ntc$_{r}osettaDocumentation$

Release 0.0.1

David Barroso

Jul 27, 2019

Contents

ntc_rosetta is leverages yangify to implement a set of "drivers" that can:

1. Transform network devices' native configuration/state into structured data that conform to YANG models
2. Transform data structures that conform to YANG models into network device's native configuration/data structures
3. Merge configurations

CHAPTER 1

Contents

1.1 Tutorials

1.1.1 Parsing (IOS)

One of the features ntc_rosetta supports is parsing native configuration and turning into data modelled after YANG models. For that purpose ntc_rosetta leverages yangify and builds on top of it to make it more consumable.

ntc_rosetta introduces the concept of "drivers". Drivers are objects that implements the parsing and translation of a given YANG model for a particular NOS. For instance, if you wanted to parse IOS configuration and convert it into data that follows the openconfig model you would load the corresponding driver like this:

[1]:
```
from ntc_rosetta import get_driver

ios = get_driver("ios", "openconfig")
ios_driver = ios()
```

The same processor can also translate the given model to native configuration.

Now, let's see how we can use this driver to parse IOS configuration and turn it into an Openconfig model. First, let's load some IOS configuration:

[2]:
```
with open("data/ios/config.txt", "r") as f:
    config = f.read()
```

[3]:
```
print(config)
```

```
interface FastEthernet1
   description This is Fa1
   shutdown
   exit
!
interface FastEthernet1.1
   description This is Fa1.1
```

(continues on next page)

(continued from previous page)

```
   exit
!
interface FastEthernet1.2
   description This is Fa1.2
   exit
!
interface FastEthernet3
   description This is Fa3
   no shutdown
   switchport mode access
   switchport access vlan 10
   exit
!
interface FastEthernet4
   shutdown
   switchport mode trunk
   switchport trunk allowed vlan 10,20
   exit
!
vlan 10
   name prod
   no shutdown
   exit
!
vlan 20
   name dev
   shutdown
   exit
!
```

Once the configuration is loaded, you need to parse it. The parser has some conventions you have to be aware of, for instance, when parsing configuration, it's going to expect you pass a `native` argument with a dictionary where the key `dev_conf` is the native configuration:

[4]:
```
parsed = ios_driver.parse(native={"dev_conf": config})
```

That's literally all you have to do parse the native configuration and turn it into structured data. We can check the result by dumping the `parsed.raw_value()`:

[5]:
```
import json
print(json.dumps(parsed.raw_value(), indent=4))
```

```
{
    "openconfig-interfaces:interfaces": {
        "interface": [
            {
                "name": "FastEthernet1",
                "config": {
                    "name": "FastEthernet1",
                    "type": "iana-if-type:ethernetCsmacd",
                    "description": "This is Fa1",
                    "enabled": false
                },
                "subinterfaces": {
                    "subinterface": [
                        {
```

(continues on next page)

(continued from previous page)

```
                        "index": 1,
                        "config": {
                            "index": 1,
                            "description": "This is Fa1.1"
                        }
                    },
                    {
                        "index": 2,
                        "config": {
                            "index": 2,
                            "description": "This is Fa1.2"
                        }
                    }
                ]
            }
        },
        {
            "name": "FastEthernet3",
            "config": {
                "name": "FastEthernet3",
                "type": "iana-if-type:ethernetCsmacd",
                "description": "This is Fa3",
                "enabled": true
            },
            "openconfig-if-ethernet:ethernet": {
                "openconfig-vlan:switched-vlan": {
                    "config": {
                        "interface-mode": "ACCESS",
                        "access-vlan": 10
                    }
                }
            }
        },
        {
            "name": "FastEthernet4",
            "config": {
                "name": "FastEthernet4",
                "type": "iana-if-type:ethernetCsmacd",
                "enabled": false
            },
            "openconfig-if-ethernet:ethernet": {
                "openconfig-vlan:switched-vlan": {
                    "config": {
                        "interface-mode": "TRUNK",
                        "trunk-vlans": [
                            10,
                            20
                        ]
                    }
                }
            }
        }
    ]
},
"openconfig-network-instance:network-instances": {
    "network-instance": [
        {
```

(continues on next page)

(continued from previous page)

```
                "name": "default",
                "config": {
                    "name": "default"
                },
                "vlans": {
                    "vlan": [
                        {
                            "vlan-id": 10,
                            "config": {
                                "vlan-id": 10,
                                "name": "prod",
                                "status": "ACTIVE"
                            }
                        },
                        {
                            "vlan-id": 20,
                            "config": {
                                "vlan-id": 20,
                                "name": "dev",
                                "status": "SUSPENDED"
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```

ntc_rosetta, also let's you parse some parts of the model, however, you need to be aware that might break the validation of the object:

[6]:
```
from yangson.exceptions import SemanticError
try:
    parsed_vlans = ios_driver.parse(
        native={"dev_conf": config},
        include=[
            "/openconfig-network-instance:network-instances/network-instance/vlans",
        ]
    )
except SemanticError as e:
    print(f"error: {e}")
```

```
error: [/openconfig-network-instance:network-instances/network-instance/0/name] ␣
↪instance-required
```

You can workaround this in two ways: 1. By disabling the validation of the object 2. By parsing all the necessary elements to make the object compliant.

You can disable the validation of the object by passing `validate=False`:

[7]:
```
parsed_vlans = ios_driver.parse(
    native={"dev_conf": config},
    validate=False,
    include=[
        "/openconfig-network-instance:network-instances/network-instance/vlans",
    ]
```

(continues on next page)

(continued from previous page)

```
)
print(json.dumps(parsed_vlans.raw_value(), indent=4))
```


```
{
    "openconfig-network-instance:network-instances": {
        "network-instance": [
            {
                "name": "default",
                "vlans": {
                    "vlan": [
                        {
                            "vlan-id": 10,
                            "config": {
                                "vlan-id": 10,
                                "name": "prod",
                                "status": "ACTIVE"
                            }
                        },
                        {
                            "vlan-id": 20,
                            "config": {
                                "vlan-id": 20,
                                "name": "dev",
                                "status": "SUSPENDED"
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```


And you can make sure your object is valid by passing the list of elements that are needed to make the object compliant:

[8]:
```
parsed_vlans = ios_driver.parse(
    native={"dev_conf": config},
    include=[
        "/openconfig-network-instance:network-instances/network-instance/name",
        "/openconfig-network-instance:network-instances/network-instance/config",
        "/openconfig-network-instance:network-instances/network-instance/vlans",
    ]
)
print(json.dumps(parsed_vlans.raw_value(), indent=4))
```


```
{
    "openconfig-network-instance:network-instances": {
        "network-instance": [
            {
                "name": "default",
                "config": {
                    "name": "default"
                },
                "vlans": {
                    "vlan": [
                        {
                            "vlan-id": 10,
                            "config": {
```


(continues on next page)

(continued from previous page)

```
                                "vlan-id": 10,
                                "name": "prod",
                                "status": "ACTIVE"
                            }
                        },
                        {
                            "vlan-id": 20,
                            "config": {
                                "vlan-id": 20,
                                "name": "dev",
                                "status": "SUSPENDED"
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```

1.1.2 Navigating Data

In the previous tutorial we saw how to parse configuration. In this tutorial we are going to see how we can navigate the data we extracted.

Let's start by parsing the configuration as in the previous example:

[1]:
```
from ntc_rosetta import get_driver

ios = get_driver("ios", "openconfig")
ios_driver = ios()

with open("data/ios/config.txt", "r") as f:
    config = f.read()

parsed = ios_driver.parse(native={"dev_conf": config})
```

Raw value

The most basic form of navigating the data is by using the method `raw_value`, which returns the object using only builtin datastructures:

[2]:
```
raw = parsed.raw_value()
raw
```

[2]:
```
{'openconfig-interfaces:interfaces': {'interface': [{'name': 'FastEthernet1',
    'config': {'name': 'FastEthernet1',
     'type': 'iana-if-type:ethernetCsmacd',
     'description': 'This is Fa1',
     'enabled': False},
    'subinterfaces': {'subinterface': [{'index': 1,
       'config': {'index': 1, 'description': 'This is Fa1.1'}},
      {'index': 2, 'config': {'index': 2, 'description': 'This is Fa1.2'}}]}},
   {'name': 'FastEthernet3',
```

(continues on next page)

(continued from previous page)

```
    'config': {'name': 'FastEthernet3',
     'type': 'iana-if-type:ethernetCsmacd',
     'description': 'This is Fa3',
     'enabled': True},
    'openconfig-if-ethernet:ethernet': {'openconfig-vlan:switched-vlan': {'config': {
↪'interface-mode': 'ACCESS',
       'access-vlan': 10}}}},
   {'name': 'FastEthernet4',
    'config': {'name': 'FastEthernet4',
     'type': 'iana-if-type:ethernetCsmacd',
     'enabled': False},
    'openconfig-if-ethernet:ethernet': {'openconfig-vlan:switched-vlan': {'config': {
↪'interface-mode': 'TRUNK',
       'trunk-vlans': [10, 20]}}}}]},
 'openconfig-network-instance:network-instances': {'network-instance': [{'name':
↪'default',
    'config': {'name': 'default'},
    'vlans': {'vlan': [{'vlan-id': 10,
       'config': {'vlan-id': 10, 'name': 'prod', 'status': 'ACTIVE'}},
      {'vlan-id': 20,
       'config': {'vlan-id': 20, 'name': 'dev', 'status': 'SUSPENDED'}}]}}]}}
```

[3]:
```
print(raw["openconfig-interfaces:interfaces"]["interface"][0]["config"]["description
↪"])
```

```
This is Fa1
```

Instance identifiers

You can also use instance identifiers to get data from the object, to do so use the method `peek`:

[4]:
```
parsed.peek("/openconfig-interfaces:interfaces/interface=FastEthernet1/config/
↪description")
```

[4]:
```
'This is Fa1'
```

[5]:
```
parsed.peek("/openconfig-interfaces:interfaces/interface=FastEthernet1/subinterfaces/
↪subinterface=1/config/description")
```

[5]:
```
'This is Fa1.1'
```

[6]:
```
parsed.peek("/openconfig-interfaces:interfaces/interface=FastEthernet3/openconfig-if-
↪ethernet:ethernet/openconfig-vlan:switched-vlan")
```

[6]:
```
{'config': {'interface-mode': 'ACCESS', 'access-vlan': 10}}
```

1.1.3 Translating (IOS)

As explained in the previous tutorial, a ntc_rosetta driver can both parse and translate between native and yang-based models. In this tutorial we are going to translate data that complies to the openconfig model into native IOS configuration.

Let's start by loading the needed driver:

[1]:
```
from ntc_rosetta import get_driver

ios = get_driver("ios", "openconfig")
ios_processor = ios()
```

Now we need some data:

[2]:
```
data = {
    "openconfig-interfaces:interfaces": {
        "interface": [
            {
                "name": "FastEthernet1",
                "config": {
                    "name": "FastEthernet1",
                    "type": "iana-if-type:ethernetCsmacd",
                    "description": "This is Fa1",
                    "enabled": False
                },
                "subinterfaces": {
                    "subinterface": [
                        {
                            "index": 1,
                            "config": {
                                "index": 1,
                                "description": "This is Fa1.1"
                            }
                        },
                        {
                            "index": 2,
                            "config": {
                                "index": 2,
                                "description": "This is Fa1.2"
                            }
                        }
                    ]
                }
            },
            {
                "name": "FastEthernet3",
                "config": {
                    "name": "FastEthernet3",
                    "type": "iana-if-type:ethernetCsmacd",
                    "description": "This is Fa3",
                    "enabled": True
                },
                "openconfig-if-ethernet:ethernet": {
                    "openconfig-vlan:switched-vlan": {
                        "config": {
                            "interface-mode": "ACCESS",
                            "access-vlan": 10
                        }
                    }
                }
            },
            {
                "name": "FastEthernet4",
                "config": {
```

(continues on next page)

(continued from previous page)


```
                    "name": "FastEthernet4",
                    "type": "iana-if-type:ethernetCsmacd",
                    "enabled": False
                },
                "openconfig-if-ethernet:ethernet": {
                    "openconfig-vlan:switched-vlan": {
                        "config": {
                            "interface-mode": "TRUNK",
                            "trunk-vlans": [
                                10,
                                20
                            ]
                        }
                    }
                }
            }
        ]
    },
    "openconfig-network-instance:network-instances": {
        "network-instance": [
            {
                "name": "default",
                "config": {
                    "name": "default"
                },
                "vlans": {
                    "vlan": [
                        {
                            "vlan-id": 10,
                            "config": {
                                "vlan-id": 10,
                                "name": "prod",
                                "status": "ACTIVE"
                            }
                        },
                        {
                            "vlan-id": 20,
                            "config": {
                                "vlan-id": 20,
                                "name": "dev",
                                "status": "SUSPENDED"
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```


Once we have the data, translating it to native is very simple:

[3]:
```
native = ios_processor.translate(candidate=data)
```

We can verify the result by just printing it:

[4]:
```
print(native)
```

```
interface FastEthernet1
   description This is Fa1
   shutdown
   exit
!
interface FastEthernet1.1
   description This is Fa1.1
   exit
!
interface FastEthernet1.2
   description This is Fa1.2
   exit
!
interface FastEthernet3
   description This is Fa3
   no shutdown
   switchport mode access
   switchport access vlan 10
   exit
!
interface FastEthernet4
   shutdown
   switchport mode trunk
   switchport trunk allowed vlan 10,20
   exit
!
vlan 10
   name prod
   no shutdown
   exit
!
vlan 20
   name dev
   shutdown
   exit
!
```

1.1.4 Merge (IOS)

In addition to translating models to native configuration, ntc_rosetta can create configuration deltas that can be applied into the device. This means that given to different sets of data, ntc_rosetta can compute the needed native commands to go from one to the other.

To see what this means let's see it with an example. Let's start by loading the driver:

[1]:
```
from ntc_rosetta import get_driver

ios = get_driver("ios", "openconfig")
ios_processor = ios()
```

Now we load some data that will represent the "running" configuration:

[2]:
```
running = {
    "openconfig-interfaces:interfaces": {
        "interface": [
            {
                "name": "FastEthernet1",
                "config": {
                    "name": "FastEthernet1",
                    "type": "iana-if-type:ethernetCsmacd",
                    "description": "This is Fa1",
                    "enabled": False
                },
                "subinterfaces": {
                    "subinterface": [
                        {
                            "index": 1,
                            "config": {
                                "index": 1,
                                "description": "This is Fa1.1"
                            }
                        },
                        {
                            "index": 2,
                            "config": {
                                "index": 2,
                                "description": "This is Fa1.2"
                            }
                        }
                    ]
                }
            },
            {
                "name": "FastEthernet3",
                "config": {
                    "name": "FastEthernet3",
                    "type": "iana-if-type:ethernetCsmacd",
                    "description": "This is Fa3",
                    "enabled": True
                },
                "openconfig-if-ethernet:ethernet": {
                    "openconfig-vlan:switched-vlan": {
                        "config": {
                            "interface-mode": "ACCESS",
                            "access-vlan": 10
                        }
                    }
                }
            },
            {
                "name": "FastEthernet4",
                "config": {
                    "name": "FastEthernet4",
                    "type": "iana-if-type:ethernetCsmacd",
                    "enabled": False
                },
                "openconfig-if-ethernet:ethernet": {
                    "openconfig-vlan:switched-vlan": {
                        "config": {
```

(continues on next page)

(continued from previous page)

```
                            "interface-mode": "TRUNK",
                            "trunk-vlans": [
                                10,
                                20
                            ]
                        }
                    }
                }
            }
        ]
    },
    "openconfig-network-instance:network-instances": {
        "network-instance": [
            {
                "name": "default",
                "config": {
                    "name": "default"
                },
                "vlans": {
                    "vlan": [
                        {
                            "vlan-id": 10,
                            "config": {
                                "vlan-id": 10,
                                "name": "prod",
                                "status": "ACTIVE"
                            }
                        },
                        {
                            "vlan-id": 20,
                            "config": {
                                "vlan-id": 20,
                                "name": "dev",
                                "status": "SUSPENDED"
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```

Now we are going to copy this data into a "candidate" variable and apply some changes:

[3]:
```
from copy import deepcopy
candidate = deepcopy(running)
```

We are going to start by disabling vlan 10:

[4]:
```
vlan_10 = candidate["openconfig-network-instance:network-instances"]["network-instance
↪"][0]["vlans"]["vlan"][0]
vlan_10["config"]["status"] = "SUSPENDED"
```

Eliminate vlan 20:

[5]:
```
candidate["openconfig-network-instance:network-instances"]["network-instance"][0][
↪"vlans"]["vlan"].pop(1)
```

[5]:
```
{'vlan-id': 20,
 'config': {'vlan-id': 20, 'name': 'dev', 'status': 'SUSPENDED'}}
```

And create a new vlan 30:

[6]:
```
vlan_30 = {
    "vlan-id": 30,
    "config": {
        "vlan-id": 30,
        "name": "staging",
        "status": "ACTIVE"
    }
}
candidate["openconfig-network-instance:network-instances"]["network-instance"][0][
↪"vlans"]["vlan"].append(vlan_30)
```

Once we have done those changes we can merge those two objects like this:

[7]:
```
config = ios_processor.merge(candidate=candidate, running=running)
```

Finally, printing the config variable should return the native commands needed for that merge operation:

[8]:
```
print(config)
```

```
no vlan 20
vlan 10
   shutdown
   exit
!
vlan 30
   name staging
   no shutdown
   exit
!
```

1.1.5 Merge native configurations (IOS)

In our previous example we merged two objects that already complied with the openconfig models. In this example, we are going to merge to native configurations.

Let's start by loading the driver as usual:

[1]:
```
from ntc_rosetta import get_driver

ios = get_driver("ios", "openconfig")
ios_processor = ios()
```

Now we are going to load a file with the "running" configuration:

[2]:
```
with open("data/ios/config.txt", "r") as f:
    running_config = f.read()
```

And now a different file with the "candidate" config:

[3]:
```
with open("data/ios/new_config.txt", "r") as f:
    candidate_config = f.read()
```

Let's see the files side by side highlighting the differences:

[4]:
```
!diff -y data/ios/config.txt data/ios/new_config.txt
```

```
interface FastEthernet1                                         interface␣
↪FastEthernet1
   description This is Fa1                                         description This␣
↪is Fa1
   shutdown                                                        shutdown
   exit                                                            exit
!                                                               !
interface FastEthernet1.1                                       interface␣
↪FastEthernet1.1
   description This is Fa1.1                                       description This␣
↪is Fa1.1
   exit                                                            exit
!                                                               !
interface FastEthernet1.2                                       interface␣
↪FastEthernet1.2
   description This is Fa1.2                                       description This␣
↪is Fa1.2
   exit                                                            exit
!                                                               !
interface FastEthernet3                                         interface␣
↪FastEthernet3
   description This is Fa3                                         description This␣
↪is Fa3
   no shutdown                                                     no shutdown
   switchport mode access                                          switchport mode␣
↪access
   switchport access vlan 10                                       switchport access␣
↪vlan 10
   exit                                                            exit
!                                                               !
interface FastEthernet4                                         interface␣
↪FastEthernet4
   shutdown                                                        shutdown
   switchport mode trunk                                           switchport mode␣
↪trunk
   switchport trunk allowed vlan 10,20                             switchport trunk␣
↪allowed vlan 10,20
   exit                                                            exit
!                                                               !
vlan 10                                                         vlan 10
   name prod                                                       name prod
   no shutdown                                                 |   shutdown
   exit                                                            exit
!                                                               !
vlan 20                                                         | vlan 30
   name dev                                                     |   name staging
   shutdown                                                     |   no shutdown
   exit                                                            exit
!                                                               !
```

As you can see vlan 20 is gone, vlan 10 has been suspended and there is a new vlan 30.

Now let's parse those configurations as we did in our first tutorial:

[5]:
```
parsed_candidate = ios_processor.parse(native={"dev_conf": candidate_config})
parsed_running = ios_processor.parse(native={"dev_conf": running_config})
```

Now that we have parsed both native configurations, doing a merge operation is identical as in our previous tutorial:

[6]:
```
config = ios_processor.merge(
    candidate=parsed_candidate.raw_value(),
    running=parsed_running.raw_value()
)
print(config)
```

```
no vlan 20
vlan 10
   shutdown
   exit
!
vlan 30
   name staging
   no shutdown
   exit
!
```

1.1.6 Parsing (JUNOS)

One of the features ntc_rosetta supports is parsing native configuration and turning into data modelled after YANG models. For that purpose ntc_rosetta leverages yangify and builds on top of it to make it more consumable.

ntc_rosetta introduces the concept of "drivers". Drivers are objects that implements the parsing and translation of a given YANG model for a particular NOS. For instance, if you wanted to parse IOS configuration and convert it into data that follows the openconfig model you would load the corresponding driver like this:

[1]:
```
from ntc_rosetta import get_driver

junos = get_driver("junos", "openconfig")
junos_driver = junos()
```

The same processor can also translate the given model to native configuration.

Now, let's see how we can use this driver to parse IOS configuration and turn it into an Openconfig model. First, let's load some IOS configuration:

[3]:
```
with open("data/junos/dev_conf.xml", "r") as f:
    config = f.read()
```

[4]:
```
print(config)
```

```
<configuration>
        <interfaces>
            <interface>
                <name>xe-0/0/1</name>
                <unit>
                    <name>0</name>
                    <family>
                        <ethernet-switching>
```

(continues on next page)

(continued from previous page)

```
                        <interface-mode>access</interface-mode>
                        <vlan>
                            <members>10</members>
                        </vlan>
                    </ethernet-switching>
                </family>
            </unit>
        </interface>
        <interface>
            <name>xe-0/0/3</name>
            <unit>
                <name>0</name>
                <family>
                    <ethernet-switching>
                        <interface-mode>trunk</interface-mode>
                        <vlan>
                            <members>10</members>
                            <members>20</members>
                        </vlan>
                    </ethernet-switching>
                </family>
            </unit>
        </interface>
        <interface>
            <name>xe-0/0/4</name>
            <unit>
                <name>0</name>
                <family>
                    <ethernet-switching>
                        <interface-mode>trunk</interface-mode>
                        <vlan>
                            <members>VLAN-100</members>
                            <members>VLAN-200</members>
                        </vlan>
                    </ethernet-switching>
                </family>
            </unit>
        </interface>
        <interface>
            <name>xe-0/0/5</name>
            <unit>
                <name>0</name>
                <family>
                    <ethernet-switching>
                        <interface-mode>access</interface-mode>
                        <vlan>
                            <members>VLAN-100</members>
                        </vlan>
                    </ethernet-switching>
                </family>
            </unit>
        </interface>
    </interfaces>
<vlans>
        <vlan>
            <name>default</name>
            <vlan-id>1</vlan-id>
```

(continues on next page)

(continued from previous page)

```
            </vlan>
            <vlan>
                <name>prod</name>
                <vlan-id>20</vlan-id>
            </vlan>
            <vlan inactive="inactive">
                <vlan-id>10</vlan-id>
            </vlan>
            <vlan>
                <name>VLAN-100</name>
                <vlan-id>100</vlan-id>
            </vlan>
            <vlan>
                <name>VLAN-200</name>
                <vlan-id>200</vlan-id>
            </vlan>
    </vlans>
</configuration>
```

Once the configuration is loaded, you need to parse it. The parser has some conventions you have to be aware of, for instance, when parsing configuration, it's going to expect you pass a `native` argument with a dictionary where the key `dev_conf` is the native configuration:

[7]:
```
parsed = junos_driver.parse(native={"dev_conf": config})
```

That's literally all you have to do parse the native configuration and turn it into structured data. We can check the result by dumping the `parsed.raw_value()`:

[9]:
```
import json
print(json.dumps(parsed.raw_value(), indent=4))
```

```
{
    "openconfig-interfaces:interfaces": {
        "interface": [
            {
                "name": "xe-0/0/1",
                "config": {
                    "name": "xe-0/0/1",
                    "type": "iana-if-type:ethernetCsmacd",
                    "enabled": true
                },
                "subinterfaces": {
                    "subinterface": [
                        {
                            "index": 0,
                            "config": {
                                "index": 0
                            }
                        }
                    ]
                },
                "openconfig-if-ethernet:ethernet": {
                    "openconfig-vlan:switched-vlan": {
                        "config": {
                            "interface-mode": "ACCESS",
                            "access-vlan": 10
```

(continues on next page)

(continued from previous page)

```
                    }
                }
            }
        },
        {
            "name": "xe-0/0/3",
            "config": {
                "name": "xe-0/0/3",
                "type": "iana-if-type:ethernetCsmacd",
                "enabled": true
            },
            "subinterfaces": {
                "subinterface": [
                    {
                        "index": 0,
                        "config": {
                            "index": 0
                        }
                    }
                ]
            },
            "openconfig-if-ethernet:ethernet": {
                "openconfig-vlan:switched-vlan": {
                    "config": {
                        "interface-mode": "TRUNK",
                        "trunk-vlans": [
                            10,
                            20
                        ]
                    }
                }
            }
        },
        {
            "name": "xe-0/0/4",
            "config": {
                "name": "xe-0/0/4",
                "type": "iana-if-type:ethernetCsmacd",
                "enabled": true
            },
            "subinterfaces": {
                "subinterface": [
                    {
                        "index": 0,
                        "config": {
                            "index": 0
                        }
                    }
                ]
            },
            "openconfig-if-ethernet:ethernet": {
                "openconfig-vlan:switched-vlan": {
                    "config": {
                        "interface-mode": "TRUNK",
                        "trunk-vlans": [
                            100,
                            200
```

(continues on next page)

(continued from previous page)

```
                        ]
                    }
                }
            }
        },
        {
            "name": "xe-0/0/5",
            "config": {
                "name": "xe-0/0/5",
                "type": "iana-if-type:ethernetCsmacd",
                "enabled": true
            },
            "subinterfaces": {
                "subinterface": [
                    {
                        "index": 0,
                        "config": {
                            "index": 0
                        }
                    }
                ]
            },
            "openconfig-if-ethernet:ethernet": {
                "openconfig-vlan:switched-vlan": {
                    "config": {
                        "interface-mode": "ACCESS",
                        "access-vlan": 100
                    }
                }
            }
        }
    ]
},
"openconfig-network-instance:network-instances": {
    "network-instance": [
        {
            "name": "default",
            "config": {
                "name": "default"
            },
            "vlans": {
                "vlan": [
                    {
                        "vlan-id": 1,
                        "config": {
                            "vlan-id": 1,
                            "name": "default",
                            "status": "ACTIVE"
                        }
                    },
                    {
                        "vlan-id": 20,
                        "config": {
                            "vlan-id": 20,
                            "name": "prod",
                            "status": "ACTIVE"
                        }
```

(continues on next page)

(continued from previous page)

```
                        },
                        {
                            "vlan-id": 10,
                            "config": {
                                "vlan-id": 10,
                                "status": "SUSPENDED"
                            }
                        },
                        {
                            "vlan-id": 100,
                            "config": {
                                "vlan-id": 100,
                                "name": "VLAN-100",
                                "status": "ACTIVE"
                            }
                        },
                        {
                            "vlan-id": 200,
                            "config": {
                                "vlan-id": 200,
                                "name": "VLAN-200",
                                "status": "ACTIVE"
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```

ntc_rosetta, also let's you parse some parts of the model, however, you need to be aware that might break the validation of the object:

[11]:
```
from yangson.exceptions import SemanticError
try:
    parsed_vlans = junos_driver.parse(
        native={"dev_conf": config},
        include=[
            "/openconfig-network-instance:network-instances/network-instance/vlans",
        ]
    )
except SemanticError as e:
    print(f"error: {e}")
```

```
error: [/openconfig-network-instance:network-instances/network-instance/0/name]␣
↪instance-required
```

You can workaround this in two ways: 1. By disabling the validation of the object 2. By parsing all the necessary elements to make the object compliant.

You can disable the validation of the object by passing `validate=False`:

[13]:
```
parsed_vlans = junos_driver.parse(
    native={"dev_conf": config},
    validate=False,
    include=[
```

(continues on next page)

(continued from previous page)

```
        "/openconfig-network-instance:network-instances/network-instance/vlans",
    ]
)
print(json.dumps(parsed_vlans.raw_value(), indent=4))
```


```
{
    "openconfig-network-instance:network-instances": {
        "network-instance": [
            {
                "name": "default",
                "vlans": {
                    "vlan": [
                        {
                            "vlan-id": 1,
                            "config": {
                                "vlan-id": 1,
                                "name": "default",
                                "status": "ACTIVE"
                            }
                        },
                        {
                            "vlan-id": 20,
                            "config": {
                                "vlan-id": 20,
                                "name": "prod",
                                "status": "ACTIVE"
                            }
                        },
                        {
                            "vlan-id": 10,
                            "config": {
                                "vlan-id": 10,
                                "status": "SUSPENDED"
                            }
                        },
                        {
                            "vlan-id": 100,
                            "config": {
                                "vlan-id": 100,
                                "name": "VLAN-100",
                                "status": "ACTIVE"
                            }
                        },
                        {
                            "vlan-id": 200,
                            "config": {
                                "vlan-id": 200,
                                "name": "VLAN-200",
                                "status": "ACTIVE"
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```

And you can make sure your object is valid by passing the list of elements that are needed to make the object compliant:

[15]:
```
parsed_vlans = junos_driver.parse(
    native={"dev_conf": config},
    include=[
        "/openconfig-network-instance:network-instances/network-instance/name",
        "/openconfig-network-instance:network-instances/network-instance/config",
        "/openconfig-network-instance:network-instances/network-instance/vlans",
    ]
)
print(json.dumps(parsed_vlans.raw_value(), indent=4))
```

```
{
    "openconfig-network-instance:network-instances": {
        "network-instance": [
            {
                "name": "default",
                "config": {
                    "name": "default"
                },
                "vlans": {
                    "vlan": [
                        {
                            "vlan-id": 1,
                            "config": {
                                "vlan-id": 1,
                                "name": "default",
                                "status": "ACTIVE"
                            }
                        },
                        {
                            "vlan-id": 20,
                            "config": {
                                "vlan-id": 20,
                                "name": "prod",
                                "status": "ACTIVE"
                            }
                        },
                        {
                            "vlan-id": 10,
                            "config": {
                                "vlan-id": 10,
                                "status": "SUSPENDED"
                            }
                        },
                        {
                            "vlan-id": 100,
                            "config": {
                                "vlan-id": 100,
                                "name": "VLAN-100",
                                "status": "ACTIVE"
                            }
                        },
                        {
                            "vlan-id": 200,
                            "config": {
                                "vlan-id": 200,
                                "name": "VLAN-200",
```

(continues on next page)

(continued from previous page)

```
                                "status": "ACTIVE"
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```

[ ]:

1.1.7 Translating (JUNOS)

As explained in the previous tutorial, a ntc_rosetta driver can both parse and translate between native and yang-based models. In this tutorial we are going to translate data that complies to the openconfig model into native Junos (XML) configuration.

Let's start by loading the needed driver:

[5]:
```
from ntc_rosetta import get_driver

junos = get_driver("junos", "openconfig")
junos_processor = junos()
```

Now we need some data:

[6]:
```
data = {
    "openconfig-interfaces:interfaces": {
        "interface": [
            {
                "name": "xe-0/0/1",
                "config": {
                    "name": "xe-0/0/1",
                    "type": "iana-if-type:ethernetCsmacd",
                    "description": "This is xe-0/0/1",
                    "enabled": False
                },
                "subinterfaces": {
                    "subinterface": [
                        {
                            "index": 1,
                            "config": {
                                "index": 1,
                                "description": "This is xe-0/0/1.1"
                            }
                        },
                        {
                            "index": 2,
                            "config": {
                                "index": 2,
                                "description": "This is xe-0/0/1.2"
                            }
                        }
                    ]
```

(continues on next page)

(continued from previous page)

```
                }
            },
            {
                "name": "xe-0/0/2",
                "config": {
                    "name": "xe-0/0/2",
                    "type": "iana-if-type:ethernetCsmacd",
                    "description": "This is xe-0/0/2",
                    "enabled": True
                },
                "openconfig-if-ethernet:ethernet": {
                    "openconfig-vlan:switched-vlan": {
                        "config": {
                            "interface-mode": "ACCESS",
                            "access-vlan": 10
                        }
                    }
                }
            },
            {
                "name": "xe-0/0/3",
                "config": {
                    "name": "xe-0/0/3",
                    "type": "iana-if-type:ethernetCsmacd",
                    "enabled": False
                },
                "openconfig-if-ethernet:ethernet": {
                    "openconfig-vlan:switched-vlan": {
                        "config": {
                            "interface-mode": "TRUNK",
                            "trunk-vlans": [
                                10,
                                20
                            ]
                        }
                    }
                }
            }
        ]
    },
    "openconfig-network-instance:network-instances": {
        "network-instance": [
            {
                "name": "default",
                "config": {
                    "name": "default"
                },
                "vlans": {
                    "vlan": [
                        {
                            "vlan-id": 10,
                            "config": {
                                "vlan-id": 10,
                                "name": "prod",
                                "status": "ACTIVE"
                            }
                        },
```

(continues on next page)

(continued from previous page)

```
                        {
                            "vlan-id": 20,
                            "config": {
                                "vlan-id": 20,
                                "name": "dev",
                                "status": "SUSPENDED"
                            }
                        }
                    ]
                }
            }
        ]
    }
}
```

Once we have the data, translating it to native is very simple:

[7]:
```
native = junos_processor.translate(candidate=data)
```

We can verify the result by just printing it:

[8]:
```
print(native)
```

```
<configuration>
  <interfaces>
    <interface>
      <name>xe-0/0/1</name>
      <disable/>
      <unit>
        <name>1</name>
        <description>This is xe-0/0/1.1</description>
      </unit>
      <unit>
        <name>2</name>
        <description>This is xe-0/0/1.2</description>
      </unit>
    </interface>
    <interface>
      <name>xe-0/0/2</name>
      <disable delete="delete"/>
      <unit>
        <name>0</name>
        <family>
          <ethernet-switching>
            <interface-mode>access</interface-mode>
            <vlan>
              <members>10</members>
            </vlan>
          </ethernet-switching>
        </family>
      </unit>
    </interface>
    <interface>
      <name>xe-0/0/3</name>
      <disable/>
      <unit>
        <name>0</name>
```

(continues on next page)

(continued from previous page)

```
        <family>
          <ethernet-switching>
            <interface-mode>trunk</interface-mode>
            <vlan>
              <members>10</members>
              <members>20</members>
            </vlan>
          </ethernet-switching>
        </family>
      </unit>
    </interface>
  </interfaces>
  <vlans>
    <vlan>
      <vlan-id>10</vlan-id>
      <name>prod</name>
      <disable delete="delete"/>
    </vlan>
    <vlan>
      <vlan-id>20</vlan-id>
      <name>dev</name>
      <disable/>
    </vlan>
  </vlans>
</configuration>
```

[ ]:

1.1.8 Advanced topics

The following material is a deep-dive into Yangson, and is not necessarily representative of how one would perform manipulations in a production environment. Please refer to the other tutorials for a better picture of Rosetta's intended use. Keep in mind that the key feature of Yangson is to be able to manipulate YANG data models in a more human-readable format, ala JSON. What lies below digs beneath the higher-level abstractions and should paint a decent picture of the funcitonal nature of Yangson.

1.1.9 Manipulating models with Rosetta and Yangson

One of the goals of many network operators is to provide abstractions in a multi-vendor environment. This can be done with YANG and OpenConfig data models, but as they say, the devil is in the details. It occured to me that you should be able to parse configuration from one vendor and translate it to another. Unfortunately as we all know, these configurations don't always translate well on a 1-to-1 basis. I will demonstrate this process below and show several features of the related libraries along the way.

The following example begins exactly the same as the Cisco parsing tutorial. Let's load up some Juniper config and parse it into a YANG data model. First, we'll read the file.

[1]:
```
from ntc_rosetta import get_driver
import json

junos = get_driver("junos", "openconfig")
junos_driver = junos()
```

(continues on next page)

(continued from previous page)

```
# Strip any rpc tags before and after `<configuration>...</configuration>`
with open("data/junos/dev_conf.xml", "r") as fp:
    config = fp.read()
print(config)
```

```
<configuration>
        <interfaces>
            <interface>
                <name>xe-0/0/1</name>
                <unit>
                    <name>0</name>
                    <family>
                        <ethernet-switching>
                            <interface-mode>access</interface-mode>
                            <vlan>
                                <members>10</members>
                            </vlan>
                        </ethernet-switching>
                    </family>
                </unit>
            </interface>
            <interface>
                <name>xe-0/0/3</name>
                <unit>
                    <name>0</name>
                    <family>
                        <ethernet-switching>
                            <interface-mode>trunk</interface-mode>
                            <vlan>
                                <members>10</members>
                                <members>20</members>
                            </vlan>
                        </ethernet-switching>
                    </family>
                </unit>
            </interface>
            <interface>
                <name>xe-0/0/4</name>
                <unit>
                    <name>0</name>
                    <family>
                        <ethernet-switching>
                            <interface-mode>trunk</interface-mode>
                            <vlan>
                                <members>VLAN-100</members>
                                <members>VLAN-200</members>
                            </vlan>
                        </ethernet-switching>
                    </family>
                </unit>
            </interface>
            <interface>
                <name>xe-0/0/5</name>
                <unit>
                    <name>0</name>
                    <family>
```

(continues on next page)

(continued from previous page)

```
                        <ethernet-switching>
                            <interface-mode>access</interface-mode>
                            <vlan>
                                <members>VLAN-100</members>
                            </vlan>
                        </ethernet-switching>
                    </family>
                </unit>
            </interface>
        </interfaces>
    <vlans>
            <vlan>
                <name>default</name>
                <vlan-id>1</vlan-id>
            </vlan>
            <vlan>
                <name>prod</name>
                <vlan-id>20</vlan-id>
            </vlan>
            <vlan inactive="inactive">
                <vlan-id>10</vlan-id>
            </vlan>
            <vlan>
                <name>VLAN-100</name>
                <vlan-id>100</vlan-id>
            </vlan>
            <vlan>
                <name>VLAN-200</name>
                <vlan-id>200</vlan-id>
            </vlan>
    </vlans>
</configuration>
```

Junos parsing

Now, we parse the config and take a look at the data model.

[158]:
```
from sys import exc_info
from yangson.exceptions import SemanticError

try:
    parsed = junos_driver.parse(
        native={"dev_conf": config},
        validate=False,
        include=[
            "/openconfig-interfaces:interfaces",
            "/openconfig-network-instance:network-instances/network-instance/name",
            "/openconfig-network-instance:network-instances/network-instance/config",
            "/openconfig-network-instance:network-instances/network-instance/vlans",
        ]
    )
except SemanticError as e:
    print(f"error: {e}")
```

(continues on next page)

(continued from previous page)

```
print(json.dumps(parsed.raw_value(), indent=2))
```


```
{
  "openconfig-interfaces:interfaces": {
    "interface": [
      {
        "name": "xe-0/0/1",
        "config": {
          "name": "xe-0/0/1",
          "type": "iana-if-type:ethernetCsmacd",
          "enabled": true
        },
        "subinterfaces": {
          "subinterface": [
            {
              "index": 0,
              "config": {
                "index": 0
              }
            }
          ]
        },
        "openconfig-if-ethernet:ethernet": {
          "openconfig-vlan:switched-vlan": {
            "config": {
              "interface-mode": "ACCESS",
              "access-vlan": 10
            }
          }
        }
      },
      {
        "name": "xe-0/0/3",
        "config": {
          "name": "xe-0/0/3",
          "type": "iana-if-type:ethernetCsmacd",
          "enabled": true
        },
        "subinterfaces": {
          "subinterface": [
            {
              "index": 0,
              "config": {
                "index": 0
              }
            }
          ]
        },
        "openconfig-if-ethernet:ethernet": {
          "openconfig-vlan:switched-vlan": {
            "config": {
              "interface-mode": "TRUNK",
              "trunk-vlans": [
                10,
                20
              ]
            }
```


(continues on next page)

(continued from previous page)

```
        }
      }
    },
    {
      "name": "xe-0/0/4",
      "config": {
        "name": "xe-0/0/4",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": true
      },
      "subinterfaces": {
        "subinterface": [
          {
            "index": 0,
            "config": {
              "index": 0
            }
          }
        ]
      },
      "openconfig-if-ethernet:ethernet": {
        "openconfig-vlan:switched-vlan": {
          "config": {
            "interface-mode": "TRUNK",
            "trunk-vlans": [
              100,
              200
            ]
          }
        }
      }
    },
    {
      "name": "xe-0/0/5",
      "config": {
        "name": "xe-0/0/5",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": true
      },
      "subinterfaces": {
        "subinterface": [
          {
            "index": 0,
            "config": {
              "index": 0
            }
          }
        ]
      },
      "openconfig-if-ethernet:ethernet": {
        "openconfig-vlan:switched-vlan": {
          "config": {
            "interface-mode": "ACCESS",
            "access-vlan": 100
          }
        }
      }
```

(continues on next page)

(continued from previous page)


```
      }
    ]
  },
  "openconfig-network-instance:network-instances": {
    "network-instance": [
      {
        "name": "default",
        "config": {
          "name": "default"
        },
        "vlans": {
          "vlan": [
            {
              "vlan-id": 1,
              "config": {
                "vlan-id": 1,
                "name": "default",
                "status": "ACTIVE"
              }
            },
            {
              "vlan-id": 20,
              "config": {
                "vlan-id": 20,
                "name": "prod",
                "status": "ACTIVE"
              }
            },
            {
              "vlan-id": 10,
              "config": {
                "vlan-id": 10,
                "status": "SUSPENDED"
              }
            },
            {
              "vlan-id": 100,
              "config": {
                "vlan-id": 100,
                "name": "VLAN-100",
                "status": "ACTIVE"
              }
            },
            {
              "vlan-id": 200,
              "config": {
                "vlan-id": 200,
                "name": "VLAN-200",
                "status": "ACTIVE"
              }
            }
          ]
        }
      }
    ]
  }
}
```

Naive translation

Since we have a valid data model, let's see if Rosetta can translate it as-is.

[159]:
```
ios = get_driver("ios", "openconfig")
ios_driver = ios()
native = ios_driver.translate(candidate=parsed.raw_value())

print(native)
```

```
interface xe-0/0/1
   no shutdown
   switchport mode access
   switchport access vlan 10
   exit
!
interface xe-0/0/3
   no shutdown
   switchport mode trunk
   switchport trunk allowed vlan 10,20
   exit
!
interface xe-0/0/4
   no shutdown
   switchport mode trunk
   switchport trunk allowed vlan 100,200
   exit
!
interface xe-0/0/5
   no shutdown
   switchport mode access
   switchport access vlan 100
   exit
!
vlan 1
   name default
   no shutdown
   exit
!
vlan 20
   name prod
   no shutdown
   exit
!
vlan 10
   shutdown
   exit
!
vlan 100
   name VLAN-100
   no shutdown
   exit
!
vlan 200
   name VLAN-200
   no shutdown
   exit
!
```

Pretty cool, right?! Rosetta does a great job of parsing and translating, but it is a case of "monkey see, monkey do". Rosetta doesn't have any mechanisms to translate interface names, for example. It is up to the operator to perform this sort of manipulation.

Down the Yangson rabbit hole

Yangson allows the developer to easily translate between YANG data models and JSON. Most all of these manipulations can be performed on dictionaries in Python and loaded into data models using `from_raw <https://yangson.labs.nic.cz/datamodel.html#yangson.datamodel.DataModel.from_raw>`__. The following examples may appear to be a little obtuse, but the goal is to demontrate the internals of Yangson.

And it's mostly functional

It is critical to read the short description of the zipper interface in the InstanceNode section of the docs. Yanson never manipulates an object, but returns a copy with the manipulated attributes.

Show me the code!

Let's take a look at fixing up the interface names and how we can manipulate data model attributes. To do that, we need to locate the attribute in the tree using the `parse_resource_id <https://yangson.labs.nic.cz/datamodel.html#yangson.datamodel.DataModel.parse_resource_id>`__ method. This method returns an 'instance route'. The string passed to the method is an xpath.

[160]:
```
# Locate the interfaces in the tree. We need to modify this one
# Note that we have to URL-escape the forward slashes per https://tools.ietf.org/html/
↪rfc8040#section-3.5.3
irt = parsed.datamodel.parse_resource_id("openconfig-interfaces:interfaces/
↪interface=xe-0%2F0%2F1")
current_data = parsed.root.goto(irt)
print("Current node configuration: ", json.dumps(current_data.raw_value(), indent=2))
modify_data = current_data.raw_value()
ifname = 'Ethernet0/0/1'
modify_data['name'] = ifname
modify_data['config']['name'] = ifname
stub = current_data.update(modify_data, raw=True)
print("Candidate node configuration: ", json.dumps(stub.raw_value(), indent=2))
```

```
Current node configuration:  {
  "name": "xe-0/0/1",
  "config": {
    "name": "xe-0/0/1",
    "type": "iana-if-type:ethernetCsmacd",
    "enabled": true
  },
  "subinterfaces": {
    "subinterface": [
      {
        "index": 0,
        "config": {
          "index": 0
        }
      }
    ]
  },
```

(continues on next page)

(continued from previous page)

```
  "openconfig-if-ethernet:ethernet": {
    "openconfig-vlan:switched-vlan": {
      "config": {
        "interface-mode": "ACCESS",
        "access-vlan": 10
      }
    }
  }
}
Candidate node configuration:  {
  "name": "Ethernet0/0/1",
  "config": {
    "name": "Ethernet0/0/1",
    "type": "iana-if-type:ethernetCsmacd",
    "enabled": true
  },
  "subinterfaces": {
    "subinterface": [
      {
        "index": 0,
        "config": {
          "index": 0
        }
      }
    ]
  },
  "openconfig-if-ethernet:ethernet": {
    "openconfig-vlan:switched-vlan": {
      "config": {
        "interface-mode": "ACCESS",
        "access-vlan": 10
      }
    }
  }
}
```

Instance routes

You will notice a `goto` method on child nodes. You *can* access successors with this method, but you have to build the path from the root `datamodel` attribute as seen in the following example. If you aren't sure where an object is in the tree, you can also rely on its `path` attribute.

Quick tangent... what is the difference between `parse_instance_id` and `parse_resource_id`? The answer can be found in the Yangson glossary and the respective RFC's.

[161]:
```
# TL;DR
irt = parsed.datamodel.parse_instance_id('/openconfig-network-instance:network-
↪instances/network-instance[1]/vlans/vlan[3]')
print(parsed.root.goto(irt).raw_value())

irt = parsed.datamodel.parse_resource_id('openconfig-network-instance:network-
↪instances/network-instance=default/vlans/vlan=10')
print(parsed.root.goto(irt).raw_value())
```

```
{'vlan-id': 10, 'config': {'vlan-id': 10, 'status': 'SUSPENDED'}}
{'vlan-id': 10, 'config': {'vlan-id': 10, 'status': 'SUSPENDED'}}
```

What about the rest of the interfaces in the list? Yangson provides an iterator for array nodes.

[162]:
```
import re

irt = parsed.datamodel.parse_resource_id("openconfig-interfaces:interfaces/interface")
iface_objs = parsed.root.goto(irt)
# Swap the name as required
p, sub = re.compile(r'xe-'), 'Ethernet'

# There are a couple challenges here.  First is that Yanson doesn't impliment __len__
# The second problem is that you cannot modify a list in-place, so we're basically
# hacking this to hijack the index of the current element and looking it up from a
↪"clean"
# instance.  This is a pet example!  It would be much easier using Python dicts.
new_ifaces = None
for iface in iface_objs:
    name_irt = parsed.datamodel.parse_instance_id('/name')
    cname_irt = parsed.datamodel.parse_instance_id('/config/name')
    if new_ifaces:
        name = new_ifaces[iface.index].goto(name_irt)
    else:
        name = iface.goto(name_irt)
    name = name.update(p.sub(sub, name.raw_value()), raw=True)
    cname = name.up().goto(cname_irt)
    cname = cname.update(p.sub(sub, cname.raw_value()), raw=True)
    iface = cname.up().up()
    new_ifaces = iface.up()
print(json.dumps(new_ifaces.raw_value(), indent=2))
```


```
[
  {
    "subinterfaces": {
      "subinterface": [
        {
          "index": 0,
          "config": {
            "index": 0
          }
        }
      ]
    },
    "openconfig-if-ethernet:ethernet": {
      "openconfig-vlan:switched-vlan": {
        "config": {
          "interface-mode": "ACCESS",
          "access-vlan": 10
        }
      }
    },
    "name": "Ethernet0/0/1",
    "config": {
      "type": "iana-if-type:ethernetCsmacd",
      "enabled": true,
      "name": "Ethernet0/0/1"
    }
  },
  {
    "subinterfaces": {
```


(continues on next page)

(continued from previous page)

```
      "subinterface": [
        {
          "index": 0,
          "config": {
            "index": 0
          }
        }
      ]
    },
    "openconfig-if-ethernet:ethernet": {
      "openconfig-vlan:switched-vlan": {
        "config": {
          "interface-mode": "TRUNK",
          "trunk-vlans": [
            10,
            20
          ]
        }
      }
    },
    "name": "Ethernet0/0/3",
    "config": {
      "type": "iana-if-type:ethernetCsmacd",
      "enabled": true,
      "name": "Ethernet0/0/3"
    }
  },
  {
    "subinterfaces": {
      "subinterface": [
        {
          "index": 0,
          "config": {
            "index": 0
          }
        }
      ]
    },
    "openconfig-if-ethernet:ethernet": {
      "openconfig-vlan:switched-vlan": {
        "config": {
          "interface-mode": "TRUNK",
          "trunk-vlans": [
            100,
            200
          ]
        }
      }
    },
    "name": "Ethernet0/0/4",
    "config": {
      "type": "iana-if-type:ethernetCsmacd",
      "enabled": true,
      "name": "Ethernet0/0/4"
    }
  },
  {
```

(continues on next page)

(continued from previous page)

```
    "subinterfaces": {
      "subinterface": [
        {
          "index": 0,
          "config": {
            "index": 0
          }
        }
      ]
    },
    "openconfig-if-ethernet:ethernet": {
      "openconfig-vlan:switched-vlan": {
        "config": {
          "interface-mode": "ACCESS",
          "access-vlan": 100
        }
      }
    },
    "name": "Ethernet0/0/5",
    "config": {
      "type": "iana-if-type:ethernetCsmacd",
      "enabled": true,
      "name": "Ethernet0/0/5"
    }
  }
]
```

[163]:
```
# Translate to Cisco-speak
native = ios_driver.translate(candidate=new_ifaces.top().raw_value())

print(native)
```

```
interface Ethernet0/0/1
   no shutdown
   switchport mode access
   switchport access vlan 10
   exit
!
interface Ethernet0/0/3
   no shutdown
   switchport mode trunk
   switchport trunk allowed vlan 10,20
   exit
!
interface Ethernet0/0/4
   no shutdown
   switchport mode trunk
   switchport trunk allowed vlan 100,200
   exit
!
interface Ethernet0/0/5
   no shutdown
   switchport mode access
   switchport access vlan 100
   exit
!
vlan 1
```

(continues on next page)

(continued from previous page)

```
   name default
   no shutdown
   exit
!
vlan 20
   name prod
   no shutdown
   exit
!
vlan 10
   shutdown
   exit
!
vlan 100
   name VLAN-100
   no shutdown
   exit
!
vlan 200
   name VLAN-200
   no shutdown
   exit
!
```

Hooray! That should work. One final approach, just to show you different ways of doing things. This is another pet example to demonstrate Yangson methods.

[164]:
```
import re
from typing import Dict

irt = parsed.datamodel.parse_resource_id("openconfig-interfaces:interfaces")
iface_objs = parsed.root.goto(irt)
# Nuke the whole branch!
iface_objs = iface_objs.delete_item("interface")

def build_iface(data: str) -> Dict:
    # Example template, this could be anything you like that conforms to the schema
    return {
        "name": f"{data['name']}",
        "config": {
            "name": f"{data['name']}",
            "description": f"{data['description']}",
            "type": "iana-if-type:ethernetCsmacd",
            "enabled": True
        },
    }

iface_data = [
    build_iface({
        "name": f"TenGigabitEthernet0/{idx}",
        "description": f"This is interface TenGigabitEthernet0/{idx}"
    }) for idx in range(10, 0, -1)
]

initial = iface_data.pop()
```

(continues on next page)

(continued from previous page)

```
# Start a new interface list
iface_objs = iface_objs.put_member("interface", [initial], raw=True)
cur_obj = iface_objs[0]

# Yangson exposes `next`, `insert_after`, and `insert_before` methods.
# There is no `append`.
while iface_data:
    new_obj = cur_obj.insert_after(iface_data.pop(), raw=True)
    cur_obj = new_obj
```

[165]:
```
# Translate to Cisco-speak
native = ios_driver.translate(candidate=cur_obj.top().raw_value())

print(native)
```

```
interface TenGigabitEthernet0/1
   description This is interface TenGigabitEthernet0/1
   no shutdown
   exit
!
interface TenGigabitEthernet0/2
   description This is interface TenGigabitEthernet0/2
   no shutdown
   exit
!
interface TenGigabitEthernet0/3
   description This is interface TenGigabitEthernet0/3
   no shutdown
   exit
!
interface TenGigabitEthernet0/4
   description This is interface TenGigabitEthernet0/4
   no shutdown
   exit
!
interface TenGigabitEthernet0/5
   description This is interface TenGigabitEthernet0/5
   no shutdown
   exit
!
interface TenGigabitEthernet0/6
   description This is interface TenGigabitEthernet0/6
   no shutdown
   exit
!
interface TenGigabitEthernet0/7
   description This is interface TenGigabitEthernet0/7
   no shutdown
   exit
!
interface TenGigabitEthernet0/8
   description This is interface TenGigabitEthernet0/8
   no shutdown
   exit
!
interface TenGigabitEthernet0/9
   description This is interface TenGigabitEthernet0/9
```

(continues on next page)

(continued from previous page)

```
   no shutdown
   exit
!
interface TenGigabitEthernet0/10
   description This is interface TenGigabitEthernet0/10
   no shutdown
   exit
!
vlan 1
   name default
   no shutdown
   exit
!
vlan 20
   name prod
   no shutdown
   exit
!
vlan 10
   shutdown
   exit
!
vlan 100
   name VLAN-100
   no shutdown
   exit
!
vlan 200
   name VLAN-200
   no shutdown
   exit
!
```

Deleting individual items

Here is an example of deleting an individual item. Navigating the tree can be a bit tricky, but it's not too bad once you get the hang of it.

[166]:
```
# Locate a vlan by ID and delete it
irt = parsed.datamodel.parse_resource_id("openconfig-network-instance:network-
↪instances/network-instance=default/vlans/vlan=10")
vlan10 = parsed.root.goto(irt)
vlans = vlan10.up().delete_item(vlan10.index)
print(json.dumps(vlans.raw_value(), indent=2))
```

```
[
  {
    "vlan-id": 1,
    "config": {
      "vlan-id": 1,
      "name": "default",
      "status": "ACTIVE"
    }
  },
```

(continues on next page)

(continued from previous page)

```
  {
    "vlan-id": 20,
    "config": {
      "vlan-id": 20,
      "name": "prod",
      "status": "ACTIVE"
    }
  },
  {
    "vlan-id": 100,
    "config": {
      "vlan-id": 100,
      "name": "VLAN-100",
      "status": "ACTIVE"
    }
  },
  {
    "vlan-id": 200,
    "config": {
      "vlan-id": 200,
      "name": "VLAN-200",
      "status": "ACTIVE"
    }
  }
]
```

1.2 Models

1.2.1 Openconfig

Official documentation.

1.2.2 ntc-yang-models

Official documentation.

1.2.3 Matrix

ntc parser

ntc translator

openconfig parser

openconfig translator

1.3 CLI

ntc_rosetta comes with a simple CLI tool to help with various things:

1. Lint parsers/translators
2. Print the supported models as an ASCII tree
3. Print the parser/translator as an ASCII tree similar to the supported models output

To execute the command line tool type `ntc_rosetta` after installing the library:

```
$ ntc_rosetta git:(docs)  ntc_rosetta
Usage: ntc_rosetta [OPTIONS] COMMAND [ARGS]...

Options:
  --help  Show this message and exit.

Commands:
  lint          Lint files/folders with Parsers/Translators: Errors/Warning...
  print-model   Prints the model as an ASCII tree
  print-parser  Prints a tree representation of a parser/translator.
```

The command line tool is self-documented and you can check it's documentation using the `--help` flag.

1.4 API

1.4.1 drivers

class ntc_rosetta.drivers.base.**Driver**

Parent class used to operate on models and native datastructures

The class has a few class attributes that need to be overriden by classes inheriting from this class.

parser

Class attribute to defines which `yangify.parser.RootParser` to use when parsing native data

translator

Class attribute to defines which `yangify.translator.RootParser` to use when translating YANG models to native data

datamodel

Class attribute that defines which `yangson.datamodel.DataModel` the parser and translator can operate on.

datamodel_name = ''

classmethod get_datamodel() → yangson.datamodel.DataModel

merge(*candidate: Dict[str, Any]*, *running: Dict[str, Any]*, *replace: bool = False*) → Any

Given two objects conforming to a data model compute the needed native commands to converge the configurations.

Parameters

- **candidate** – Desired configuration (must conform to the datamodel of the driver)
- **running** – Original configuration (must conform to the datamodel of the driver)
- **replace** – Whether to return a list of commands that reinitializes blocks (i.e. defaulting an interface)

parse(*native: Optional[Dict[str, Any]] = None*, *validate: bool = True*, *include: Optional[List[str]] = None*, *exclude: Optional[List[str]] = None*) → ntc_rosetta.drivers.base.ParseResult

Parser native data and maps it into an object conforming to the datamodel.

Parameters

- **native** – native data extracted from a device
- **validate** – whether to validate the model or not
- **include** – if specified only return data matching the YANG paths here
- **excldue** – if specify exclude data matching the YANG paths here

parser
alias of `yangify.parser.RootParser`

translate(*candidate: Dict[str, Any], replace: bool = False*) → Any
Translates data conforming to the model into native configuration the device understands.

Parameters

- **candidate** – Data to translate (must conform to the datamodel of the driver)
- **replace** – Whether to return a list of commands that reinitializes blocks (i.e. defaulting an interface)

translator
alias of `yangify.translator.RootTranslator`

class ntc_rosetta.drivers.base.**ParseResult**(*root: yangson.instance.RootNode*, *datamodel: yangson.datamodel.DataModel*)
Result of parsing a configuration

root
Root of the response

datamodel
Datamodel related to the parsed object

peek(*path: str*) → Union[int, decimal.Decimal, str, Tuple[None], ArrayValue, ObjectValue, None]
Return the value in the given path (YANG)

raw_value() → Union[bool, int, str, List[None], Dict[str, Union[bool, int, str, List[None], Dict[str, RawValue], List[Dict[str, RawValue]], List[Union[bool, int, str, List[None]]]]], List[Dict[str, Union[bool, int, str, List[None], Dict[str, RawValue], List[Dict[str, RawValue]], List[Union[bool, int, str, List[None]]]]]], List[Union[bool, int, str, List[None]]]]
Parsed data in python's native datastructures (dicts, lists, strings, etc)

1.4.2 helpers

ntc_rosetta.helpers.json_helpers.**query**(*query: str, data: Dict[str, Any], force_list: bool = False, default: Optional[Any] = None*) → Any
Query a nested dictionary using *jmespath <http://jmespath.org>*

Parameters

- **query** – jmespath query
- **data** – data to query
- **force_list** – return a list even if the object is not a list
- **default** – return this if the query returns None

ntc_rosetta.helpers.xml_helpers.**find_or_create**(*root: lxml.etree.Element*, *xpath: str*) → lxml.etree.Element
Return a subelement or create if it doesn't exist.

Parameters

- **root** – Root of the object
- **xpath** – xpath to apply to the root object

1.4.3 ntc_rosetta

ntc_rosetta.**get_driver**(*driver: str*, *model: str = 'openconfig'*) → Type[ntc_rosetta.drivers.base.Driver]
Returns a driver class to operate on a given device OS/model pair

Parameters **model** – Which model to retrieve, currently suppporting "openconfig" and "ntc"

1.4.4 yang

ntc_rosetta.yang.**get_data_model**(*model: str = 'openconfig'*) → yangson.datamodel.DataModel
Returns an instantiated data model.

1.5 CHANGELOG

1.5.1 0.1.0

- first version

1.6 CONTRIBUTING

All contributions are welcome and even though there are no strict or formal requirements to contribute there are some basic rules contributors need to follow:

1. Make sure your contribution is original and it doesn't violate anybody's copyright
2. Make sure tests pass
3. Make sure your contribution is tested

Below you can find more information depending on what you are trying to contribute, in case of doubt don't hesitate to open a PR with your contribution and ask for help.

1.6.1 Running tests

To run tests you need `docker` and `GNU Make`. If you meet the requirements all you need to do is execute `make tests`. All the tests will run inside docker containers you don't need to worry about the environment.

1.6.2 Adding documentation

If you want to contribute documentation you need to be sligthly familiar with sphinx as that's the framework used in this project (and most python projects) to build the documentation.

In addition, if you want to contribute with tutorials or code examples you need to be familiar with jupyter. The advantage of using jupyter notebooks over just plain text is that notebooks can be tested. This means code examples and tutorials will be tested by the CI and ensure they stay relevant and work.

The easiest way of working with jupyter is by executing `make jupyter` in your local machine and pointing your browser to http://localhost:8888/notebooks/docs. If you are adding a new notebook don't forget to add it to sphinx's documentation.

1.6.3 Coding Style

We use black to format the code.

1.6.4 Adding new features

New features need to come with tests and a tutorial in the form of a jupyter notebook so it can be tested.

1.6.5 Contributing to drivers

If you are adding or doing changes to drivers you need to ensure:

1. Existing tests pass without modifying them (unless warranted)
2. You add relevant tests to ensure future contributors don't break your contribution unknowingly

In this type of contribution you don't need to change the tests, you only need to add one or more test case that covers your changes. To do so you will need to provide a directory per test case with a few files. For instance:

1. To test parsers
    - `dev_conf` - Native configuration
    - `result.json` - Expected result after parsing the configuration
2. To test translators
    - `data.json` - Configuration in structured format you want to translate
    - `res_merge` - Native representation in "merge" mode
    - `res_replace` - Native representation in "replace" mode
3. To test merge operations
    - `data_candidate.json` - Candidate configuration in structured format
    - `data_running.json` - Running configuration in structured format
    - `res_merge` - The expected commands the device will need to execute to go from the running configuration to the candidate one in "merge" mode.
    - `res_replace` - The expected commands the device will need to execute to go from the running configuration to the candidate one in "replace" mode.

1.6.6 mypy

We use mypy to bring static typing to our code. This adds some complexity but results in cleaner, less error-prone and more understandable code.

Indices and tables

- genindex
- modindex
- search

Python Module Index

n

Index