

NECONFc ssclient Client Library for Python

Reference Guide

November 11, 2021

SegueSoft Inc. Web site: <http://www.seguesoft.com>

The materials contained in this document are copyrighted by SegueSoft Inc.
Unauthorized use of this information will result in immediate forfeiture
of all license rights and liability for damages from such use.

Table of Contents

| | |
|--|----|
| Why sscient..... | 4 |
| Sample Scripts..... | 4 |
| How to run..... | 4 |
| Command Reference..... | 5 |
| Session Object Creation Command..... | 5 |
| create_session..... | 5 |
| create_session_tls..... | 6 |
| Session Object Methods..... | 8 |
| build_subtree_filter..... | 8 |
| cancel_commit..... | 9 |
| close_session..... | 9 |
| commit..... | 9 |
| copy_config..... | 10 |
| confirmed_commit..... | 10 |
| create_subscription..... | 11 |
| delete_config..... | 12 |
| discard_changes..... | 12 |
| edit_config..... | 13 |
| extract_rpc_error_from_rpc_reply..... | 16 |
| extract_module_qualified_node_id_and_value_from_rpc_reply..... | 17 |
| extract_prefixed_node_id_and_value_from_rpc_reply..... | 17 |
| extract_value_from_rpc_reply..... | 18 |
| sget..... | 19 |
| sget_config..... | 20 |
| get_notification_stream_names..... | 21 |
| get_schema..... | 22 |
| kill_session..... | 22 |
| listen_callhome..... | 23 |
| lock..... | 23 |
| partial_lock..... | 23 |
| partial_unlock..... | 24 |
| print_rpc_stdout..... | 24 |
| send_any_action..... | 25 |
| send_any_rpc..... | 25 |
| send_rpc_raw_xml..... | 26 |
| set_indent_flag..... | 26 |
| set_yang_path..... | 27 |
| unlock..... | 27 |
| validate_config..... | 28 |
| xget..... | 28 |
| xget_config..... | 29 |
| High-level query, retrieval and edit commands..... | 29 |
| yang_lookup..... | 29 |
| hget..... | 31 |
| hget_return_all..... | 32 |
| hget_config..... | 32 |
| hget_config_return_all..... | 33 |

| | |
|--|----|
| create merge replace delete remove..... | 33 |
| Utility commands..... | 35 |
| extract_notification_name_timestamp..... | 35 |
| wait_notification..... | 35 |
| RESTCONF API..... | 36 |
| create_session_restconf..... | 36 |
| rget, head, options..... | 37 |
| post_data, put, post_actions, post_operations..... | 38 |
| delete..... | 39 |
| patch..... | 39 |

Why ssclient

ssclient is a Python library that aims to simplify client-side scripting and application development when working with NETCONF and YANG.

With ssclient, you don't need to manually work with XML, the library hides that and handles it automatically for you. For example:

- 1) When you want to retrieve config data, you do not need to manually create XML subtree filter string, the library will build the filter expression for you.
- 2) When you want to send any user-defined RPC, simply providing all required input arguments, and the library will automatically build the RPC XML and send it.
- 3) All XML reply can be turned in a list of (nodeID, value) tuple for easy handling. You can also easily extract specific node value, rpc-error info from a XML reply message

Sample Scripts

A complete set of sample scripts demonstrating how to use Seguesoft's NETCONF library package can be found in the 'ssclient/sample-scripts' folder under the library's top installation directory.

The most effective way to learn how to use this library is through studying and understanding these sample scripts.

How to run

SegueSoft's Netconf client library requires Python 3.8 and above

- 1) Set up PYTHONPATH

For example, on Linux you should do:

```
export PYTHONPATH=/path-to/netconfc_install_dir/ssclient:
```

On Windows control panel

```
set PYTHONPATH to C:\SegueSoft\NETCONFc\ssclient
```

- 1). How to use sample scripts?

Install Python3.8

```
pip install brotli configobj lxml PySocks paramiko pyang  
pycryptodome regex rfc3986 requests h2 Pypubsub
```

Then you can run these sample scripts by typing:

```
cd /path-to/segue-netconfc-demo/ssclient-sample-scripts/sample-scripts  
python get_and_get_config.py
```

Command Reference

Session Object Creation Command

create_session

| | |
|------------|---|
| Syntax | <pre>create_session(host="localhost", port=830, client_sock=None, user="Unspecified", password="Unspecified", private_keyfile=None, timeout=30, keep_alive=0, check_unknown_host=True, capabilities = None, custom_capabilities =None; socks_proxy={})</pre> <p>This method sends a <create-session> RPC to create a NETCONF over SSH session. The created session object is then used to send other NETCONF RPC commands.</p> <p>To create a NETCONF over TLS session, see <code>create_session_tls</code>.</p> |
| Parameters | <p>host: mandatory IP address or hostname of the Netconf sever to connect to</p> <p>port: optional port number of the Netconf sever to connect to. If not specified, default to standard Netconf over SSH port 830</p> <p>client_sock: optional the call-home socket obtained by calling <code>listen_callhome()</code></p> <p>user: mandatory user name of the Netconf sever</p> <p>password: mandatory if using SSH password authentication The password for the 'user' or the password for the <code>private_keyfile</code> specified</p> <p>private_keyfile: mandatory if using SSH public key authentication The private key file with full path for the 'user'. The private key file assumes openSSH PEM format key file</p> <p>timeout: optional timeout period for connecting to the remote Netconf sever. If not specified,</p> |

| | |
|---------------|--|
| | <p>default 30 seconds.</p> <p>keep_alive: optional seconds to wait before sending a keepalive packet (or 0 to disable keepalives).</p> <p>Turn on/off keep-alive packets (default is off). If this is set, after the given seconds without sending any data over the connection, a "keep-alive" packet will be sent (and ignored by the remote host). This can be useful to keep connections alive over a NAT.</p> <p>check_unknown_host If this option is set to False, server host key will not be checked</p> <p>If this option is set to True (default), an unknown server host key will need users to confirm as shown below:</p> <p>Unknown host: a7:04:df:a0:78:82:1c:f1:0f:ed:da:54:b3:fa:7a:bf Are you sure you want to connect? [Y/n]</p> <p>capabilities: optional Capabilities are space or newline separated strings which are added to the built-in capability statements</p> <p>custom_capabilities: optional Custom_capabilities are also space or newline separated strings but they are used to replace the built-in capabilities</p> <p>socks_proxy If the server needs to be connected via a SOCKS4 or SOCKS5 proxy, provide required proxy settings in a dictionary: socks_proxy={"server":", "port":1080, "type":"SOCKS5", "user": ", "password":""}</p> |
| Returns | A Session object for the session created. The returned object is then used to send other Netconf RPC commands. |
| Sample script | get.py |
| See also | close_session, kill_session |

create_session_tls

| | |
|--------|--|
| Syntax | <pre>create_session_tls(host="localhost", port=6513, client_sock=None, client_cert = "", client_key = "", trusted_certs="" timeout=30,</pre> |
|--------|--|

| | |
|------------|--|
| | <pre> keep_alive=0, check_unknown_host=True, capabilities = None, custom_capabilities =None; socks_proxy={}) </pre> <p>This method sends a <create-session> RPC to create a NETCONF over TLS session. The created session object is then used to send other NETCONF RPC commands.</p> |
| Parameters | <p>host: mandatory IP address or hostname of the NETCONF sever to connect to</p> <p>port: optional port number of the NETCONF sever to connect to. If not specified, default to standard NETCONF over TLS port 6513</p> <p>client_sock: optional the call-home socket obtained by calling listen_callhome()</p> <p>client_cert: mandatory Client certificate file with full path. The client certificate must be in PEM format and may contain client's private key.</p> <p>client_key: mandatory Client private key file with full path. This parameter can be left blank if the client certificate file already contains client's private key .The client key file must be in PEM format.</p> <p>trusted_certs: mandatory Trusted CA certificate file with full path in PEM format. This is needed to validate the server's certificate and can be a chained certificate file.</p> <p>timeout: optional timeout period for connecting to the remote Netconf sever. If not specified, default 30 seconds.</p> <p>keep_alive: optional seconds to wait before sending a keepalive packet (or 0 to disable keepalives).</p> <p>Turn on/off keep-alive packets (default is off). If this is set, after the given seconds without sending any data over the connection, a "keep-alive" packet will be sent (and ignored by the remote host). This can be useful to keep connections alive over a NAT.</p> <p>check_unknown_host If this option is set to False, server host name in the certificate will not be checked. This uses <code>ssl.match_hostname(cert, hostname)</code> so you must install Python2.7.9 in order to use it.</p> |

| | |
|---------------|--|
| | <p>If this option is set to True (default), an unknown server hostname will need users to confirm as shown below:</p> <p>Unknown host, are you sure you want to connect? [Y/n]</p> <p>capabilities: optional Capabilities are space or newline separated strings which are added to the built-in capability statements</p> <p>custom_capabilities: optional Custom_capabilities are also space or newline separated strings but they are used to replace the built-in capabilities</p> <p>socks_proxy If the server needs to be connected via a SOCKS4 or SOCKS5 proxy, provide required proxy settings in a dictionary: socks_proxy={"server":", "port":1080, "type":"SOCKS5", "user": ", "password":"} }</p> |
| Returns | A Session object for the session created. The returned object is then used to send other Netconf RPC commands. |
| Sample script | create_session_tls.py |
| See also | close_session, kill_session |

Session Object Methods

build_subtree_filter

| | |
|------------|---|
| Syntax | <p>build_subtree_filter(*args, **args)</p> <p>Create a subtree filter expression to be used with <get>, <get-config> or <creates-subscription> commands.</p> |
| Parameters | <p>*args: nodeID1, nodeID2,</p> <p>A nodeID can be either a YANG absolute-schema-nodeid, or a tuple of (absolute-schema-nodeid, contentValuesToMatch).</p> <p>Example: "/Netconf-state/datastores/datastore" or ("/Netconf-state/datastores/datastore", "running")</p> |

| | |
|---------------|--|
| | <p>kwargs:</p> <p>namespaces = prefix/namespace mapping dict used in nodeIDs</p> <p>If qualified name is used, you must also use namespaces argument</p> |
| Returns | A Netconf subtree filter expression |
| Sample script | create_subscription.py under the installation root directory |

cancel_commit

| | |
|---------------|--|
| Syntax | <p>cancel_commit(persistID=None)</p> <p>This method sends a <cancel-commit> RPC to cancel an ongoing confirmed commit.</p> |
| Parameters | <p>persist-ID: optional</p> <p>Only used when canceling a confirmed commit from a different session.</p> <p>The persistID is the 'persist' string token set in the preceding confirmed commit request.</p> |
| Returns | A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem) |
| Sample script | confirmed_commit.py |
| See also | commit, confirmed_commit |

close_session

| | |
|---------------|--|
| Syntax | <p>close_session()</p> <p>This method sends a <close-session> RPC.</p> |
| Returns | A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem) |
| Sample script | sample-script/confirmed_commit.py |
| See also | create_session, kill_session |

commit

| | |
|------------|---|
| Syntax | <p>commit(persistID=None)</p> <p>This method sends a <commit> RPC to commit the candidate configuration into the device's running configuration</p> |
| Parameters | persist-ID: optional |

| | |
|---------------|---|
| | <p>Only used when issuing a conforming commit for a confirmed commit.</p> <p>The persistID is the 'persist' string token set in the preceding confirmed commit request.</p> |
| Returns | A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem) |
| Sample script | commit.py |
| See also | confirmed_commit |

copy_config

| | |
|---------------|--|
| Syntax | <p>copy_config(source, target, withDefaults=None)</p> <p>This method sends a <copy-config> RPC.</p> |
| Parameters | <p>source: one of 'candidate' 'running' 'startup' url</p> <p>target: one of 'candidate' 'running' 'startup' url</p> <p>withDefaults: optional, one of 'explicit' 'report-all' 'trim' 'report-all-tagged'.</p> <p>The <copy-config> operation is only affected by the <with-defaults> parameter if the target of the operation is specified with the <url> parameter</p> <p>For example, the current running config can be saved by using a file url as the target: "Target: file://localhost/checkpoint.conf, or "Target: file:///checkpoint.conf, # note it has three '/' since host name is # omitted, "Target: ftp://user:pass@host/config</p> |
| Returns | A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem) |
| Sample script | copy_config.py |

confirmed_commit

| | |
|--------|---|
| Syntax | <p>confirmed_commit(timeout=600, persistID=None)</p> <p>This method sends a Netconf confirmed <commit> RPC. A follow-up confirming <commit> must be issued within the timeout period to complete the ongoing commit process.</p> <p>The confirming commit is a simple commit operation that you can send by calling commit(persistID=None) command.</p> |
|--------|---|

| | |
|---------------|--|
| Parameters | <p>timeout: optional Timeout period for confirmed commit, in seconds. If unspecified, the confirming timeout defaults to 600 seconds</p> <p>persist-ID: optional Make the confirmed commit survive a session termination, and set a token on the ongoing confirmed commit. This value is a string token that will be used by a follow-up commit or a confirming commit from any session.</p> |
| Returns | A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem) |
| Sample script | confirmed_commit.py |
| See also | commit, cancel_commit |

create_subscription

| | |
|------------|---|
| Syntax | <pre>create_subscription(stream="default", start_time=None, stop_time=None, subtree_filter=None, xpath_filter=None, callback=None):</pre> <p>This method sends a <create_subscription> RPC to subscribe to notification streams.</p> |
| Parameters | <p>steam: optional An optional parameter, <stream>, that indicates which stream of events is of interest. If not present, events in the default Netconf stream will be sent</p> <p>start_time: optional If used, the input format mus be "YYYY-MM-DD HH:MM:SS". For example, "2014-04-01 12:00:00" This parameter is used to trigger the replay feature and indicate that the replay should start at the time specified.</p> <p>stop_time: optional If used, the input format mus be "YYYY-MM-DD HH:MM:SS". For example, "2014-04-01 12:00:00"</p> |

| | |
|---------------|---|
| | <p>This parameter is used to indicate the newest notifications of interest. Must be used with and be later than <startTime>.</p> <p>subtree_filter: optional This parameter is the standard subtree filter as defined in Netconf RFC6241. If used it indicates which subset of all possible events is of interest. You can only use either subtree_filter or xpath_filter, not both.</p> <p>You can use the method 'build_subtree_filter' to build subtree filter expression automatically.</p> <p>xpath_filter: optional This parameter is the standard xpath filter as defined in Netconf RFC6241. If used it indicates which subset of all possible events is of interest. You can only use either subtree_filter or xpath_filter, not both.</p> |
| Returns | A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem) |
| Sample script | notification_subscription.py |
| See also | wait_notification |

delete_config

| | |
|---------------|---|
| Syntax | <p>delete_config(target)</p> <p>This method sends a <delete-config> RPC to delete a configuration datastore.</p> <p>Factory reset: when deleting <startup> config store, server will reset the device to its factory defaults.</p> <p>Note the <running> and <candidate> configuration datastore cannot be deleted.</p> |
| Parameters | <p>target: 'startup' url</p> <p>Examples for url: file://localhost/checkpoint.conf, or if host name is omitted, file:///checkpoint.conf, #note it has three '/' ftp://user:pass@host/config</p> |
| Returns | A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem) |
| Sample script | delete_config.py under the installation root directory |
| See also | discard_changes |

discard_changes

| | |
|--------|-------------------|
| Syntax | discard_changes() |
|--------|-------------------|

| | |
|---------------|--|
| | <p>This method sends a Netconf <discard-changes> RPC. This command actually reset candidate datastore to be the same as current running datastore.</p> <p>If the candidate configuration is not to be committed, the <discard-changes> operation can be used to revert the candidate configuration to the current running configuration.</p> |
| Returns | A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem) |
| Sample script | discard_changes.py under the installation root directory |
| See also | delete_config |

edit_config

| | |
|------------|--|
| Syntax | <p>edit_config(*args, **kwargs)</p> <p>This method sends an <edit-config> RPC. Use this method to create new leafs, list, merge with the existing config data, or replace existing config data.</p> <p>Its typically usage scenarios are demonstrated in the following samples. Please be sure to study related sample scripts.</p> |
| Parameters | <p>*args: Form 1: nodeID1, value1_or_ValueDict1, nodeID2, value2_or_valueDict2, ...or Form 2: (nodeID1, value1_or_ValueDict1), (nodeID2, value2_or_valueDict2), nodeID is YANG absolute schema node identifier as defined in RFC6020.</p> <p>The nodeID should be passed down in the order they should appear in the actual RPC message.</p> <p>The nodeID should be qualified with prefix. If no prefix is used then the RPC generated will not contain namespace attribute, and it is expected that a Netconf server will search all it's known namespaces for a match. To speed things up, qualified names are recommended.</p> <p>The value_or_valueDict can be either a simple value or a dictionary containing operation attribute values. If you want to assign XML attributes such as 'operation' to a node, pass in a valueDict, otherwise just use a simple value.</p> <p>The valueDict is a dict that may contain one of more of the following keys:</p> <pre> 'value' : value to set the targeted node to 'operation' : merge replace create delete remove 'insert_operation': first last before after, 'target_key' : List instance-identifier predicates 'target_value' : Leaf-list value, 'with_default' : True False 'new_list_entry' : "" 'raw_XML_data': : set to True of the value is XML data (for </pre> |

encoding anyxml or anydata)

The '**new_list_entry**' is used to indicate if a new list entry should be created when editing multiple list entries. It is only used if the list node itself is not specified. In other words it must be specified in the first key node's valueDict if this is the case . See samples/edit_list.py for a concrete example.

For example, you can create two list entries using:

```
(/nacm:nacm/nacm:rule-list, {'value': ''})
(/nacm:nacm/nacm:rule-list/nacm:name, {'value': 'almighty'})
(/nacm:nacm/nacm:rule-list/nacm:rule, {'value': ''})
(/nacm:nacm/nacm:rule-list/nacm:rule/nacm:name, {'value': 'almighty1'})
(/nacm:nacm/nacm:rule-list/nacm:rule/nacm:action, {'value': 'permit'})
(/nacm:nacm/nacm:rule-list/nacm:rule, {'value': ''})
(/nacm:nacm/nacm:rule-list/nacm:rule/nacm:name, {'value': 'almighty2'})
(/nacm:nacm/nacm:rule-list/nacm:rule/nacm:action, {'value': 'permit'})
```

or

```
(/nacm:nacm/nacm:rule-list/nacm:name, {'value': 'almighty'})
(/nacm:nacm/nacm:rule-list/nacm:rule/nacm:name, {'new_list_entry': True, 'value': 'almight1'})
(/nacm:nacm/nacm:rule-list/nacm:rule/nacm:action, {'value': 'permit'})
(/nacm:nacm/nacm:rule-list/nacm:rule/nacm:name, {'new_list_entry': True, 'value': 'almighty2'})
(/nacm:nacm/nacm:rule-list/nacm:rule/nacm:action, {'value': 'permit'})
```

The purpose of each dictionary key element above is explained below.

The '**value**' is the value that you want to modify for the node identified by the nodeID, For container node, list entry node, etc. it is an empty string.

The '**operation**' is one of standard Netconf edit-config operations:

'merge' | 'replace' | 'create' | 'delete' | 'remove'

For details of each operation means see RFC6241 <edit-config>

The '**insert_operation**' is only applicable for YANG list or leaf-list that is defined as "ordered-by-user". It can be one of the following values: '

'first' | 'last' | 'before' | 'after'

When using '**insert_operation**', you must also pass in '**target_key**' for a YANG list, or '**target_value**' for leaf-list. They are used for creating yang:key or yang:value attribute (RFC6020 7.8.6)

For a '**ordered-by-user**' list, you must include "**target_key**" in the valueDict. The value of '**target_key**' is a list of (keyname, value) tuple:

```
[ (KeyName, 'fred'), ]
[ (ip, '192.0.2.1'), (port, '80')],
```

or a already built YANG instance-identifier predicates. Example:

```
[ex:name='fred']
[ex:ip='192.0.2.1'][ex:port='80']
```

For the complete syntax of predicates, see RFC6020: 9.13.

For an 'ordered-by-user' leaf-list, you must include "target_value" element in the valueDict.

The 'with_default' element in valueDict indicates whether to set 'default:true' attribute on the targeted node. When "with_default: True" is included in the valueDict, the value of the "value" key must be the schema default value of the node identified by nodeID. As defined in RFC6243, section 4.5.2.

Note for "delete" and "remove" operation, "with_default" does not apply.

****kwargs:**

url=urlValue

The config is given in the url. If this option is specified, the arguments nodeID1, nodeID2 as well as the 'config=' option will be ignored.

the <url> element can be used as an alternative to the <config> parameter. The file that the url refers to contains the configuration data hierarchy to be modified, encoded in XML under the element <config> in the "urn:ietf:params:xml:ns:Netconf:base:1.0" namespace.

config=configData

If you already have manually constructed config xml data, use this option.

When this option is used, the arguments nodeID1, nodeID2 etc. will be ignored.

It must be a well formatted xml config data hierarchy to be modified, such as:

For example:

```
<config>
  <interfaces xmlns="http://myorg.org/ns/ymy-interfaces">
    <interface>
      <name>eth0</name>
    </interface>
  </interfaces>
</config>
```

target = 'candidate' | 'running' | 'startup'

Default: 'candidate'

default_operations = "merge" | "replace" | "none"

The default value for the <default-operation> parameter is "merge".

The <default-operation> parameter is optional.

test_operation = "test-then-set" | "set" | "test-only"

error_option = "rollback-on-error" | "stop-on-error" | "continue-on-error"

namespaces = prefix/namespace mapping dict used in nodeIDs

| | |
|----------------|---|
| Returns | A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem) |
| Sample scripts | <p><code>edit_leaf_and_leaflist.py</code> How to create/merge/delete/remove leafs, and insert value into ordered-by-user leaflist</p> <p><code>edit_list.py</code> How to create/merge/delete/remove list entries, and insert entry into ordered-by-user list</p> <p><code>edit_url</code> How to send edit-config if the config data hierarchy to be modified is contained in a url.</p> <p><code>edit_xml.py</code> How to send edit-config if the config data hierarchy to be modified is contained in a XML string.</p> |
| See also | <code>discard_changes</code> , <code>commit</code> , <code>confirmed_commit</code> |

extract_rpc_error_from_rpc_reply

| | |
|------------|--|
| Syntax | <p><code>extract_rpc_error_from_rpc_reply (replyXML_or_lxmlElem, sequence=0)</code></p> <p>This method is used to extract error code from a <code>rpc_reply</code> that contains <code>rpc_error</code></p> |
| Parameters | <p><code>replyXML_or_lxmlElem</code>: RPC reply XML string or lxml element</p> <p><code>sequence</code>: optional specify the which <code>rpc_error</code> (0...n) to retrieve if there are multiple <code>rpc_error</code> elements</p> |
| Returns | <p>Return: A dictionary contains the following key:</p> <ul style="list-style-type: none"> <code>error_type</code> <code>error_tag</code> <code>error_severity</code> <code>error_app_tag</code> <code>error_path</code> <code>error_message</code> <code>bad_attribute</code> <code>bad_element</code> <code>bad_namespace</code> <code>session_id</code> <code>ok_element</code> <code>err_element</code> |

| | |
|---------------|---|
| | <code>noop_element</code> |
| Sample script | <code>extract_rpc_error_from_msg.py</code> |
| See also | <code>extract_notification_name_timestamp</code> , <code>extract_value_from_rpc_reply</code> , <code>extract_prefixed_node_id_and_value_from_rpc_reply</code> |

extract_module_qualified_node_id_and_value_from_rpc_reply

| | |
|---------------|---|
| Syntax | <code>extract_module_qualified_node_id_and_value_from_rpc_reply(replyXML_or_elem)</code> Use this method to convert XML output of a RPC REPLY into a list of (modulename qualified node ID, value) tuples for easy handling. A nodeID uses module named prefixed node names. Each sub-ID uses the module name in its closest ancestor . |
| Parameters | <code>replyXML_or_elem</code> : this can be a reply XML string, or a reply lxml element |
| Returns | A tuple of (nodeID_List, nsmap) |
| Sample script | Converting data retrieved via <get> and <get-config>: |
| See also | <code>extract_prefixed_node_id_and_value_from_rpc_reply</code> , <code>extract_notification_name_timestamp</code> , <code>extract_value_from_rpc_reply</code> , |

extract_prefixed_node_id_and_value_from_rpc_reply

| | |
|------------|---|
| Syntax | <code>extract_prefixed_node_id_and_value_from_rpc_reply(replyXML_or_elem)</code> Use this method to convert XML output of a RPC REPLY into a list of (nodeID, value) tuples for easy handling. A nodeID uses prefixed node names. |
| Parameters | <code>replyXML_or_elem</code> : this can be a reply XML string, or a reply lxml element |
| Returns | A tuple of (nodeID_List, nsmap) |

| | |
|---------------|--|
| Sample script | Converting data retrieved via <get> and <get-config>: |
| See also | extract_module_qualified_node_id_and_value_from_rpc_reply, extract_notification_name_timestamp, extract_value_from_rpc_reply, |

extract_value_from_rpc_reply

| | |
|---------------|--|
| Syntax | <pre>extract_value_from_rpc_reply(replyXML_or_lxmlElem, nodeID_with_prefix, namespaces={})</pre> <p>This method is used to extract value form a reply message xml string. Use it to process reply to <get>, <get-config> or to get the value of “output” parameters in user-defined RPC.</p> |
| Parameters | <p>replyXML_or_lxmlElem: a RPC reply XML string, or a RPC reply lxml element</p> <p>nodeID_in_xml_reply Qualified names must be used to specify nodeID.</p> <ul style="list-style-type: none"> * If it is the reply to <get> or <get-config> that contains a data element, this parameter contains the nodeID under the top 'data' element, for example: /ncm:netconf-state/ncm:schemas/ncm:schema/ncm:identifier See an example in get.py * Otherwise, if the reply contains a rpc-error, this parameter contains the nodeID starting from 'rpc-error'. Example: /nc:rpc-error/nc:error-info/ncx:error-number See an example in extract_rpc_error_from_msg.py * If the reply is the output of any other RPC, this parameter contains the relative nodeID under 'output' element. Using RFC6022 <get-schema> RPC as an example, the nodeID_with_prefix to retrieve lock-id in the reply is: /ncm:data See an example in sample-script/get_schema.py <p>namespaces={}: prefix/namespace mapping dict used in nodeIDs</p> |
| Returns | The exacted value. If there are multiple values (e.g. a leaf in a list) then they will be separated by space. |
| Sample script | This example shows how to extract the lock-id returned by <partial-lock> and then use it in the subsequent <partial-unlock>: |

| | |
|----------|---|
| | lock_unlock_partial_lock_unlock.py Other examples: get.py get_config.py sample-script/get_schema.py |
| See also | extract_rpc_error_from_rpc_reply, extract_prefixed_node_id_and_value_from_rpc_reply |

sget

| | |
|------------|--|
| Syntax | sget(*args, **kwargs) Standard NETCONF <get> command for retrieving data using subtree filtering. |
| Parameters | <p>*args: nodeID_1, nodeID_2,</p> <p>Each nodeID can be a nodeID, or a tuple/ of (nodeID, valueForContentMatch). Each nodeID is an absolute schema node identifier starting with “/”, or a YANG instance ID.</p> <p>Example: "/ncm:netconf-state/ncm:datastores/ncm:datastore" ("/ncm:netconf-state/ncm:datastores/ncm:datastore", "running") "/ncm:netconf-state/schemas/schema/identifier" "/ncm:netconf-state/schemas/schema[identifier='netopeer cfgnetopeer'] [version='2013-02-14'] [format='yang']/location"</p> <p>kwargs: The keyword argument can be one of the following:</p> <p>filter=filterExpression If you already have a manually constructed subtree filter expression, use this option. When this option is used, the arguments arguments nodeID_1, nodeID_2 etc. will be ignored.</p> <p>It must be an empty string (empty filter) or a well formatted subtree filter expression string such as:</p> <pre><if:interfaces xmlns=http://Netconfcentral.org/ns/yuma-interfaces> <if:interface> <if:name /> </if:interface> </if:interfaces></pre> |

| | |
|---------------|--|
| | <p>withDefaults= 'explicit' 'report-all' 'trim' 'report-all-tagged'</p> <p>See RFC6243 for details on how to use this keyword argument.</p> <p>namespaces = prefix/namespace mapping dict used in nodeIDs.</p> |
| Returns | <p>A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem)</p> <p>If the call returns no data then it is considered a “False”</p> |
| Sample script | restconf_example.py |
| See also | get_config, xget, xget_config |

sget_config

| | |
|------------|--|
| Syntax | <p>sget_config(*args, **kwargs)</p> <p>Standard NETCONF <get-config> command for retrieving config data using subtree filtering</p> |
| Parameters | <p>*args:</p> <p>nodeID1, nodeID2,</p> <p>nodeID can either be an absolute schema nodeID, or a tuple of (absolute schema nodeID, contentValueToMatch), or a YANG instance ID.</p> <p>For example:</p> <pre>"/ncm:netconf-state/ncm:datastores/ncm:datastore" ("/ncm:etconf-state/ncm:datastores/ncm:datastore", "running") "/ncm:netconf-state/schemas/schema/identifier"</pre> <pre>"/ncm:netconf-state/schemas/schema[identifier='netopeer cfgnetopeer'] [version='2013-02-14'] [format='yang']/location"</pre> <p>kwargs: The keyword argument can be one of the following:</p> <p>filter=filterExpression If you already have a manually constructed subtree filter expression, use this option.</p> <p>When this option is used, the arguments nodeID1, nodeID2 etc. will be ignored.</p> |

| | |
|---------------|---|
| | <p>It must be an empty string (empty filter) or a well formatted filter expression string such as:</p> <pre><if:interfaces xmlns=http://Netconfcentral.org/ns/yuma-interfaces> <if:interface> <if:name /> </if:interface> </if:interfaces></pre> <p>source = one of 'candidate' 'running' 'startup' if unspecified default to 'running'</p> <p>withDefaults= 'explicit' 'report-all' 'trim' 'report-all-tagged' See RFC6243 for details.</p> <p>namespaces = prefix/namespace mapping dict used in nodeIDs.</p> |
| Returns | <p>A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem)</p> <p>If the call returns no data then it is considered a “False”</p> |
| Sample script | get_config.py |
| See also | get, xget, get, xget_config |

get_notification_stream_names

| | |
|---------------|--|
| Syntax | <pre>get_notification_stream_names()</pre> <p>This method retrieve all available notification stream names on the server</p> |
| Returns | A string of notification stream names separated by space |
| Example | See create_subscription.py |
| Sample script | create_subscription.py |
| See also | create_subscription |

get_schema

| | |
|---------------|--|
| Syntax | <pre>get_schema(identifier="", version="", schemaFormat="")</pre> <p>This method sends a Netconf <get-schema> request.</p> |
| Parameters | <p>identifier: Identifier string for the schema list entry.</p> <p>version: Optional Version of the schema requested. If this parameter is not present, and more than one version of the schema exists on the server, a 'data-not-unique' error is returned, as described above.</p> <p>schemaFormat: Optional "yang" "yin" "xsd" "rng" "rnc"</p> <p>The data modeling language of the schema. If this parameter is not present, and more than one formats of the schema exists on the server, a 'data-not-unique' error is returned, as described above.</p> |
| Returns | A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem) |
| Sample script | ssclient-sample-scripts/samples-scripts/get_schema.py under the installation root directory |
| See also | create_subscription |

kill_session

| | |
|------------|--|
| Syntax | <pre>kill_session(session-id)</pre> <p>This method sends a <kill-session> RPC.</p> |
| Parameters | <p>Session-id: The ID of the session to kill</p> |
| Returns | A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem) |
| See also | create_session, kill_session |

listen_callhome

| | |
|------------|---|
| Syntax | listen_callhome(port, mutableSocketList, stopEvent=None, timeout=360): This method creates a TCP server listening on the giving port. When receiving a call-home connection request, set the created client-side socket in the mutableSocketList [client_socket, exceptionErr] |
| Parameters | port: the server listening port (SSH uses 4334, TLS uses 4335) mutableSocketList: an empty list to receive the created socket and error tuple stopEvent: an optional Python threading event object. When set, the listner will exit. Timeout: optional, the default time is 360 seconds to stop listening. Example: client_socket, error = listen_callhome(4334) |
| Returns | A tuple of (client_socket, error). The returned client_socket can then be directly used in create-session() |
| | |

lock

| | |
|---------------|--|
| Syntax | lock(target) Use this method to lock datastores before editing |
| Parameters | target: "running" "candidate" "startup" |
| Returns | A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem) |
| Sample script | lock_unlock_partial_lock_unlock.py |
| See also | Unlock, partial-lock |

partial_lock

| | |
|------------|---|
| Syntax | partial_unlock(*args, **kwargs) A NETCONF operation that locks parts of the running datastore. |
| Parameters | *args: |

| | |
|---------------|--|
| | <p>One or more XPath expression that specifies the scope of the lock.</p> <p>**kwargs : <i><u>namespaces</u> = {} : prefix/namespace mapping</i></p> |
| Returns | A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem) |
| Sample script | lock_unlock_partial_lock_unlock.py |
| See also | partial-unlock, lock, unlock |

partial_unlock

| | |
|---------------|---|
| Syntax | <p>partial_unlock(lock_id="")</p> <p>Use this method to unlock a partially locked running configuration store</p> |
| Parameters | <p>lock_id:</p> <p>Identity of the lock to be unlocked. This lock-id MUST have been received as a response to a lock request by the manager during the current session, and MUST NOT have been sent in a previous unlock request.</p> |
| Returns | A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem) |
| Sample script | lock_unlock_partial_lock_unlock.py |
| See also | Partial-lock, lock, unlock |

print_rpc_stdout

| | |
|------------|--|
| Syntax | <p>print_rpc_stdout(flag=True)</p> <p>Use this method to enable or disable printing RPC data to stdout</p> |
| Parameters | Flag: True (to enable) or False (to disable) |
| | |

send_any_action

| | |
|----------------|--|
| Syntax | <code>send_any_action(*args, **kwargs)</code> This method sends an <action> command with associated data nodes. |
| Parameters | <p>*args: Form 1: nodeID1, value1, nodeID2, value2, ...or Form 1: nodeID1, valueDict1, nodeID2, valueDict2, ... Form 2: (nodeID1, value1), (nodeID2, value2), or (nodeID1, valueDict1), (nodeID2, valueDict2),</p> <p>The nodeID should be passed down in the order they should appear in the actual RPC message. The nodeID should be qualified with prefix. ValueDict can be { 'value' : value_to_set, 'raw_XML_data', True_or_False'}</p> <p>**kwargs: namespaces = prefix/namespace mapping dict used in nodeIDs</p> |
| Returns | A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem) |
| Sample scripts | <code>send_any_action.py</code> |
| See also | <code>send_any_rpc</code> |

send_any_rpc

| | |
|------------|--|
| Syntax | <code>send_any_rpc(rpc_name, *args, **kwargs)</code> Use this method to build and send any RPC (usually user-defined RPCs) |
| Parameters | <p>rpc_name: the name of the RPC to send. For example:, 'ncm:get-schema'</p> <p>*args: All input parameters that need to be assigned a value. i.e., input1, value1, input2, value2,...</p> <p>Each 'input' parameter is a relative nodeID, starting from (relative to) the RPC "input" node. All node name must be qualified with a prefix. The prefix is mapped to namespace according to namespaces keyword argument.</p> <p>For example: "ncm:identifier", identifier</p> <p>#foo is a container</p> |

| | |
|---------------|--|
| | <p>"pfx:foo/pfx:bar", "barvalue"</p> <p>**kwargs <i><code>namespaces = {} : prefix/namespace mapping</code></i></p> |
| Returns | <p>A tuple that contains a Boolean indicating if the operation succeeded and the XML reply message: (True / False, reply message XML)</p> <p>Output parameters can be extracted from the reply XML using <code>ncc.extract_value_from_rpc_reply</code>. If there is error, use <code>ncc.extract_rpc_error_from_rpc_reply</code> to extract error codes.</p> |
| Sample script | <code>send_any_rpc.py</code> under the installation root directory |
| See also | <p><code>extract_value_from_rpc_reply</code></p> <p><code>extract_rpc_error_from_rpc_reply</code></p> |

send_rpc_raw_xml

| | |
|---------------|--|
| Syntax | <p><code>send_rpc_raw_xml(msgxml)</code></p> <p>Use this method to send a RPC using its XML data that has been manually constructed.</p> |
| Parameters | <p><code>msgxml</code>:</p> <p>A RPC encoded in XML</p> |
| Returns | A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem) |
| Sample script | <code>send_raw_RPC_XML.py</code> |
| See also | <p><code>extract_value_from_rpc_reply</code></p> <p><code>extract_rpc_error_from_rpc_reply</code></p> |

set_indent_flag

| | |
|--------|---|
| Syntax | <p><code>set_indent_flag(flag=True)</code></p> <p>Use this method to determine if the RPC sent should be indented when transmitting on the wire</p> |
|--------|---|

| | |
|------------|---|
| Parameters | flag: a Boolean value: True or False |
| Returns | Nothing |

set_yang_path

| | |
|------------|---|
| Syntax | <code>set_yang_path(*args, recursive=True)</code> Set paths to YANG models used by the session and load them. The YANG path directories MUST be writable by the application since the application will need to generate XML based definition files. If no path is given, use the default path which is the same as in NETCONF GUI |
| Parameters | *args: Paths to YANG modules. Example: <code>set_yang_path("c:/path1", "c:/path2")</code> |
| Example | <code>high_level_load_yang_modules_lookup_and_validate_value.py</code> |
| Returns | All YANG module names loaded |

unlock

| | |
|------------|--|
| Syntax | <code>unlock(target)</code> Use this method to unlock locked datastores |
| Parameters | targe: "running" "candidate" "startup" |
| Returns | A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem) |

| | |
|---------------|------------------------------------|
| Sample script | lock_unlock_partial_lock_unlock.py |
| See also | lock, partial_unlock |

validate_config

| | |
|---------------|--|
| Syntax | validate_config(source) Use this method to request server to validate a config store |
| Parameters | source: The source can be 'running', 'candidate', 'startup', or a url that points to a config file, or an inline config content. The inline config content must be a well-formed <config> element representing an entire configuration datastore, not a subset of the running datastore. |
| Returns | A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem) |
| Sample script | validate_config.py |

xget

| | |
|---------------|---|
| Syntax | xget(filter, **kwargs) This method sends a NETCONF <get> request with xpath filtering. |
| Parameters | filter: xpath filter string kwargs: The optional keyword argument can be one of the following: withDefaults= 'explicit' 'report-all' 'trim' 'report-all-tagged' . See RFC6243 for details. namespaces = prefix/namespace mapping dict used in nodeIDs |
| Returns | A tuple contains five value with the first value indicating if the request succeeded. (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem) |
| Sample script | get_and_get_config.py |
| See also | sget, hget |

xget_config

| | |
|---------------|---|
| Syntax | <p><code>xget_config(filter, **kwargs)</code></p> <p>This method sends a NETCONF <get-config> request with xpath filter.</p> |
| Parameters | <p>filter: xpath filter string</p> <p>kwargs: The optional keyword argument can be one of the following:</p> <p>source = 'candidate' 'running' 'startup' If unspecified default to 'running'</p> <p>withDefaults= 'explicit' 'report-all' 'trim' 'report-all-tagged' . See RFC6243 for details.</p> <p>namespaces = prefix/namespace mapping dict used in nodeIDs</p> |
| Returns | A tuple that contains a Boolean indicating if the operation succeeded and the XML reply message: (True / False, reply message XML) |
| | |
| Sample script | <code>get_and_get_config.py</code> |
| See also | <code>sget_config</code> , <code>hget_config</code> |

High-level query, retrieval and edit commands**yang_lookup**

| | |
|------------|--|
| Syntax | <p><code>yang_lookup(ID, property, nsmap={})</code></p> <p>This method queries YANG property information of a data node from a loaded YANG module.</p> |
| Parameters | <p>ID: A node ID It can be given in the form of either “/module-name:node-name/module-name:node-name” ... or “/prefix:node-name/prefix:node-name....”. If the module-name or prefix of any node-name is the same as its previous node then it can be skipped. For example:</p> |

| | |
|---------------|--|
| | <ul style="list-style-type: none"> * /ietf-interfaces:interfaces/interface/ietf-ip:ipv4/enabled * /if:interfaces/interface/ip:ipv4/enabled * /if:interfaces/if:interface/ip:ipv4/ip:enabled <p>When using prefix an nsmmap dictionary maps the prefix used to its corresponding namespace {"if" : "<i>urn:ietf:params:xml:ns:yang:ietf-interfaces</i>"} must also be given</p> <p>property:</p> <ul style="list-style-type: none"> access : returns 'readwrite' or 'readonly' category: returns node type such as 'leaf', 'container', 'leaf-list', etc. config: returns True if it is config (readwrite) node otherwise False base-type: return the base type of a node (base of its syntax) default: returns default value if defined bit: returns bit strings base: returns the base of an identityref fraction-digits: returns fraction-digits defined for a decimal64 type node path: returns path value for a leafref pattern: returns patterns defined for a string type node require-instance: returns 'true' or 'false' enum: return enumerations defined for an enum type node if-feature: returns if-feature defined for a data node must: returns must condition defined for a data node unique: returns unique value defined for a data node min-elements: returns min-elements value defined for a data node max-elements: returns max-elements value defined for a data node ordered-by: returns ordered-by value defined for a data node key: returns key defined for a node length: returns length defined for an integer type data node mandatory: returns True or False range: returns range defined for a string type node presence: returns True if the container is a presence container status: returns status of a data node union_type: returns union types type: return syntax type of a data node <p>nsmmap: prefix/namespace mapping dictionary used in ID. Example {"if" : "<i>urn:ietf:params:xml:ns:yang:ietf-interfaces</i>"}</p> |
| Sample script | high_level_load_yang_modules_lookup_and_validate_value.py |
| See also | get_and_get_config.py high_level_create_merge_replace_delete_remove.py |

hget

| | |
|---------------|---|
| Syntax | <p><code>hget(*args, **kwargs)</code></p> <p>This method is a high level API to retrieve data from server.</p> |
| Parameters | <p>args: One or more IDs. ID is a node ID or YANG instance ID It can be given in the form of either “/module-name:instance-ID/module-name:instance-ID” ... or “/prefix:instance-ID/prefix:instance-ID....”. If the module- name or prefix of any node-name is the same as its previous node then it can be skipped.</p> <p style="color: green;">When using module name as prefix, no namespaces (nsmmap) is needed</p> <p>For example:</p> <pre> */ncm:netconf-state/schemas/schema[identifier='ietf-netconf-monitoring'][version='2010-10-04'][format='yang']/identifier" */ietf-interfaces:interfaces/interface/ietf-ip:ipv4/enabled */if:interfaces/interface/ip:ipv4/enabled */if:interfaces/if:interface/ip:ipv4/ip:enabled </pre> <p>When using prefix an nsmmap dictionary maps the prefix used to its corresponding namespace {“if”：“urn:ietf:params:xml:ns:yang:ietf-interfaces”} must also be given</p> <p>kwargs: The optional keyword argument can be one of the following:</p> <p><code>withDefaults= 'explicit' 'report-all' 'trim' 'report-all-tagged' .</code> See RFC6243 for details.</p> <p><code>namespaces = prefix/namespace mapping dict used in nodeIDs</code></p> |
| Returns | <p>A tuple of ((IID, valueAndnsmmapDict), unknownList , resultTuple).</p> <p>UnknownList is instance ID without corresponding loaded modules</p> <p>The resultTuple is a tuple of (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem)</p> <p>This method is the same as <code>hget_return_all</code>, except that it has the content matching nodes specified in the request removed from the returned IID. NETCONF standard requires the server to return all content node matching node as specified in the request.</p> |
| Sample script | <code>get_and_get_conf.py</code> |
| See also | <code>sget</code> , <code>xget</code> |

hget_return_all

| | |
|------|---|
| Note | See hget for details. This method is the same as hget, except that it returns all leaf values including content matching nodes in the IID. |
|------|---|

hget_config

| | |
|------------|--|
| Syntax | <p><code>hget_config(*args, **kwargs)</code></p> <p>This method is a high level API to retrieve data from server.</p> |
| Parameters | <p>args: One or more IDs. ID is a node ID or YANG instance ID It can be given in the form of either “/module-name:instance-ID/module-name:instance-ID” ... or “/prefix:instance-ID/prefix:instance-ID...”. If the module- name or prefix of any node-name is the same as its previous node then it can be skipped.</p> <p>When using module name as prefix, no namespaces (nsmmap) is needed</p> <p>For example:</p> <pre> */ncm:netconf-state/schemas/schema[identifier='ietf-netconf-monitoring'][version='2010-10-04'][format='yang']/identifier" * /ietf-interfaces:interfaces/interface/ietf-ip:ipv4/enabled * /if:interfaces/interface/ip:ipv4/enabled * /if:interfaces/if:interface/ip:ipv4/ip:enabled </pre> <p>When using prefix an nsmmap dictionary maps the prefix used to its corresponding namespace {"if":"urn:ietf:params:xml:ns:yang:ietf-interfaces"} must also be given</p> <p>kwargs: The optional keyword argument can be one of the following:</p> <p>source = one of 'candidate' 'running' 'startup' if unspecified default to 'running'</p> <p>withDefaults= 'explicit' 'report-all' 'trim' 'report-all-tagged' . See RFC6243 for details.</p> <p>namespaces = prefix/namespace mapping dict used in nodeIDs</p> |
| Returns | <p>A tuple of ((IID, valueAndnsmmapDict), unknownList, resultTuple).</p> <p>UnknownList is instance ID without corresponding loaded modules</p> <p>The resultTuple is a tuple of (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem)</p> |

| | |
|---------------|--|
| | This method is the same as <code>hget_return_all</code> , except that it has the content matching nodes specified in the request removed from the returned IID. NETCONF standard requires the server to return all content node matching node as specified in the request. |
| Sample script | <code>get_and_get_conf.py</code> |
| See also | <code>hget</code> , <code>xget</code> , <code>sget</code> |

hget_config_return_all

| | |
|------|--|
| Note | See <code>hget_config</code> for details. This method is the same as <code>hget_config</code> , except that it returns all leaf values including content matching nodes in the IID. |
|------|--|

create merge replace delete remove

| | |
|------------|--|
| Syntax | <pre>create(*args, **kwargs) merge(*args, **kwargs) replace(*args, **kwargs) delete(*args, **kwargs) remove(*args, **kwargs)</pre> <p>This method is a high level API to create/merge/replace/delete/remove data node in the server.</p> |
| Parameters | <p>args: ID1, value1_or_valueDict1, ID2, value2_or_valueDict2...</p> <p>ID is a node ID or YANG instance ID</p> <p>It can be given in the form of either “/module-name:ID/module-name:ID” ... or “/prefix:instance-ID/prefix:ID....”.</p> <p>If the module- name or prefix of any node-name is the same as its previous node then it can be skipped. For example:</p> <pre>* /ietf-system:system/contact * /sys:system/contact", "Joe", * /if:interfaces/interface/ip:ipv4/enabled * /if:interfaces/if:interface/ip:ipv4/ip:enabled</pre> <p>When using prefix an nsmapi dictionary maps the prefix used to its corresponding namespace {“if”:“urn:ietf:params:xml:ns:yang:ietf-interfaces”} must also be given</p> <p>The valueDict is a dict that may contain one of more of the following keys:</p> <pre>'value' : value to set the targeted node to 'insert_operation': first last before after, 'target_key' : List instance-identifier predicates</pre> |

```
'target_value'      : Leaf-list value,
'with_default'      : True | False
```

The purpose of each dictionary key element above is explained below.

The 'value' is the value that you want to modify for the node identified by the nodeID, For container node, list entry node, etc. it is an empty string.

For details of each operation means see RFC6241 <edit-config>

The 'insert_operation' is only applicable for YANG list or leaf-list that is defined as "ordered-by-user". It can be one of the following values: '

```
'first' | 'last' | 'before' | 'after'
```

When using 'insert_operation', you must also pass in 'target_key' for a YANG list, or 'target_value' for leaf-list. They are used for creating yang:key or yang:value attribute (RFC6020 7.8.6)

For a 'ordered-by-user' list, you must include "target_key" in the valueDict. The value of 'target_key' is an YANG instance-identifier predicates. Example:

```
[ex:name='fred']
[ex:ip='192.0.2.1'][ex:port='80']
```

For the complete syntax of predicates, see RFC6020: 9.13.

For an 'ordered-by-user' leaf-list, you must include "target_value" element in the valueDict.

The 'with_default' element in valueDict indicates whether to set 'default:true' attribute on the targeted node. When "with_default: True" is included in the valueDict, the value of the "value" key must be the schema default value of the node identified by nodeID. As defined in RFC6243, section 4.5.2.

Note for "delete" and "remove" operation, "with_default" does not apply.

****kwargs:**

```
target = 'candidate' | 'running' | 'startup'
        Default: 'candidate'
```

```
default_operations = "merge" | "replace" | "none"
        The default value for the <default-operation> parameter is "merge".
```

The <default-operation> parameter is optional.

```
test_operation = "test-then-set" | "set" | "test-only"
error_option = "rollback-on-error" | "stop-on-error" | "continue-on-
```

| | |
|---------------|---|
| | error" namespaces = prefix/namespace mapping dict used in IDs |
| Returns | The result is a tuple of (True/False, ReplyXML, Reply_lxml_Elem, RequestXML, Request_lxml_Elem) |
| Sample script | high_level_create_merge_replace_delete_remove.py |
| See also | high_level_hget_and_hget_conf.py |

Utility commands

extract_notification_name_timestamp

| | |
|---------------|--|
| Syntax | extract_notification_name_timestamp(notificationXML) This method is mostly used in a notification callback to parses a notification XML string. |
| Parameters | notificationXML: a notification message XML string |
| Returns | A tuple: (event Time, notification name, notification namespace) |
| Sample script | notification_subscription.py |
| See also | extract_rpc_error_from_rpc_reply, extract_value_from_rpc_reply |

wait_notification

| | |
|---------------|---|
| Syntax | wait_notification() This method causes application enter a loop and wait for notification. If any notification comes in, the registered callback function is executed. |
| Parameters | Nothing |
| Returns | Nothing |
| Sample script | notification_subscription.py |

| | |
|----------|---------------------|
| See also | create_subscription |
|----------|---------------------|

RESTCONF API

create_session_restconf

| | |
|------------|--|
| Syntax | <pre>create_session_restconf(host="localhost", port=8443, data_encoding = "json", user="", password="", scheme="https", authType="basic", client_cert = "", client_key=None, cacert=None, verify=True, timeout=30, http_version="1.1", defaultRootResource="/restconf", useDefaultRootResource = False,)</pre> <p>This method create a NETCONF over TLS session. The created session object is then used to send other RESTCONF commands.</p> |
| Parameters | <p>host: mandatory IP address or hostname of the RESTCONF sever to connect to</p> <p>port: optional port number of the RESTCONF sever to connect to. If not specified, default to the standard port 8443</p> <p>user, password: Only applicable when 'scheme' is set to use 'http'</p> <p>authType : Basic or Digest. Only applicable when 'scheme' is set to use 'http'</p> <p>scheme: https ot http (default 'https')</p> <p>http_version : 1.1 or 2 (default 2)</p> <p>defaultRootResource: The root resource to use when the root resource discovery fails (Default "/restconf"</p> <p>useDefaultRootResource: True or False (default to False),</p> |

| | |
|---------------|---|
| | <p>client_cert: mandatory Client certificate file with full path. The client certificate must be in PEM format and may contain client's private key.</p> <p>client_key: mandatory Client private key file with full path. This parameter can be left blank if the client certificate file already contains client's private key. The client key file must be in PEM format.</p> <p>cacert: mandatory Trusted CA certificate file with full path in PEM format. This is needed to validate the server's certificate and can be a chained certificate file.</p> <p>timeout: optional timeout period for connecting to the remote sever. If not specified, default 30 seconds.</p> <p>verify : True or False. Whether or not to verify the certificate sent by the RESTCONF server. (default True)</p> <p>socks_proxy If the server needs to be connected via a SOCKS4 or SOCKS5 proxy, provide required proxy settings in a dictionary: socks_proxy={"server":", "port":1080, "type":"SOCKS5", "user": ", "password":"} }</p> |
| Returns | A Session object for the session created. The returned object is then used to send other RESTCONF commands. |
| Sample script | restconf_example.py |
| | |

rget, head, options

| | |
|--------|---|
| Syntax | <p>rget(resourceID, **kwargs)</p> <p>Send a RESTCONF get, head or options</p> <p>The GET method is sent by the client to retrieve data and metadata for a resource.</p> <p>The HEAD method is sent by the client to retrieve just the header fields.</p> <p>The OPTIONS method is sent by the client to discover which methods are supported by the server for a specific resource (e.g., GET, POST, DELETE).</p> |
|--------|---|

| | |
|---------------|--|
| Parameters | <p>resourceID: A RESTConf data resource identifier</p> <p>kwargs: The keyword argument can be one of the following:</p> <p>withDefaults: 'explicit' 'report-all' 'trim' 'report-all-tagged' See RFC6243 for details on how to use this keyword argument.</p> <p>content: 'all' 'config' 'nonconfig'</p> <p>depth: 'unbounded' or the number of levels of child resources that are returned by the server for a GET method request.</p> <p>fields: select nodes to return</p> |
| Returns | <p>A Python Requests Response object</p> <p>https://docs.python-requests.org/en/latest</p> <p>https://docs.python-requests.org/en/latest/user/quickstart/#response-content</p> |
| Sample script | restconf_example.py |

post_data, put, post_actions, post_operations

| | |
|--------|---|
| Syntax | <pre>post_data(resourceID, data, **kwargs) post_actions(resourceID, data, **kwargs) post_operations(resourceID, data, **kwargs) put(resourceID, data, **kwargs)</pre> <p>Send a RESTCONF POST and PUT</p> <p>The POST method is sent by the client to create a data resource or invoke an operation resource. The server uses the target resource type to determine how to process the request.</p> <p>post_data: Set resourceID to an empty string to create a top-level configuration data resource or create a configuration data child resource</p> <p>post_action: Invoke an Action operation</p> <p>post_operations: Invoke an RPC operation</p> <p>put: Set resourceID to an empty string to create or replace a top-level configuration data resource or create or replace a configuration data child resource</p> |
|--------|---|

| | |
|---------------|---|
| Parameters | <p>resourceID: A RESTConf data resource identifier</p> <p>kwargs: The keyword argument can be one of the following:</p> <p>insert: 'first' 'last' 'before' 'after'</p> <p>point: the node resource ID when inserting 'after' or 'before'</p> <p>headers: {} headers dictionary. For example:</p> <pre>{ "If-Unmodified-Since" : "Tue, 02 Nov 2021 15:46:13 GMT" }</pre> |
| Returns | <p>A Python Requests Response object</p> <p>https://docs.python-requests.org/en/latest</p> <p>https://docs.python-requests.org/en/latest/user/quickstart/#response-content</p> |
| Sample script | restconf_example.py |

delete

| | |
|---------------|--|
| Syntax | <p>patch(resourceID, **kwargs)</p> <p>Send a RESTCONF DELETE</p> <p>If the target resource represents a configuration leaf-list or list data node, then it MUST represent a single YANG leaf-list or list instance.</p> |
| Parameters | <p>resourceID: A RESTConf data resource identifier to delete</p> <p>kwargs: The keyword argument can be one of the following:</p> <p>headers: {} headers dictionary. For example:</p> <pre>{ "If-Unmodified-Since" : "Tue, 02 Nov 2021 15:46:13 GMT" }</pre> |
| Returns | <p>A Python Requests Response object</p> <p>https://docs.python-requests.org/en/latest</p> <p>https://docs.python-requests.org/en/latest/user/quickstart/#response-content</p> |
| Sample script | restconf_example.py |

patch

| | |
|------------|--|
| Syntax | <p>patch(resourceID, data, **kwargs)</p> <p>Send a RESTCONF PATCH</p> <p>Plain patch can be used to create or update, but not delete, a child resource within the target resource.</p> |
| Parameters | <p>resourceID: A RESTConf data resource identifier</p> <p>kwargs: The keyword argument can be one of the following:</p> |

| | |
|---------------|---|
| | headers: {} headers dictionary. For example: <pre>{"If-Unmodified-Since" : "Tue, 02 Nov 2021 15:46:13 GMT"}</pre> |
| Returns | A Python Requests Response object https://docs.python-requests.org/en/latest https://docs.python-requests.org/en/latest/user/quickstart/#response-content |
| Sample script | restconf_example.py |