PyNSO Documentation

Release 1.1.0

Anthony Shaw

April 28, 2016

Contents

PyNSO is a Python library to interface to Cisco Network Services Orchestration (NSO).

Cisco® Network Services Orchestrator (NSO) provides a single pane of glass for orchestrating a multivendor network. To offer an exceptional range of support for multivendor devices, it uses network element drivers (NEDs). Traditionally, device adaptors are a major roadblock, since they are not upgraded at the same pace as device interfaces. But adding support for new devices can take months. Cisco NSO NEDs, in contrast, can add new commands and devices in weeks. These drivers also provide extremely fine-grained representation of relevant configuration commands. Using NEDs, NSO also makes device configuration commands available over a networkwide, multivendor command line interface (CLI), APIs, and user interface. In addition, NSO services like VPN can configure a complex multivendor network.

CHAPTER 1

NED Overview

Network element drivers comprise the network-facing part of NSO. They communicate over the native protocol supported by the device, such as Network Configuration Protocol (NETCONF), Representational State Transfer (REST), Extensible Markup Language (XML), CLI, and Simple Network Management Protocol (SNMP).

CHAPTER 2

Authors

This Python module was developed by the Research and Development team at Dimension Data. This module is provided under the Apache 2.0 license and the source code is available at github.com.

CHAPTER 3

Installation (stable version)

PyNSO is available on PyPi. You can install latest stable version using pip:

```
pip install pynso
```

CHAPTER 4

Installation (development version)

You can install latest development version from our Git repository:

```
pip install -e git+https://github.com/DimensionDataCBUSydney/pynso.git@trunk#egg=pynso
```

CHAPTER 5

Upgrading

If you used pip to install the library you can also use it to upgrade it:

```
pip install --upgrade pynso
```

CHAPTER 6

Using it

Using PyNSO is simple, the datastores are based on the NETCONF standard and the translation between Python dictionaries and YANG is completed by the module automatically.

```
from pprint import pprint

from pynso.client import NSOClient
from pynso.datastores import DatastoreType

# Setup a client
client = NSOClient('10.159.91.14', 'admin', 'admin')

# Get information about the API
print('Getting API version number')
pprint(client.info()['version'])

# Get the information about the running datastore
print('Getting the contents of the running datastore')
pprint(client.get_datastore(DatastoreType.RUNNING))

# Get a data path
print('Getting a specific data path: snmp:snmp namespace and the agent data object')
pprint(client.get_data(DatastoreType.RUNNING, ('snmp:snmp', 'agent')))
```

CHAPTER 7

Documentation

7.1 Main

7.1.1 Getting started

In order to enable REST in NSO, REST must be enabled in ncs.conf. The web server configuration for REST is shared with the WebUI's config. However, the WebUI does not have to be enabled for REST to work.

Here's a minimal example of what is needed in the conf file:

```
<rest>
    <enabled>true</enabled>
</rest>
<webui>
    <enabled>false</enabled>
    <transport>
        <tcp>
            <enabled>true</enabled>
            <ip>0.0.0.0</ip>
            <port>8080</port>
        </tcp>
    </transport>
</webui>
```

7.1.2 PyNSO API - Client

class `pynso.client.`**`NSOClient`**(*host*, *username*, *password*, *port=8080*, *ssl=False*)

`apply_rollback`(*datastore*, *name*)
Apply a system rollback

`create_data_value`(*datastore*, *data_path*, *data*, *params=None*)
Create (PUT) a data entry in a datastore

Parameters

- **`datastore`** (`DatastoreType`) – The target datastore
- **`data_path`** (`list` of `str` or `tuple`) – The list of paths
- **`data`** (`dict`) – The new value at the given path

Return type `bool`

Returns `True` if successful, otherwise error.

delete_path(*datastore*, *data_path*, *params=None*)
Delete a data entry in a datastore

Parameters

- **datastore** (`DatastoreType`) – The target datastore
- **data_path** (`list` of `str` or `tuple`) – The list of paths

Return type `bool`

Returns `True` if successful, otherwise error.

get_data(*datastore*, *data_path*, *params=None*)
Get a data entry in a datastore

Parameters

- **datastore** (`DatastoreType`) – The target datastore
- **data_path** (`list` of `str` or `tuple`) – The list of paths

get_datastore(*datastore*, *params=None*)
Get the details of a datastore

Parameters **datastore** (`DatastoreType`) – The target datastore

get_rollback(*name*)
Get a list of stored rollbacks

get_rollbacks()
Get a list of stored rollbacks

info()
Returns API information

set_data_value(*datastore*, *data_path*, *data*, *params=None*)
Update (POST) a data entry in a datastore

Parameters

- **datastore** (`DatastoreType`) – The target datastore
- **data_path** (`list` of `str` or `tuple`) – The list of paths
- **data** (`dict`) – The new value at the given path

Return type `bool`

Returns `True` if successful, otherwise error.

7.1.3 Enumerations

class `pynso.datastores.`**DatastoreType**
An enum of the resource types in the API.

Variables

- **config** – Link to the "config" resource
- **running** – Link to the "running" resource.
- **operational** – Link to the "operational" resource.

- **operations** – Container for available operations (i.e: YANG rpc statements).
- **rollbacks** – Container for available rollback files.

Note: Unless noted otherwise, all of the examples and code snippets in the documentation are licensed under the Apache 2.0 license.

A

C

D

G

I

N

S