API

Summary

Application Programmable Interface (API) allows users to create custom software solutions to communicate with RouterOS to gather information, adjust the configuration, and manage the router. API closely follows syntax from the command-line interface (CLI). It can be used to create translated or custom configuration tools to aid ease of use in running and managing routers with RouterOS.

API service must be enabled before trying to establish the API connection. By default, API uses TCP:*8728* and TCP:*8729* (secure).

API-SSL service is capable of working in two modes - with and without a certificate. In the case no certificate is used in */ip service* settings then an anonymous Diffie-Hellman cipher has to be used to establish a connection. If a certificate is in use, a TLS session can be established.

Protocol

Communication with the router is done by sending sentences and receiving one or more sentences in return. A sentence is a sequence of words terminated by zero-length words. Word is part of a sentence encoded in a certain way - encoded length and data. Communication happens by sending sentences to the router and receiving replies to sent sentences. Each sentence sent to the router using API should contain a command as a first word followed by words in no particular order, the end of the sentence is marked by a zero-length word. When the router receives a full sentence (command word, no or more attribute words, and zero-length word) it is evaluated and executed, then a reply is formed and returned.

API words

Words are part of a sentence. Each word has to be encoded in a certain way - the length of the word followed by the word content. The length of the word should be given as a count of bytes that are going to be sent.

The length of the word is encoded as follows:

Value of length	# of bytes	Encoding
0 <= len <= 0x7F	1	len, lowest byte
0x80 <= len <= 0x3FFF	2	len | 0x8000, two lower bytes
0x4000 <= len <= 0x1FFFFF	3	len | 0xC00000, three lower bytes

0x200000 <= len <= 0xFFFFFFF	4	len | 0xE0000000
len >= 0x10000000	5	0xF0 and len as four bytes

- Each word is encoded as length, followed by that many bytes of content;
- Words are grouped into sentences. The end of a sentence is terminated by a zero-length word;
- The scheme allows encoding of length up to **0x7FFFFFFFFF**, only four-byte length is supported;
- **len** bytes are sent most significant first (network order);
- If the first byte of the word is **>= 0xF8**, then it is a reserved control byte. After receiving an unknown control byte API client cannot proceed, because it does not know how to interpret the following bytes;
- Currently, control bytes are not used;

In general, *words* can be described like this **<<***encoded word length***><***word content***>>**. *Word content* can be separated into 5 parts: *command word*, *attribute word*, *API attribute word*. *query word,* and *reply word*

Command word

The first word in the sentence has to be a command followed by attribute words and a zero-length word or terminating word. The name of the command word should begin with '/'. Names of commands closely follow CLI, with spaces replaced with '/'. Some commands are specific to API;

Command word structure in the strict order:

- encoded length
- content prefix /
- CLI converted command

API-specific commands:

```
login
cancel
```

Command word content examples:

```
/login
```

```
/user/active/listen
```

```
/interface/vlan/remove
```

```
/system/reboot
```

Attribute word

Each *command word* has its list of *attribute words* depending on content.

Attribute word structure consists of 5 parts in this order:

- encoded length
- content prefix equals sign - **=**
- attribute name
- separating equals sign - **=**
- value of an attribute if there is one. It is possible that the attribute does not have a value

⚠ Value can hold multiple *equal* signs in the value of an *attribute word* since the way the word is encoded.

⚠ Value can be empty.

Examples without encoded length prefix:

```
=address=10.0.0.1
```

```
=name=iu=c3Eeg
```

```
=disable-running-check=yes
```

> Order of attribute words and API parameters is not important and should not be relied on

API attribute word

API attribute word structure is in the strict order:

- encoded length
- content prefix with the dot.
- attribute name
- name postfixed with equals =sign
- attribute value

Currently, the only such API attribute is the *tag*.

> If the sentence contains an *API attribute word* tag then each returned sentence in reply from the router to that tagged sentence will be tagged with the same tag.

Query word

Sentences can have additional query parameters that restrict their scope. A detailed explanation is in the query section.

Example of a sentence using query word attributes:

```
/interface/print
?type=ether
?type=vlan
?#|!
```

- Query words begin with '?'.
- Currently, only the *print* command handles query words.

> The order of query words is significant

Reply word

It is only sent by the router in response to the full sentence received from the client.

- The first word of reply begins with '!';
- Each sentence sent generates at least one reply (if a connection does not get terminated);
- The last reply for every sentence is the reply that has the first word !done;
- Errors and exceptional conditions begin with !trap;
- Data replies begin with !re
- If the API connection is closed, RouterOS sends !fatal with a reason as a reply and then closes the connection;

API sentences

API sentence is the main object of communication using API.

- Empty sentences are ignored.
- A sentence is processed after receiving zero length word.
- There is a limit on the number and size of sentences that the client can send before it has logged in.
- Order of attribute words should not be relied on. As order and count are changeable by *.proplist* attribute.
- The sentence structure is as follows:
  - The first word should contain a *command word*;
  - Should contain *zero-length word* to terminate the sentence;
  - Can contain none or several *attribute words*. There is no particular order in what attribute words have to be sent in the sentence, order is not important for *attribute words*;

- Can contain none or several *query words*. The order of *query words* in the sentence is important.

> ⚠ *Zero-length word* terminates the sentence. If it is not provided router will not start to evaluate sent words and will consider all the input as part of the same sentence.

Initial login

Note: that each command and response ends with an empty word.

Login method post-v6.43:

/login
=name=admin
=password=
!done

- Now the client sends a username and password in the first message.
- Password is sent in plain text.
- in case of an error, the reply contains =message=*error message*.
- In case of a successful login, the client can start to issue commands.

Tags

- It is possible to run several commands simultaneously, without waiting for the previous one to complete. If the API client is doing this and needs to differentiate command responses, it can use the 'tag' API parameter in command sentences.
- If you include the 'tag' parameter with a non-empty value in the command sentence, then the 'tag' parameter with the same value will be included in all responses generated by this command.
- If you do not include the 'tag' parameter or its value is empty, then all responses for this command will not have a 'tag' parameter.

Command description

- /cancel
  - optional argument: =tag=*tag of command to cancel*, without it, cancels all running commands
  - does not cancel itself
  - all canceled commands are interrupted and in the usual case generate '!trap' and '!done' responses
  - please note that /cancel is separate command and can have its own unique '.tag' parameter, that is not related to '=tag' argument of this command

- listen
  - listen command is available where console print command is available, but it does not have the expected effect everywhere (i.e. may not work)
  - "!re" sentences are generated as something changes in a particular item list
  - when an item is deleted or disappears in any other way, the '!re' sentence includes the value '=.dead=yes'
  - This command does not terminate. To terminate it, use /cancel command.

- getall
  - getall command is available where console print command is available (getall is an alias for print).
  - replies contain =.id=*Item internal number* property.

- print
  - API print command differs from the console counterpart in the following ways:
    - where an argument is not supported. Items can be filtered using query words (see below).
    - .proplist argument is a comma-separated list of property names that should be included for the returned items.
      - returned items may have additional properties.
      - order of returned properties is not defined.
      - if a list contains duplicate entries, handling of such entries is not defined.

- if a property is present in ".proplist", but absent from the item, then that item does not have this property value (?name will evaluate to false for that item).
- if ".proplist" is absent, all properties are included as requested by the print command, even those that have slow access time (such as file contents and performance counters). Thus the use of .proplist is encouraged. The omission of .proplist may have a high-performance penalty if the "=detail=" argument is set.

Queries

The print command accepts query words that limit the set of returned sentences.

- Query words begin with '?'.
- The order of query words is significant. A query is evaluated starting from the first word.
- A query is evaluated for each item in the list. If the query succeeds, the item is processed, if a query fails, the item is ignored.
- A query is evaluated using a stack of boolean values. Initially, the stack contains an infinite amount of 'true' values. At the end of the evaluation, if the stack contains at least one 'false' value, the query fails.
- Query words operate according to the following rules:

Query	Description
?name	pushes 'true' if an item has a value of property *name*, 'false' if it does not.
?-name	pushes 'true' if an item does not have a value of property *name*, 'false' otherwise.
?*name*=*x* **?=*name*=*x***	pushes 'true' if the property *name* has a value equal to *x*, 'false' otherwise.
?<name=*x*	pushes 'true' if the property *name* has a value less than *x*, 'false' otherwise.
?>name=*x*	pushes 'true' if the property *name* has a value greater than *x*, 'false' otherwise.
?#*operations*	applies operations to the values in the stack. • operation string is evaluated from left to right. • the sequence of decimal digits followed by any other character or end of the word is interpreted as a stack index. top value has an index 0. • an index that is followed by a character pushes a copy of the value at that index. • an index that is followed by the end of the word replaces all values with the value at that index. • **!** character replaces the top value with the opposite. • **&** pops two values and pushes the result of logical 'and' operation. • **|** pops two values and pushes the result of logical 'or' operation. • **.** after an index does nothing. • **.** after another character pushes a copy of the top value.

Regular expressions are not supported in API, so do not try to send a query with the **~** symbol

Examples:

- Get all ethernet and VLAN interfaces:

```
/interface/print
?type=ether
?type=vlan
?#|
```

- Get all routes that have a non-empty comment:

```
/ip/route/print
?>comment=
```

- Forum thread with a detailed explanation of the use of queries

OID

The print command can return OID values for properties that are available in SNMP.

In the console, OID values can be seen by running the 'print oid' command. In API, these properties have a name that ends with ".oid", and can be retrieved by adding their name to the value of '.proplist'. An example:

/system/resource/print
=.proplist=uptime,cpu-load,uptime.oid,cpu-load.oid
!re
=uptime=01:22:53
=cpu-load=0
=uptime.oid=.1.3.6.1.2.1.1.3.0
=cpu-load.oid=.1.3.6.1.2.1.25.3.3.1.2.1
!done

!trap

When for some reason API sentence fails trap is sent in return accompanied by a **message** attribute and on some occasions **category** argument.

message

When an API sentence fails, some generic message or message from the used internal process is returned to give more details about the failure

```
<<< /ip/address/add
<<< =address=192.168.88.1
<<< =interface=asdf <<<

>>> !trap
>>> =category=1
>>> =message=input does not match any value of interface
```

category

if it is a general error, it is categorized and the error category is returned. possible values for this attribute are

- 0 - missing item or command
- 1 - argument value failure
- 2 - execution of command interrupted
- 3 - scripting related failure
- 4 - a general failure
- 5 - API related failure
- 6 - TTY related failure
- 7 - value generated with :return command

Command examples

/system/package/getall

/system/package/getall
!re
=.id=*5802
=disabled=no
=name=routeros-x86
=version=3.0beta2
=build-time=oct/18/2006 16:24:41
=scheduled=
!re
=.id=*5805
=disabled=no
=name=system
=version=3.0beta2
=build-time=oct/18/2006 17:20:46
=scheduled=
... more !re sentences ...
!re
=.id=*5902
=disabled=no
=name=advanced-tools
=version=3.0beta2
=build-time=oct/18/2006 17:20:49
=scheduled=
!done

/user/active/listen

/user/active/listen
!re
=.id=*68
=radius=no

=when=oct/24/2006 08:40:42
=name=admin
=address=0.0.0.0
=via=console
!re
=.id=*68
=.dead=yes
... more !re sentences ...

/cancel, simultaneous commands

/login
!done
=ret=856780b7411eefd3abadee2058c149a3
/login
=name=admin
=response=005062f7a5ef124d34675bf3e81f56c556
!done
-- first start listening for interface changes (tag is 2)
/interface/listen
.tag=2
-- disable interface (tag is 3)
/interface/set
=disabled=yes
=.id=ether1
.tag=3
-- this is done for disable command (tag 3)
!done
.tag=3
-- enable interface (tag is 4)
/interface/set
=disabled=no

=.id=ether1
.tag=4
-- this update is generated by a change made by the first set command (tag 3)
!re
=.id=*1
=disabled=yes
=dynamic=no
=running=no
=name=ether1
=mtu=1500
=type=ether
.tag=2
-- this is done for enable command (tag 4)
!done
.tag=4
-- get interface list (tag is 5)
/interface/getall
.tag=5
-- this update is generated by a change made by the second set command (tag 4)
!re
=.id=*1
=disabled=no
=dynamic=no
=running=yes
=name=ether1
=mtu=1500
=type=ether
.tag=2
-- these are replies to getall command (tag 5)
!re
=.id=*1
=disabled=no

=dynamic=no
=running=yes
=name=ether1
=mtu=1500
=type=ether
.tag=5
!re
=.id=*2
=disabled=no
=dynamic=no
=running=yes
=name=ether2
=mtu=1500
=type=ether
.tag=5
-- here interface getall ends (tag 5)
!done
.tag=5
-- stop listening - request to cancel command with tag 2, cancel itself uses tag 7
/cancel
=tag=2
.tag=7
-- listen command is interrupted (tag 2)
!trap
=category=2
=message=interrupted
.tag=2
-- cancel command is finished (tag 7)
!done
.tag=7

-- listen command is finished (tag 2)
!done
.tag=2

Example client

A simple API client in Python3

Example output:

```
debian@localhost:~/api-test$ ./api.py 10.0.0.1 admin ''
<<< /login
<<<
>>> !done
>>> =ret=93b438ec9b80057c06dd9fe67d56aa9a
>>>
<<< /login
<<< =name=admin
<<< =response=00e134102a9d330dd7b1849fedfea3cb57
<<<
>>> !done
>>>
/user/getall

<<< /user/getall
<<<
>>> !re
>>> =.id=*1
>>> =disabled=no
>>> =name=admin
>>> =group=full
>>> =address=0.0.0.0/0
>>> =netmask=0.0.0.0
>>>
>>> !done
>>>
```

See also

API examples

API implementations in different languages, provided by different sources. They are not ordered in any particular order.

- in Python3 by MikroTik
- in .NET (C#) high-level API solution forum thread additional info by danikf
- in PHP by boen_robot
- in C by Håkon Nessjøen
- in Java by Gideon LeGrange
- in Erlang by Valery Comtihon
- in GO by André Luiz dos Santos
- in Python3 by Arturs Laizans
- in C++17 by Ayman Al-Qadhi