Scripting Host

Document revision 2.7 (Thu Sep 22 13:33:55 GMT 2005)
This document applies to V2.9

Table of Contents

General Information

Summary

This manual provides introduction to RouterOS built-in powerful scripting language.

Scripting host provides a way to automate some router maintenance tasks by means of executing user-defined scripts bounded to some event occurence. A script consists of configuration commands and expressions (ICE - internal console expression). The configuration commands are standard RouterOS commands, e.g. `/ip firewall filter add chain=forward protocol=gre action=drop` that are described in the relevant manuals, while expressions are prefixed with **:** and are accessible from all submenus.

The events used to trigger script execution include, but are not limited to the System Scheduler, the Traffic Monitoring Tool, and the Netwatch Tool generated events.

Specifications

Packages required: ***system***
License required: ***level1***
Home menu level: ***/system script***
Standards and Technologies: ***None***
Hardware usage: ***Not significant***

Related Documents

- Software Package Management
- System Scheduler
- Network Monitor
- Traffic Monitor
- Serial Port Monitor

Console Command Syntax

Description

Console commands are made of the following parts, listed in the order you type them in console:

- **prefix** - indicates whether the command is an ICE, like **:** in `:put` or that the command path starts from the root menu level, like / in

```
[admin@MikroTik] ip firewall mangle> /ping 10.0.0.1
```

- **path** - a relative path to the desired menu level, like **.. filter** in

```
[admin@MikroTik] ip firewall mangle> .. filter print
```

- **path_args** - this part is required to select some menu levels, where the actual path can vary across different user inputs, like **mylist** in

```
[admin@MikroTik] ip firewall mangle> /routeing prefix-list list mylist
```

- **action** - one of the actions available at the specified menu level, like **add** in

```
[admin@MikroTik] ip firewall mangle> /ip firewall filter add chain=forward action=drop
```

- **unnamed parameter** - these are required by some actions and should be entered in fixed order after the action name, like in **10.0.0.1** in

```
[admin@MikroTik] ip firewall mangle> /ping 10.0.0.1
```

- **name[=value]** - a sequence of parameter names followed by respective values, if required, like **ssid=myssid** in

```
/interface wireless set wlan1 ssid=myssid
```

Notes

Variable substitution, command substitution and expressions are allowed only for **path_args** and **unnamed parameter** values. **prefix**, **path**, **action** and **name[=value]** pairs can be given only directly, as a word. Therefore, `:put (1 + 2)` is valid and `:("pu" . "t") 3` is not.

Example

The parts of internal console commands are futher explained in the following examples:

```
/ping 10.0.0.1 count=5
```

prefix	/
action	**ping**
unnamed parameter	**10.0.0.1**
name[=value]	**count=5**

```
.. ip firewall rule input
```

path	**.. ip firewall rule**
path_args	**input**

```
:for i from=1 to=10 do={:put $i}
```

prefix	:
action	for
unnamed parameter	i
pname[=value]	from=1 to=10 do={:put $i}

```
/interface monitor-traffic ether1,ether2,ipip1
```

prefix	/
path	interface
action	monitor-traffic
unnamed parameter	ether1,ether2,ipip1

Expression Grouping

Description

This feature provides an easy way to execute commands from within one command level, by enclosing them in braces '{ }'.

Notes

Subsequent script commands are executed from the same menu level as the entire script. Consider the following example:

```
[admin@MikroTik] ip route> /user {
{... /ip route
{... print}
Flags: X - disabled
 #   NAME                                                GROUP ADDRESS
 0   ;;; system default user
     admin                                               full  0.0.0.0/0
 1   uuu                                                 full  0.0.0.0/0
[admin@MikroTik] ip route>
```

Although the current command level is changed to /**ip route**, it has no effect on next commands entered from prompt, therefore **print** command is still considered to be /**user print**.

Example

The example below demonstrates how to add two users to the **user** menu.

```
[admin@MikroTik] ip route> /user {
{... add name=x password=y group=write
{... add name=y password=z group=read
{... print}
Flags: X - disabled
 #   NAME                                                GROUP ADDRESS
 0   ;;; system default user
     admin                                               full  0.0.0.0/0
 1   x                                                   write 0.0.0.0/0
 2   y                                                   read  0.0.0.0/0
[admin@MikroTik] ip route>
```

Variables

Description

RouterOS scripting language suports two types of variables, which are global (system wide) and local (accessible only within the current script), respectively. A variable can be referenced by '$' (dollar) sign followed by the name of the variable with the exception of **set** and **unset** commands that take variable name without preceding dollar sign. Variable names should be composed of contain letters, digits and '-' character. A variable must be declared prior to using it in scripts. There are four types of declaration available:

- **global** - defined by global keyword, global variables can be accessed by all scripts and console logins on the same router. However, global variables are not kept across reboots.
- **local** - defined by local keyword, local variables are not shared with any other script, other instance of the same script or other console logins. The value of local variable value is lost when script finishes.
- **loop index variables** - defined within for and foreach statements, these variables are used only in do block of commands and are removed after command completes.
- **monitor variables** - some monitor commands that have do part can also introduce variables. You can obtain a list of available variables by placing :environment print statement inside the do block of commands.

You can assign a new value to variable using **set** action. It takes two unnamed parameters: the name of the variable and the new value of the variable. If a variable is no longer needed, it's name can be freed by **:unset** command. If you free local variable, it's value is lost. If you free global variable, it's value is still kept in router, it just becomes inaccessible from current script.

Notes

Loop variables "shadows" already introduced variables with the same name.

Example

```
[admin@MikroTik] ip route> /
[admin@MikroTik] > :global g1 "this is global variable"
[admin@MikroTik] > :put $g1
this is global variable
[admin@MikroTik] >
```

Command Substitution and Return Values

Description

Some console commands are most useful if their output can be feed to other commands as an argument value. In RouterOS console this is done by using the return values from commands. Return values are not displayed on the screen. To get the return value from a command, it should be enclosed in square brackets '[ ]'. Upon execution the return value of the the command will become the value of these brackets. This is

called command substitution.

The commands that produce return values are, but not limited to: **find**, which returns a reference to a particular item, **ping**, which returns the number of sucessful pings, **time**, which returns the measured time value, **incr** and **decr**, which return the new value of a variable, and **add**, which returns the internal number of newly created item.

Example

Consider the usage of **find** command:

```
[admin@MikroTik] > /interface
[admin@MikroTik] interface> find type=ether
[admin@MikroTik] interface>
[admin@MikroTik] interface> :put [find type=ether]
*1,*2
[admin@MikroTik] interface>
```

This way you can see internal console numbers of items. Naturally, you can use them as arguments in other commands:

```
[admin@MikroTik] interface> enable [find type=ether]
[admin@MikroTik] interface>
```

Operators

Description

RouterOS console can do simple calculations with numbers, time values, IP addresses, strings and lists. To get result from an expression with operators, enclose it in parentheses '(' and ')'. The expression result serves as a return value for the parentheses.

Command Description

- - unary minus. Inverts given number value.

- - binary minus. Substracts two numbers, two time values, two IP addresses or an IP address and a number

! - logical NOT. Unary operator, which inverts given boolean value

/ - division. Binary operator. Divides one number by another (gives number) or a time value by a number (gives time value).

. - concatenation. Binary operator, concatenates two string or append one list to another or appends an element to a list.

^ - bitwise XOR. The argumens and the result are both IP addresses

~ - bit inversion. Unary operator, which inverts bits in IP address

***** - multiplication. Binary operator, which can multiply two numbers or a time value by a number.

& - bitwise AND The argumens and the result are both IP addresses

&& - logical AND. Binary operator. The argumens and the result are both logical values

+ - binary plus. Adds two numbers, two time values or a number and an IP address.

< - less. Binary operator which compares two numbers, two time values or two IP addresses. Returns boolean value

<< - left shift. Binary operator, which shifts IP address by a given amount of bits. The first argument is an IP address, the second is an integer and the result is an IP address.

<= - less or equal. Binary operator which compares two numbers, two time values or two IP addresses. Returns boolean value

> - greater. Binary operator which compares two numbers, two time values or two IP addresses. Returns boolean value

>= - greater or equal. Binary operator which compares two numbers, two time values or two IP addresses. Returns boolean value

>> - right shift. Binary operator, which shifts IP address by a given amount of bits. The first argument is an IP address, the second is an integer and the result is an IP address.

| - bitwise OR. The argumens and the result are both IP addresses

|| - logical OR. Binary operator. The argumens and the result are both logical values

Notes

When comparing two arrays note, that two arrays are equal only if their respective elements are equal.

Example

Operator priority and evaluation order

```
[admin@MikroTik] ip firewall rule forward> :put (10+1-6*2=11-12=2+(-3)=-1)
false
[admin@MikroTik] ip firewall rule forward> :put (10+1-6*2=11-12=(2+(-3)=-1))
true
[admin@MikroTik] ip firewall rule forward
```

logical NOT

```
[admin@MikroTik] interface> :put (!true)
false
[admin@MikroTik] interface> :put (!(2>3))
true
[admin@MikroTik] interface>
```

unary minus

```
[admin@MikroTik] interface> :put (-1<0)
true
[admin@MikroTik] >
1
```

bit inversion

```
[admin@MikroTik] interface> :put (~255.255.0.0)
0.0.255.255
[admin@MikroTik] interface>
```

sum

```
[admin@MikroTik] interface> :put (3ms + 5s)
00:00:05.003
[admin@MikroTik] interface> :put (10.0.0.15 + 0.0.10.0)
cannot add ip address to ip address
```

```
[admin@MikroTik] interface> :put (10.0.0.15 + 10)
10.0.0.25
[admin@MikroTik] interface>
```

subtraction

```
[admin@MikroTik] interface> :put (15 - 10)
5
[admin@MikroTik] interface> :put (10.0.0.15 - 10.0.0.3)
12
[admin@MikroTik] interface> :put (10.0.0.15 - 12)
10.0.0.3
[admin@MikroTik] interface> :put (15h - 2s)
14:59:58
[admin@MikroTik] interface>
```

multiplication

```
[admin@MikroTik] interface> :put (12s * 4)
00:00:48
[admin@MikroTik] interface> :put (-5 * -2)
10
[admin@MikroTik] interface>
```

division

```
[admin@MikroTik] interface> :put (10s / 3)
00:00:03.333
[admin@MikroTik] interface> :put (5 / 2)
2
[admin@MikroTik] interface>
[admin@MikroTik] > :put (0:0.10 / 3)
00:00:02
[admin@MikroTik] >
```

comparison

```
[admin@MikroTik] interface> :put (10.0.2.3<=2.0.3.10)
false
[admin@MikroTik] interface> :put (100000s>27h)
true
[admin@MikroTik] interface> :put (60s,1d!=1m,3600s)
true
[admin@MikroTik] interface> :put (bridge=routing)
false
[admin@MikroTik] interface> :put (yes=false)
false
[admin@MikroTik] interface> :put (true=aye)
false
[admin@MikroTik] interface>
```

logical AND, logical OR

```
[admin@MikroTik] interface> :put ((yes && yes) || (yes && no))
true
[admin@MikroTik] interface> :put ((no || no) && (no || yes))
false
[admin@MikroTik] interface>
```

bitwise AND, bitwise OR, bitwise XOR

```
[admin@MikroTik] interface> :put (10.16.0.134 & ~255.255.255.0)
0.0.0.134
[admin@MikroTik] interface>
```

shift operators

```
[admin@MikroTik] interface> :put (~((0.0.0.1 << 7) - 1))
```

```
255.255.255.128
[admin@MikroTik] interface>
```

Concatenation

```
[admin@MikroTik] interface> :put (1 . 3)
13
[admin@MikroTik] interface> :put (1,2 . 3)
1,2,3
[admin@MikroTik] interface> :put (1 . 3,4)
13,4
[admin@MikroTik] interface> :put (1,2 . 3,4)
1,2,3,4
[admin@MikroTik] interface> :put ((1 . 3) + 1)
14
[admin@MikroTik] interface>
```

Data types

Description

The RouterOS console differentiates between several data types, which are string, boolean, number, time interval, IP address, internal number and list. The console tries to convert any value to the most specific type first, backing if it fails. The order in which the console attempts to convert an entered value is presented below:

- list
- internal number
- number
- IP address
- time
- boolean
- string

Internal scripting language supplies special functions to explicitly control type conversion. The **toarray**, **tobool**, **toid**, **toip**, **tonum**, **tostr** and **totime** functions convert a value accordingly to **list**, **boolean**, **internal number**, **IP address**, **number**, **string** or **time**.

The number type is internally represented as 64 bit signed integer, so the value a number type variable can take is in range from -9223372036854775808 to 9223372036854775807. It is possible to input number value in hexadecimal form, by prefixing it with **0x**, e.g.:

```
[admin@MikroTik] > :global MyVar 0x10
[admin@MikroTik] > :put $MyVar
16
[admin@MikroTik] >
```

Lists are treated as comma separated sequence of values. Putting whitespaces around commas is not recommended, because it might confuse console about words' boundaries.

Boolean values can be either **true** or **false**. Console also accepts **yes** for **true**, and **no** for **false**.

Internal numbers are preceided ***** sign.

Time intervals can be entered either using HH:MM:SS.MS notation, e.g.:

```
[admin@MikroTik] > :put 01:12:1.01
01:12:01.010
[admin@MikroTik] >
```

or as sequence of numbers, optionally followed by letters specifying the units of time measure (**d** dor days, **h** for hours, **m** for minutes, **s** for seconds and **ms** for milliseconds), e.g.:

```
[admin@MikroTik] > :put 2d11h12
2d11:00:12
[admin@MikroTik] >
```

As can bee seen, time values with omitted unit specificators are treated as expressed in seconds.

- **d, day, days** - one day, or 24 hours
- **h, hour, hours** - one hour
- **m, min** - one minute
- **s** - one second
- **ms** - one millisecond, id est 0.001 second

Possible aliases for time units:

The console also accepts time values with decimal point:

```
[admin@MikroTik] > :put 0.1day1.2s
02:24:01.200
[admin@MikroTik] >
```

Command Reference

Description

RouterOS has a number of built-in console commands and expressions (ICE) that do not depend on the current menu level. These commands do not change configuration directly, but they are useful for automating various maintenance tasks. The full ICE list can be accessed by typing '?' after the ':' prefix (therefore it can be safely assumed that all ICE have ':' prefix), for example:

```
[admin@MikroTik] > :
beep         execute  global  list     pick     time     toip    typeof
delay        find     if      local    put      toarray  tonum   while
do           for      led     log      resolve  tobool   tostr
environment  foreach  len     nothing  set      toid     totime
[admin@MikroTik] >
```

Command Description

beep - forces the built-in PC beeper to produce a signal for length seconds at frequency Hz. (*integer*; default: **1000**) - signal frequency measured in Hz (*time*; default: **100ms**) - signal length

```
[admin@MikroTik] > :beep length=2s frequency=10000

[admin@MikroTik] >
```

delay - does nothing for a given amount of time. (*time*) - amount of time to wait

- **omitted** - delay forever

do - executes commands repeatedly until given conditions are met. If no parameters are given, do just executes its payload once, which does not make much use. If a logical condition is specified for the while parameter, it will be evaluated after executing commands, and in case it is true, do statement is executed again and again until false. The if parameter, if present, is evaluated only once before doing anything else, and if it is false then no action is taken (*text*) - actions to execute repeatedly (yes | no) - condition, which is evaluated each time after the execution of enclosed statements (yes | no) - condition, which is evaluated once before the execution of enclosed statements

```
[admin@MikroTik] > {:global i 10; :do {:put $i; :set i ($i - 1);} \
\... while (($i < 11) && ($i > 0)); :unset i;}
10
9
8
7
6
5
4
3
2
1
[admin@MikroTik] >
```

environment print - prints information about variables that are currently initialised. All global variables in the system are listed under the heading Global Variables. All variables that are introduced in the current script (variables introduced by :local or created by :for or :foreach statements) are listed under the heading Local Variables.

Creating variables and displaying a list of them

```
[admin@MikroTik] > :local A "This is a local variable"
[admin@MikroTik] > :global B "This is a global one"
[admin@MikroTik] > :environment print
Global Variables
B=This is a global one
Local Variables
A=This is a local variable
[admin@MikroTik] >
```

find - searches for substring inside a string or for an element with particular value inside an array, depending on argument types and returns position at which the value is found. The elements in list and characters in string are numbered from 0 upwards (*text* |) - the string or value list the search will be performed in (*text*) - value to be searched for (*integer*) - position after which the search is started

```
[admin@MikroTik] interface pppoe-server> :put [:find "13sdf1sdfss1sfsdf324333"  ]
0
[admin@MikroTik] interface pppoe-server> :put [:find "13sdf1sdfss1sfsdf324333" 3  ]
1
[admin@MikroTik] interface pppoe-server> :put [:find "13sdf1sdfss1sfsdf324333" 3  3]
17
[admin@MikroTik] interface pppoe-server> :put [:find "1,1,1,2,3,3,4,5,6,7,8,9,0,1,2,3"
3 ]
4
[admin@MikroTik] interface pppoe-server> :put [:find "1,1,1,2,3,3,4,5,6,7,8,9,0,1,2,3"
3 3]
4
[admin@MikroTik] interface pppoe-server> :put [:find "1,1,1,2,3,3,4,5,6,7,8,9,0,1,2,3"
3 4]
5
[admin@MikroTik] interface pppoe-server> :put [:find "1,1,1,2,3,3,4,5,6,7,8,9,0,1,2,3"
3 5]
15
[admin@MikroTik]
```

for - executes supplied commands over a given number of iterations, which is explicity set through from and to parameters (*name*) - the name of the loop counter variable (*integer*) - start value of the loop counter variable (*integer*) - end value of the loop counter variable (*integer*; default: **1**) - increment value. Depending on the loop counter variable start and end values, step parameter can be treated also as decrement (*text*) - contains the command to be executed repeatedly

```
[admin@MikroTik] > :for i from=1 to=100 step=37  do={:put ($i . " - " . 1000/$i)}
1 - 1000
38 - 26
75 - 13
[admin@MikroTik] >
```

foreach - executes supplied commands for each element in list (*name*) - the name of the loop counter variable () - list of values over which to iterate (*text*) - contains the command to be executed repeatedly

Printing a list of available interfaces with their respective IP addresses

```
:foreach i in=[/interface find type=ether ] \
\... do={:put ("+--" . [/interface get $i name]); \
\... :foreach j in=[/ip address find interface=$i]
\... do={:put ("|  `--" . [/ip address get $j address])}}
+--ether1
|  `--1.1.1.3/24
|  `--192.168.50.1/24
|  `--10.0.0.2/24
+--ether2
|  `--10.10.0.2/24
[admin@MikroTik] >
```

global - declares global variable (*name*) - name of the variable (*text*) - value, which should be assigned to the variable

```
[admin@MikroTik] > :global MyString "This is a string"
[admin@MikroTik] > :global IPAddr 10.0.0.1
[admin@MikroTik] > :global time   0:10
[admin@MikroTik] > :environment print
Global Variables
IPAddr=10.0.0.1
time=00:10:00
MyString=This is a string
Local Variables
[admin@MikroTik] >
```

if - conditional statement. If a given logical condition evaluates to true then the do block of commands is executed. Otherwice an optional else block is executed. (yes | no) - logical condition, which is evaluated once before the execution of enclosed statements (*text*) - this block of commands is executed if the logical condition evaluates to true (*text*) - this block of commands is executed if the logical condition evaluates to false

Check if the firewall has any rules added

```
[admin@MikroTik] > :if ([:len [/ip firewall filter find]] > 0) do={:put true}
else={:put false}
true
[admin@MikroTik] >
```

Check whether the gateway is reachable. In this example, the IP address of the gateway is **10.0.0.254**

```
[admin@MikroTik] > :if ([/ping 10.0.0.254 count=1] = 0) do {:put "gateway unreachable"}
10.0.0.254 ping timeout
1 packets transmitted, 0 packets received, 100% packet loss
gateway unreachable
[admin@MikroTik] >
```

led - allows to control the LEDs (Light Emitting Diodes) of the RouterBOARD 200 series

embedded boards. This command is available only on RouterBoard 200 platform with the routerboard package installed (yes | no) - controls first LED (yes | no) - controls second LED (yes | no) - controls third LED (yes | no) - controls fourth LED (*time*) - specifies the length of the action

- **omitted** - altar LED state forever

Switch on LEDs 2 and 3 for 5 seconds

```
[admin@MikroTik] > :led led2=yes led3=yes length=5s
```

len - returns the number of characters in string or the number of elements in list depending on the type of the argument (*name*) - string or list the length of which should be returned

```
[admin@MikroTik] > :put [:len gvejimezyfopmekun]
17
[admin@MikroTik] > :put [:len gve,jim,ezy,fop,mek,un]
6
[admin@MikroTik] >
```

list - displays a list of all available console commands that match given search key(s) (*text*) - first search key (*text*) - second search key (*text*) - third search key

Display console commands that have **hotspot**, **add** and **user** parts in the command's name and path

```
[admin@MikroTik] > :list user hotspot "add "
List of console commands under "/" matching "user" and "hotspot" and "add ":

ip hotspot profile add name= hotspot-address= dns-name= \
\... html-directory= rate-limit= http-proxy= smtp-server= \
\... login-by= http-cookie-lifetime= ssl-certificate= split-user-domain= \
\... use-radius= radius-accounting= radius-interim-update= copy-from=
ip hotspot user add server= name= password= address= mac-address= \
\... profile= routes= limit-uptime= limit-bytes-in= limit-bytes-out= \
\... copy-from= comment= disabled=
ip hotspot user profile add name= address-pool= session-timeout= \
\... idle-timeout= keepalive-timeout= status-autorefresh= \
\... shared-users= rate-limit= incoming-filter= outgoing-filter= \
\... incoming-mark= outgoing-mark= open-status-page= on-login= on-logout= copy-from=
[admin@MikroTik] >
```

local - declares local variable (*name*) - name of the variable (*text*) - value, which should be assigned to the variable

```
[admin@MikroTik] > :local MyString "This is a string"
[admin@MikroTik] > :local IPAddr 10.0.0.1
[admin@MikroTik] > :local time   0:10
[admin@MikroTik] > :environment print
Global Variables
Local Variables
IPAddr=10.0.0.1
time=00:10:00
MyString=This is a string
[admin@MikroTik] >
```

log - adds a message specified by message parameter to the system logs. (*name*) - name of the logging facility to send message to (*text*) - the text of the message to be logged

Send message to **info** log

```
[admin@MikroTik] > :log info "Very Good thing happened. We have received our first
packet!"
[admin@MikroTik] > /log print follow
...
19:57:46 script,info Very Good thing happened. We have received our first packet!
...
```

nothing - has no action, and returns value of type "nothing". In conditions nothing behaves as "false"

Pick a symbol that does not exist from a string

```
[admin@MikroTik] > :local string qwerty
[admin@MikroTik] > :if ([:pick $string 10]=[:nothing]) do={
{... :put "pick and nothing commands return the same value"}
pick and nothing commands return the same value
[admin@MikroTik] >
```

pick - returns a range of elements or a substring depending on the type of input value (*text* |) - the string or value list from which a substring or a subrange should be returned (*integer*) - start position of substring or subrange (*integer*) - end position for substring or subrange

```
[admin@MikroTik] > :set a 1,2,3,4,5,6,7,8
[admin@MikroTik] > :put [:len $a]
8
[admin@MikroTik] > :put [:pick $a]
1
[admin@MikroTik] > :put [:pick $a 0 4]
1,2,3,4
[admin@MikroTik] > :put [:pick $a 2 4]
3,4
[admin@MikroTik] > :put [:pick $a 2]
3
[admin@MikroTik] > :put [:pick $a 5 1000000]
6,7,8
[admin@MikroTik] > :set a abcdefghij
[admin@MikroTik] > :put [:len $a]
10
[admin@MikroTik] > :put [:pick $a]
a
[admin@MikroTik] > :put [:pick $a 0 4]
abcd
[admin@MikroTik] > :put [:pick $a 2 4]
cd
[admin@MikroTik] > :put [:pick $a 2]
c
[admin@MikroTik] > :put [:pick $a 5 1000000]
fghij
```

put - echoes supplied argument to the console (*text*) - the text to be echoed to the console

Display the MTU of **ether1** interface

```
[admin@MikroTik] > :put [/interface get ether1 mtu]
1500
[admin@MikroTik] >
```

resolve - returns IP address of the host resolved from the DNS name. The DNS settings should be configured on the router (/ip dns submenu) prior to using this command. (*text*) - domain name to be resolved into an IP address

DNS configuration and **resolve** command example

```
[admin@MikroTik] ip route> /ip dns set primary-dns=159.148.60.2
[admin@MikroTik] ip route> :put [:resolve www.example.com]
192.0.34.166
```

set - assigns new value to a variable (*name*) - the name of the variable (*text*) - the new value of the variable

Measuring time needed to resolve www.example.com

```
[admin@MikroTik] > :put [:time [:resolve www.example.com ]]
00:00:00.006
[admin@MikroTik] >
```

time - measures the amount of time needed to execute given console commands (*text*) - the console

commands to measure execution time of

Measuring time needed to resolve www.example.com

```
[admin@MikroTik] > :put [:time [:resolve www.example.com ]]
00:00:00.006
[admin@MikroTik] >
```

while - executes given console commands repeatedly while the logical conditions is true (yes | no) - condition, which is evaluated each time before the execution of enclosed statements (*text*) - console commands that should be executed repeatedly

```
[admin@MikroTik] > :set i 0; :while ($i < 10) do={:put $i; :set i ($i + 1)};
0
1
2
3
4
5
6
7
8
9
[admin@MikroTik] >
```

Special Commands

Description

Monitor

It is possible to access values that are shown by most **monitor** actions from scripts. A **monitor** command that has a **do** parameter can be supplied either script name (see /**system scripts**), or console commands to execute.

Get

Most **print** commands produce values that are accessible from scripts. Such **print** commands have a corresponding **get** command on the same menu level. The **get** command accepts one parameter when working with regular values or two parameters when working with lists.

Notes

Monitor command with **do** argument can also be called directly from scripts. It will not print anything then, just execute the given script.

The names of the properties that can be accessed by **get** are the same as shown by **print** command, plus names of item flags (like the disabled in the example below). You can use [T ab] key completions to see what properties any particular **get** action can return.

Example

In the example below **monitor** action will execute given script each time it prints stats on the screen, and it will assign all printed values to local variables with the same name:

```
[admin@MikroTik] interface> monitor-traffic ether2 once do={:environment print}
    received-packets-per-second: 0
       received-bits-per-second: 0bps
        sent-packets-per-second: 0
           sent-bits-per-second: 0bps

Global Variables
i=1
Local Variables
sent-bits-per-second=0
received-packets-per-second=0
received-bits-per-second=0
sent-packets-per-second=0
[admin@MikroTik] interface>
```

Additional Features

Description

To include comment in the console script prefix it with '#'. In a line of script that starts with '#' all characters until the newline character are ignored.

To put multiple commands on a single line separate them with ';'. Console treats ';' as the end of line in scripts.

Any of the **{}[]"'\$** characters should be escaped in a regular string with '\' character. Console takes any character following '\' literally, without assigning any special meaning to it, except for such cases:

```
\a     bell (alarm), character code 7
\b     backspace, character code 8
\f     form feed, character code 12
\n     newline, character code 10
\r     carriage return, character code 13
\t     tabulation, character code 9
\v     vertical tabulation, character code 11
\_     space, character code 32
```

Note that '\', followed by any amount of whitespace characters (spaces, newlines, carriage returns, tabulations), followed by newline is treated as a single whitespace, except inside quotes, where it is treated as nothing. This is used by console to break up long lines in scripts generated by export commands.

Script Repository

Home menu level: ***/system script***

Description

All scripts are stored in the **/system script** menu along with some service information such as script name, script owner, number of times the script was executed and permissions for particular script.

In RouterOS, a script may be automatically started in three different ways:

- via the scheduler
- on event occurence - for example, the netwatch tool generates an event if a network host it is configured to monitor becomes unaccessible
- by another script

It is also possible to start a script manually via /**system script run** command.

Property Description

last-started (*time*) - date and time when the script has been last invoked. The argument is shown only if the run-count!=0.

owner (*name*; default: **admin**) - the name of the user who created the script

policy (*multiple choice: ftp | local | policy | read | reboot | ssh | telnet | test | web | write*; default: **reboot,read,write,policy,test**) - the list of the policies applicable:

- **ftp** - user can log on remotely via ftp and send and retrieve files from the router
- **local** - user can log on locally via console
- **policy** - manage user policies, add and remove user
- **read** - user can retrieve the configuration
- **reboot** - user can reboot the router
- **ssh** - user can log on remotely via secure shell
- **telnet** - user can log on remotely via telnet
- **test** - user can run ping, traceroute, bandwidth test
- **web** - user can log on remotely via http
- **write** - user can retrieve and change the configuration

run-count (*integer*; default: **0**) - script usage counter. This counter is incremented each time the script is executed. The counter will reset after reboot.

source (*text*; default: **""**) - the script source code itself

Command Description

run (*name*) - executes a given script (*name*) - the name of the script to execute

Notes

You cannot do more in scripts than you are allowed to do by your current user rights, that is, you cannot use disabled policies. For example, if there is a policy group in /**user group** which allows you **ssh,local,telnet,read,write,policy,test,web** and this group is assigned to your user name, then you cannot make a script that reboots the router.

Example

The following example is a script for writing message "Hello World!" to the **info** log:

```
[admin@MikroTik] system script> add name="log-test" source={:log info "Hello World!"}
[admin@MikroTik] system script> run log-test

[admin@MikroTik] system script> print
 0 name="log-test" owner="admin"
policy=ftp,reboot,read,write,policy,test,winbox,password last-started=mar/20/2001
22:51:41
   run-count=1 source=:log info "Hello World!"
[admin@MikroTik] system script>
```

Task Management

Home menu level: ***/system script job***

Description

This facility is used to manage the active or scheduled tasks.

Property Description

name (*read-only: name*) - the name of the script to be referenced when invoking it
owner (*text*) - the name of the user who created the script
source (*read-only: text*) - the script source code itself

Example

```
[admin@MikroTik] system script> job print
  # SCRIPT  OWNER                   STARTED
  0 DelayeD admin                   dec/27/2003 11:17:33

[admin@MikroTik] system script>
```

You can cancel execution of a script by removing it from the job list

```
[admin@MikroTik] system script> job remove 0
[admin@MikroTik] system script> job print

[admin@MikroTik] system script>
```

Script Editor

Command name: ***/system script edit***

Description

RouterOS console has a simple full-screen editor for scripts with support for multiline script writing.

Keyboard Shortcuts

- **Delete** - deletes character at cursor position
- **Ctrl+h, backspase** - deletes character before cursor. Unindents line
- **Tab** - indents line
- **Ctrl+b, LeftArrow** - moves cursor left
- **Ctrl+f, RightArrow** - moves cursor right
- **Ctrl+p, UpArrow** - moves cursor up
- **Ctrl+n, DownArrow** - moves cursor down
- **Ctrl+a, Home** - moves cursor to the beginning of line or script
- **Ctrl+e, End** - moves cursor to the end of line or script

- **Ctrl+y** - inserts contents of buffer at cursor position
- **Ctrl+k** - deletes characters from cursor position to the end of line
- **Ctrl+u** - undoes last action
- **Ctrl+o** - exits editor accepting changes
- **Ctrl+x** - exits editor discarding changes

Command Description

edit (*name*) - opens the script specified by the name argument in full-screen editor

Notes

All characters that are deleted by **backspace, delete** or **Ctrl+k** keys are accumulated in the buffer. Pressing any other key finishes adding to this buffer (**Ctrl+y** can paste it's contents), and the next delete operation will replace it's contents. Undo doesn't change contents of cut buffer.

Script editor works only on VT102 compatible terminals (terminal names "vt102", "linux", "xterm", "rxvt" are recognized as VT102 at the moment). Delete, backspace and cursor keys might not work with all terminal programs, use 'Ctrl' alternatives in such cases.

Example

The following example shows the script editor window with a sample script open:

This script is used for writing message "hello" and 3 messages "kuku" to the system log.