  1580      1022
[admin@rack1_b34_CCR1036] /interface> set 0 name=LAN
```

What print command does is temporary saves buffer with id numbers referencing to internal ID numbers, so obviously if you are trying to use non-existent buffer values script will fail, like, for example this script:

```
/system script add name=script1 source={
  /ip route set 0 gateway=3.3.3.3
}
```

Script does not know what you assume by "1" and will throw an error. Proper way is to use internal ID numbers, those numbers can be seen if you are doing `print as-value` or returned by find command, for example:

```
[admin@rack1_b34_CCR1036] /ip route> :put [find where dst-address="10.0.0.0/8"]
*1
```

So in this case proper script would be:

```
/system script add name=script1 source={
  /ip route set *1 gateway=3.3.3.3
}
```

Note that it is not recommended to use internal numbers directly, since items can be removed and re-added in which case internal id number will change and script will fail again, so instead use find command directly in your code:

```
/system script add name=script1 source={
  /ip route set [find dst-address="0.0.0.0/0"] gateway=3.3.3.3
}
```

Why find does not work even if correct value is specified?

Lets say we want to print specific address:

```
[admin@rack1_b34_CCR1036] /ip address> print where address=111.111.1.1/24
Flags: X - disabled, I - invalid, D - dynamic
 #   ADDRESS            NETWORK         INTERFACE
```

So why it does not work?

Console tries to convert variable types as hard as it can, but it is not always possible to do it correctly, so lets look closely why this particular example does not work. First lets check what variable type "address" is:

```
[admin@rack1_b34_CCR1036] /ip address> :put [:typeof ([print as-value]->0->"address")]
str
```

So obviously we are comparing string to ip-prefix. And conversion from ip-prefix to string does not happen, so what we can do to solve the problem? Convert variable to correct format:

```
[admin@rack1_b34_CCR1036] /ip address> print where address=[:tostr 111.111.1.1/24]

Flags: X - disabled, I - invalid, D - dynamic
 #   ADDRESS            NETWORK         INTERFACE
 0   111.111.1.1/24     111.111.1.0     ether2
```

Or use string directly:

```
[admin@rack1_b34_CCR1036] /ip address> print where address="111.111.1.1/24"

Flags: X - disabled, I - invalid, D - dynamic
 #   ADDRESS            NETWORK         INTERFACE
 0   111.111.1.1/24     111.111.1.0     ether2
```

Obviously second method is not suitable if you are getting ip prefix from a variable, then conversion should be done as in first example or by writing variable to string with "$myVar".

How to define empty array

RouterOS does not allow to define empty array in a way that you think it should work:

```
[admin@1p_DUT_wAP ac] /interface> :global array {}
syntax error (line 1 column 17)
```

Insted a work around is to convert empty string to an array:

```
[admin@rack1_b36_CCR1009] > :global array [:toarray ""]
[admin@rack1_b36_CCR1009] > :environment print
array={}
```

From here we can use this array to set elements:

```
[admin@rack1_b36_CCR1009] > :set ($array->"el0") "el0_val"
[admin@rack1_b36_CCR1009] > :environment print
array={el0="el0_val"}
```

How to remove variables

You could use `/system script environment remove` to remove unused variables, however more preferred method is to unset variable.

Setting no value to existing parameter will unset it, see example below:

```
[admin@MikroTik] /system script environment> :global myVar 1
[admin@MikroTik] /system script environment> print
 # NAME               VALUE
 0 myVar              1
[admin@MikroTik] /system script environment> :set myVar
[admin@MikroTik] /system script environment> print
 # NAME               VALUE
[admin@MikroTik] /system script environment>
```

Get values for properties if 'get' command is not available

For example, how do you get usable output for scripting from `/interface wireless info hw-info` command? Use as-value:

```
[admin@1p_DUT_wAP ac] /interface wireless info> :put [hw-info wlan1 as-value ]
ranges=2312-2732/5/b;g;gn20;gn40;2484-2484/5/b;g;gn20;gn40;rx-chains=0;1;tx-chains=0;1
```

Output is 1D array so you can easily get interested property value

```
[admin@1p_DUT_wAP ac] /interface wireless info> :put ([hw-info wlan1 as-value ]->"tx-chains")
0;1
```

Always check what value and type command returns

Lets say we want to get gateway of specific route using as-value, if we execute following command it will return nothing

```
[admin@rack1_b36_CCR1009] /ip address> :put ([/ip route print as-value where gateway="ether1"]->"gateway")
```

Command assumes that output will be 1D array from which we could extract element gateway.

At first lets check if print is actually find anything:

```
[admin@rack1_b36_CCR1009] /ip address> :put ([/ip route print as-value where gateway="ether1"])
.id=*400ae12f;distance=255;dst-address=111.111.111.1/32;gateway=ether1;pref-src=111.111.111.1
```

So obviously there is something wrong with variable itself or variable type returned. Lets check it more closely:

```
[admin@rack1_b36_CCR1009] /ip address> :global aa ([/ip route print as-value where gateway="ether1"
])
[admin@rack1_b36_CCR1009] /ip address> :environment print
aa={{.id=*400ae12f; distance=255; dst-address=111.111.111.1/32; gateway={"ether1"}; pref-src=111.11
1.111.1}}
```

Now it is clear that returned value is 2D array with one element. So the right sequence to extract gateway will be:

- get 2d array
- get first element
- get "gateway" from picked element

```
[admin@rack1_b36_CCR1009] /ip address> :put ([:pick [/ip route print as-value where gateway="ether1"] 0]->"
gateway")
ether1
```

Be careful when adding array to string

If you want to print an array or add an array to existing string, be very careful as it may lead to unexpected results. For example ,we have array with two elements and we want to print the array value on screen:

```
[admin@1p_DUT_wAP ac] /> :global array {"cccc", "ddddd"}
[admin@1p_DUT_wAP ac] /> :put ("array value is: " . $array )
array value is: cccc;array value is: ddddd
```

Obviously this is not what we expected, because what . does is adds string to each array element and then prints the output. Instead you need to convert to string first:

```
[admin@1p_DUT_wAP ac] /> :put ("array value is: " . [:tostr  $array] )
array value is: cccc;ddddd
```

Get/Set unnamed elements in array

Lets say we have an array of elements { "el1"; "el2"; "el3" }. It is possible to pick elements of an array with pick command, but is not so neat as syntax below:

```
[admin@1p_DUT_wAP ac] /> :global test { "el1"; "el2"; "el3" }
[admin@1p_DUT_wAP ac] /> :put ($test->1)
el2
```

The same syntax can be used to set values:

```
[admin@1p_DUT_wAP ac] /> :set ($test->2) "el3_changed"
[admin@1p_DUT_wAP ac] /> :environment print
test={"el1"; "el2"; "el3_changed"}
```

Set element value in 2D array

Syntax used in example above can also be used to set element value in 2D array:

```
[admin@1p_DUT_wAP ac] /> :global test {{"11";"12";"13"};{"21";"22";"23"}}
[admin@1p_DUT_wAP ac] > :set ($test->1->1) "22_changed"
[admin@1p_DUT_wAP ac] > :put [($test->1->1)]
22_changed
[admin@1p_DUT_wAP ac] > :environment print
test={{"11"; "12"; "13"}; {"21"; "22_changed"; "23"}}
```

Read value of global variable defined in other script

Lets say we have one script that declares variable and sets the value:

```
/system script add name=script1 source={
  :global myVar "hello!"
}
```

And we want to write the value of that variable in log from another script. Simply adding `/log info $myVar` will fail to return correct value, because second script does not know anything about variables defined in another scripts. To make it work properly variable need to be defined, so correct second script code is:

```
/system script add name=script2 source={
  :global myVar;
  :log info "value is: $myvar"
}
```

Accessing global variable from function

Logically you would think that globally defined variables should be accessible in functions too, but that is not really the case. Lets see an example:

```
:global myVar "test"
:global myFunc do={
  :put "global var=$myVar"
}
[$myFunc]
```

Output is:

```
global var=
```

So obviously global variable is not accessible directly. To make it work we need do declare global variable inside the function:

```
:global myVar "test"
:global myFunc do={
  :global myVar;
  :put "global var=$myVar"
}
[$myFunc]
```

Output:

```
global var=test
```

Running function from another function

Same as above applies also to functions. If you want to run function from another function then it need to be declared.

```
:global test do={
  :return ($1 + 1)
}

:global testtest do={
  :local x 5
  :local y [$test $x]
  :put "typeof = $[:typeof $y]"
  :put "testets_res=$y"
}
```

Code above will not work as expected, output will be:

```
typeof = nil
testets_res=
```

To fix this we need to declare global "test" in "testtest" function

```
:global testtest do={
  :global test
  :local x 5
  :local y [$test $x]
  :put "typeof = $[:typeof $y]"
  :put "testets_res=$y"
}
```

Always use unique variable names

One of the most common scripting mistakes that most users are doing is not using unique varible names, for example, variable defined in function has the same name as globally defined variable, which leads to unexpected result:

```
:global my2 "123"

:global myFunc do={ :global my2; :put $my2; :set my2 "lala"; :put $my2 }
$myFunc my2=1234
:put "global value $my2"
```

Output will be:

```
1234
lala
global value 123
```

Another common case is when user defined variable have the same name as RouterOS built in variable, for example, we want to print route with dst address defined in variable:

```
[admin@1p_DUT_wAP ac] /ip route> :global "dst-address" "0.0.0.0/0"
[admin@1p_DUT_wAP ac] /ip route> print where dst-address=$"dst-address"
Flags: X - disabled, A - active, D - dynamic, C - connect, S - static, r - rip, b - bgp, o - ospf, m - mme,
B - blackhole, U - unreachable, P - prohibit
 #      DST-ADDRESS        PREF-SRC        GATEWAY            DISTANCE
 0 ADS  0.0.0.0/0                          10.155.136.1              1
 1 ADC  10.155.136.0/24    10.155.136.41   ether1                    0
```

Obviously result is not as expected, simple solution, use unique variable name:

```
[admin@1p_DUT_wAP ac] /ip route> :global myDst "0.0.0.0/0"
[admin@1p_DUT_wAP ac] /ip route> print where dst-address=$myDst
Flags: X - disabled, A - active, D - dynamic, C - connect, S - static, r - rip, b - bgp, o - ospf, m - mme,
B - blackhole, U - unreachable, P - prohibit
 #      DST-ADDRESS        PREF-SRC        GATEWAY            DISTANCE
 0 ADS  0.0.0.0/0                          10.155.136.1              1
```

Get values from looped interactive commands like "monitor"

Frequently asked question s how to get values in script returned by, for example, monitor command? First problem with such commands is that they are running infinitely until user action is applied, obviously you cannot do that from script. Instead you can run with additional parameter once, it will allow to execute command only once and stop. Another problem is getting variable value sin script, there is no as-value, there is no get, but they have do. What it does is allows to access variables returned by the command as in example below:

```
[admin@1p_DUT_wAP ac] /interface> monitor-traffic ether1 once do={:global myBps $"rx-bits-per-second" }
...
[admin@1p_DUT_wAP ac] /interface> :environment print
myBps=71464
```

Get file content received by fetch tool

Fetch tool allows for ease of use downloading file content into memory and allowing to access this data by script. To make it work use as-value parameter and output=user:

```
[admin@rack1_b34_CCR1036] > :put ([/tool fetch ftp://admin:@10.155.136.41/test.txt
 output=user as-value ]->"data")
my file content
```

Check script permissions

Lets say we have a script that creates and writes content to the file:

```
/system script add name=script1 policy=ftp,read,write source={
       /file print file=test;
       /file set test.txt content="my content"
}
```

Now lets add scheduler that will try to execute this script:

```
/system scheduler
add interval=10s name=test on-event=script2 policy=read,write
```

So now we wait 10 seconds, file not created, we wait another 10 seconds and still no file. What is going on? If you look closely script requires policy "ftp", to create a file, but scheduler has only "read" and "write" policies, so script will not be executed. Fix is to set scheduler to run with correct policies "read, write,ftp".

This applies also if you are trying to run script from netwatch, ppp on event and so on, which are limited to specific policies "read,write,test,reboot", so you will not be able to run advanced scripts that creates backups, creates files and so on.

Limitation could be fixed by using dont-require-permissions, but be very careful, read below.

Be careful when using dont-require-permissions

It is possible to set script with dont-require-permissions parameter. Basically it allows anyone without adequate permissions to execute the script. For example, if script has policies "read,write,test,sensitive", but user or application that executes the script has less, for example, "read,write", then by setting d`ont-require-permissions=yes` will allow to run script anyway.

This could potentially allow to change sensitive information using script even if user doe snot have enough permissions.