ncclient Documentation

Release 0.3.2

Shikhar Bhushan nd Leonidas Poulopoulos

January 26, 2014

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 4741**.
- 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.

It is suitable for Python 2.6+ (not Python 3 yet, though), and depends on paramiko 1.7.7.1+, an SSH library.

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

```python
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")
```

Contents:

CHAPTER 1

manager – High-level API

1.1 Customizing

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

1.2 Factory functions

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

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.

1.4 Special kinds of parameters

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

1.4.1 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.

1.4.2 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.
  - For *"subtree"* the *criteria* should be an XML string or an `Element` object containing the criteria.
- A *<filter>* element as an XML string or an `Element` object.

CHAPTER 2

Complete API documentation

2.1 capabilities – NETCONF Capabilities

2.2 xml_ – XML handling

2.2.1 Namespaces

2.2.2 Conversion

2.3 transport – Transport / Session layer

2.3.1 Base types

2.3.2 SSH session implementation

2.3.3 Errors

2.4 operations – Everything RPC

2.4.1 Base classes

2.4.2 Operations

Retrieval

Editing

Locking

Session

2.4.3 Exceptions

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*

Python Module Index