ncclient Documentation

Release 0.6.9

Shikhar Bhushan nd Leonidas Poulopoulos

Aug 08, 2020

CONTENTS

ncclient is a Python library for NETCONF clients. It aims to offer an intuitive API that sensibly maps the XML-encoded nature of NETCONF to Python constructs and idioms, and make writing network-management scripts easier. Other key features are:

- Supports all operations and capabilities defined in **RFC 6241**.
- Request pipelining.
- Asynchronous RPC requests.
- Keeping XML out of the way unless really needed.
- Extensible. New transport mappings and capabilities/operations can be easily added.

The best way to introduce is through a simple code example:

```
from ncclient import manager

# use unencrypted keys from ssh-agent or ~/.ssh keys, and rely on known_hosts
with manager.connect_ssh("host", username="user") as m:
    assert(":url" in m.server_capabilities)
    with m.locked("running"):
        m.copy_config(source="running", target="file:///new_checkpoint.conf")
        m.copy_config(source="file:///old_checkpoint.conf", target="running")
```

As of version 0.4 there has been an integration of Juniper's and Cisco's forks. Thus, lots of new concepts have been introduced that ease management of Juniper and Cisco devices respectively. The biggest change is the introduction of device handlers in connection params. For example to invoke Juniper's functions and params one has to re-write the above with **device_params={'name':'junos'}**:

```
from ncclient import manager

with manager.connect(host=host, port=830, username=user, hostkey_verify=False, device_
↪params={'name':'junos'}) as m:
    c = m.get_config(source='running').data_xml
    with open("%s.xml" % host, 'w') as f:
        f.write(c)
```

Respectively, for Cisco Nexus, the name is **nexus**. Device handlers are easy to implement and prove to be futureproof.

The latest pull request merge includes support for Huawei devices with name **huawei** in device_params.

CHAPTER

ONE

SUPPORTED DEVICE HANDLERS

- Juniper: *device_params={'name':'junos'}*
- **Cisco:**
  - CSR: *device_params={'name':'csr'}*
  - Nexus: *device_params={'name':'nexus'}*
  - IOS XR: *device_params={'name':'iosxr'}*
  - IOS XE: *device_params={'name':'iosxe'}*
- **Huawei:**
  - *device_params={'name':'huawei'}*
  - *device_params={'name':'huaweiyang'}*
- Alcatel Lucent: *device_params={'name':'alu'}*
- H3C: *device_params={'name':'h3c'}*
- HP Comware: *device_params={'name':'hpcomware'}*
- Server or anything not in above: *device_params={'name':'default'}*

Contents:

1.1 `manager` – High-level API

1.1.1 Customizing

These attributes control what capabilties are exchanged with the NETCONF server and what operations are available through the `Manager` API.

1.1.2 Factory functions

A `Manager` instance is created using a factory function.

1.1.3 Manager

Exposes an API for RPC operations as method calls. The return type of these methods depends on whether we are in `asynchronous or synchronous mode`.

In synchronous mode replies are awaited and the corresponding `RPCReply` object is returned. Depending on the `exception raising mode`, an *rpc-error* in the reply may be raised as an `RPCError` exception.

However in asynchronous mode, operations return immediately with the corresponding `RPC` object. Error handling and checking for whether a reply has been received must be dealt with manually. See the `RPC` documentation for details.

Note that in case of the `get()` and `get_config()` operations, the reply is an instance of `GetReply` which exposes the additional attributes `data` (as `Element`) and `data_xml` (as a string), which are of primary interest in case of these operations.

Presence of capabilities is verified to the extent possible, and you can expect a `MissingCapabilityError` if something is amiss. In case of transport-layer errors, e.g. unexpected session close, `TransportError` will be raised.

1.1.4 Special kinds of parameters

Some parameters can take on different types to keep the interface simple.

Source and target parameters

Where an method takes a *source* or *target* argument, usually a datastore name or URL is expected. The latter depends on the *:url* capability and on whether the specific URL scheme is supported. Either must be specified as a string. For example, *"running"*, *"ftp://user:pass@host/config"*.

If the source may be a *config* element, e.g. as allowed for the *validate* RPC, it can also be specified as an XML string or an `Element` object.

Filter parameters

Where a method takes a *filter* argument, it can take on the following types:

- A tuple of *(type, criteria)*.

  Here *type* has to be one of *"xpath"* or *"subtree"*.

  - For *"xpath"* the *criteria* should be a string containing the XPath expression or a tuple containing a dict of namespace mapping and the XPath expression.
  - For *"subtree"* the *criteria* should be an XML string or an `Element` object containing the criteria.

- A list of *spec*

  Here *type* has to be *"subtree"*.

  - the *spec* should be a list containing multiple XML string or multiple `Element` objects.

- A <*filter*> element as an XML string or an `Element` object.

1.2 Complete API documentation

1.2.1 `capabilities` – NETCONF Capabilities

`ncclient.capabilities.`**`schemes`**(*url_uri*)
Given a URI that has a *scheme* query string (i.e. *:url* capability URI), will return a list of supported schemes.

`class` `ncclient.capabilities.`**`Capabilities`**(*capabilities*)
Represents the set of capabilities available to a NETCONF client or server. It is initialized with a list of capability URI's.

Members

`":cap" in caps`
Check for the presence of capability. In addition to the URI, for capabilities of the form *urn:ietf:params:netconf:capability:$name:$version* their shorthand can be used as a key. For example, for *urn:ietf:params:netconf:capability:candidate:1.0* the shorthand would be *:candidate*. If version is significant, use *:candidate:1.0* as key.

`iter(caps)`
Return an iterator over the full URI's of capabilities represented by this object.

1.2.2 `xml_` – XML handling

Namespaces

Conversion

1.2.3 `transport` – Transport / Session layer

Base types

SSH session implementation

Errors

1.2.4 `operations` – Everything RPC

Base classes

Operations

Retrieval

Editing

Flowmon

Locking

Session

Subscribing

Exceptions

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

C

M

N

R

S