Using the YANG Development Kit (YDK) with Cisco IOS XE

1. Overview

The YANG Development Kit (YDK) is a software development kit that provides APIs that are generated from YANG data models. These APIs, Python and C++ language bindings, can be downloaded and installed on your local machine to simplify application development and implementation. But how do these APIs make your life easier?

Model driven: You have no need to learn new data abstractions or hierarchies. YDK mirrors the structure of your YANG data models. If you become familiar with the structure of a particular model, you will feel right at home using the API for that model. There’s no need to handle YANG files directly.

Built-in data validation: YDK takes care of data validation so you don’t have to. Data models not only define a data hierarchy, but also specify the constraints associated with the data (type, valid values, ranges, etc.) YDK services automatically perform thorough data validation (types, values, semantics, deviations, etc.) locally before your data is exchanged with the network device.

Protocol, transport, and encoding support: You are not required to code the specifics of a management protocol (such as NETCONF RPCs) or to manipulate encoded data directly (for example, XML or JSON). A set of predefined services takes care of the management protocol, the transport for your management session, and the encoding and decoding of your data. Your automation code can focus on the details of the data exchanged with the device and the automation logic.
2. Model driven programmability stack

YDK uses SSH or HTTP as the transport protocol and NETCONF or RESTCONF as the protocol between client and server. It uses XML or JSON for encoding. Figure 1 shows YDK’s position in the programmability stack. The YANG models in the model layer can be native or open, and all the apps in the application layer are written by end developers using YDK APIs.

Figure 1. Programmability stack

3. Services and providers

YDK provides a couple of abstractions to help developers build their application with the generated APIs:

- **Services** provide a simple API interface to be used with the bindings and perform operations on model objects.
- **Providers** implement the underlying protocol details, such as the NetConfServiceProvider and the CodecServiceProvider.

In the example below, we import CRUDService from ydk.services and NetconfServiceProvider from ydk.providers. We also instantiate an instance of the provider that allows users to type the device information to create a NETCONF session using NetconfServiceProvider.

```python
from ydk.services import CRUDService
from ydk.providers import NetconfServiceProvider

# create NETCONF provider
provider = NetconfServiceProvider(address=device.hostname,
port=device.port,
username=device.username,
password=device.password,
protocol=device.scheme
```
CRUD service is one of the core services that YDK provides to create, read, update, and delete the configurational and operational data. While all operations can be invoked on configuration data, only read operations can be invoked on operational data.

First we need to import the CRUDService class, and next we instantiate the CRUDService. Then we invoke the create method of the CRUDService class, passing in the service provider instance and the new entity.

```python
from ydk.services import CRUDService
crud = CRUDService()
crud.create(provider, name)
```

Provider is an instance of ydk.providers.ServiceProvider, and name is an example of a user-defined entity.

4. App development

In this section two Cisco IOS® XE apps are created using YDK. The first app is a script to configure a hostname on a device, and the second app configures an IP address under a Gigabit Ethernet interface, followed by an explanation of how the app will be created.

4.1 First app – configure a hostname

```bash
#!/usr/bin/env python
#
# Copyright 2016 Cisco Systems, Inc.
#
# Licensed under the Apache License, Version 2.0 (the “License”);
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an “AS IS” BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
""
Create configuration for model Cisco-IOS-XE-native.


positional arguments:
  device NETCONF device (ssh://user:password@host:port)

optional arguments:
  -h, --help       show this help message and exit
  -v, --verbose    print debugging messages
""
```
from ydk.services import CRUDService
from ydk.providers import NetconfServiceProvider
from ydk.models.cisco_ios_xe import Cisco_IOS_XE_native
    as xe_native

if __name__ == "__main__":
    """Execute main program."""

    # create NETCONF provider
    provider = NetconfServiceProvider(address="172.26.198.36",
                                        port=830,
                                        username="cisco",
                                        password="cisco",
                                        protocol="ssh")

    # create CRUD service
    crud = CRUDService()
    native = xe_native.Native()  # create object

    native.hostname = "Access_Switch"

    # create configuration on NETCONF device
    crud.create(provider, native)

    provider.close()
    exit()

# End of script

Services and providers were already covered in a previous section. We expand on the previous example by looking in more detail at how to use other parts of the data models:

• Create an object ‘native’ and assign it to xe_native.Native(). At the start of the app, we imported Cisco_IOS_XE_native as xe_native.
• Hostname is one of the attributes under Cisco_IOS_XE_native.Native. It is a string type, as shown in Figure 3. You can find this class in ydk.cisco.com for the XE platform as shown in Figure 2.

Figure 2. Documentation for the native class
• Next, assign `native.hostname` to the name of the switch. Here it is Access_Switch.
• Then we use the CRUDService crud.create(provider, native) to configure the device using NETCONF.
• When we run this app, NETCONF will open a session with the networking device and configure the hostname as Access_Switch.
• After that, `provider.close()` will close the NETCONF session.

```python
native = xe_native.Native()  # create object
    native.hostname = "Access_Switch"
    # create configuration on NETCONF device
crud.create(provider, native)

provider.close()
```

4.1.1 App execution

Configuration of the switch before running the app is as follows:

```
Switch#sh run | inc hostname
hostname Switch
Switch#
```

`config_hostname.py` is the name of the app. You can execute that app as follows:

```bash
cisco@U-Disk-Scripts$ ./config_hostname.py
cisco@U-Disk-Scripts$
```

Below is the configuration of the switch after running the app. You can see that the hostname of the switch has changed to Access_Switch.

```
Access_Switch#sh run | inc hostname
hostname Access_Switch
Access_Switch#
```

4.2 Second app – configure IP address

In the second app we are adding logging and parser commands from the first app. This app will configure an IP address for the interface GigabitEthernet 2/0/10 and a description of Connected_to_Core_Switch. The use of parsers and logging will be explained after the example. The interface configuration on the switch is also shown before and after the execution of the script.
#!/usr/bin/env python
#
# Copyright 2016 Cisco Systems, Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

# Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an “AS IS” BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
#
"

Create configuration for model Cisco-IOS-XE-native.


positional arguments:
    device         NETCONF device (ssh://user:password@host:port)

optional arguments:
    -h, --help     show this help message and exit
    --verbose      print debugging messages

"

from argparse import ArgumentParser
from urlparse import urlparse
from ydk.types import Empty
from ydk.services import CRUDService
from ydk.providers import NetconfServiceProvider
from ydk.models.cisco_ios_xe import Cisco_IOS_XE_native \   
    as xe_native
import logging

def config(interface):
    """Add config data to the object."""

gigabitethernet = interface.Gigabitethernet()
gigabitethernet.name = “2/0/10”
gigabitethernet.description = “Connected_to_Core_Switch”
gigabitethernet.ip.address.primary.address = “15.10.1.1”
gigabitethernet.ip.address.primary.mask = “255.255.255.0”
ip_add.gigabitethernet.append(gigabitethernet)
To configure an IP address and description on an interface, we do the following:

- Create an object called interface, assign it to `xe_native.Native.Interface()`, and then call the function `config(interface)`.
- In the function `config(interface)`, we assign and then set the name and description as `gigabitethernet.name` and `gigabitethernet.description`. The name and description are leafs and are defined under gigabitethernet. Documentation for this structure can be found at ydk.cisco.com, as shown in Figure 4.
• ‘address’ and ‘mask’ are the leaves under the class `gigabitethernet.ip.address.primary`. Documentation for this structure can be found at ydk.cisco.com, as shown in Figure 5.

**Figure 4.** Documentation for the Native.Interface.Gigabitethernet class

```
class Native.Interface.Gigabitethernet:
    Bases: object
    Class Hierarchy: Native Interface Gigabitethernet

This class represents configuration data.
GigabitEthernet IEEE 802.3z
    name <key>
    type: str
```

• Assign the IP address of `gigabitethernet.ip.address.primary.address` as 15.10.1.1 and the mask `gigabitethernet.ip.address.primary.mask` as 255.255.255.0.

• In the last line we append gigabitethernet because it is a list in the YANG model under the /native/interface container, as shown below.

```python
def config(interface):
    """Add config data to the object."""
```
gigabitethernet = interface.Gigabitethernet()
gigabitethernet.name = "2/0/10"
gigabitethernet.description = "Connected_to_Core_Switch"
gigabitethernet.ip.address.primary.address = "15.10.1.1"
gigabitethernet.ip.address.primary.mask = "255.255.255.0"
ip_add.gigabitethernet.append(gigabitethernet)

interface = xe_native.Native.Interface()  # create object
config(interface)  # add object configuration

# create configuration on NETCONF device
crud.create(provider, interface)

This section explains the logging and parsers of the main function. The commands under the main function are used for parsing. In the first app example, we gave device details in the provider section. But in this app we are giving device details while executing the app. This way of giving device details provides the advantage of being able to run the same app on multiple devices without changing any provider details from the app.

if __name__ == "__main__":
    """Execute main program."""
    parser = ArgumentParser()
    parser.add_argument("-v", "--verbose", help="print debugging messages",
                        action="store_true")
    parser.add_argument("device",
                        help="NETCONF device (ssh://user:password@host:port")")
    args = parser.parse_args()
    device = urlparse(args.device)

YDK uses common Python logging. All modules are based on the YDK log. When logging is enabled while running a app, you can see the details of the logs that are sent to the client from the server. These log messages also show NETCONF connection creation, termination, and the YANG model that was sent to the device.

import logging

logger = logging.getLogger("ydk")
logger.setLevel(logging.DEBUG)
handler = logging.StreamHandler()
formatter = logging.Formatter("%(asctime)s - %(name)s - %(levelname)s - %(message)s")
handler.setFormatter(formatter)
logger.addHandler(handler)

All the logging for YDK will be used from the Python logging facility. Below are a few examples of classes with their description:

Loggers expose the interface that an app uses directly.
Handlers send the log records to the appropriate destination.
Formatters specify the layout of log records in the final output.

More details about Python logging can be found at the following link:

https://docs.python.org/3/library/logging.html
4.2.1 App execution

Here is the interface configuration before running the app:

```
Access_Switch#sh run int gi 2/0/10
Building configuration...
```

Current configuration: 69 bytes

```
! interface GigabitEthernet2/0/10
   no switchport
   no ip address
end
```

There are two ways to provide device details such as username, password, and IP address. The first app shows how to specify the device details in the provider section. This second app shows how to specify the device details during execution, thereby enabling the same app to be used on multiple devices.

```
cisco@U-Disk-Scripts$ ./config_ip.py ssh://cisco:cisco@172.26.198.36 -v
2017-03-06 09:55:11,017 - ydk.providers.netconf_provider - INFO - NetconfServiceProvider connected to 172.26.198.36:None using ssh
2017-03-06 09:55:12,106 - ydk.services.crud_service - INFO - CREATE operation initiated
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:a56d21ff-e32b-4ab8-9114-06ff8a0blfea">
  <edit-config>
    <target>
      <running/>
    </target>
    <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
        <interface>
          <GigabitEthernet>
            <name>2/0/10</name>
            <description>Connected_to_Core_Switch</description>
            <ip>
              <primary>
                <address>15.10.1.1</address>
                <mask>255.255.255.0</mask>
              </primary>
            </ip>
          </GigabitEthernet>
        </interface>
      </native>
    </config>
  </edit-config>
</rpc>
```

```
2017-03-06 09:55:12,573 - ydk.providers._provider_plugin - DEBUG -
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:a56d21ff-e32b-4ab8-9114-06ff8a0blfea">
  <ok/>
</rpc-reply>
```
Here is the interface configuration after running the app:

```
Access_Switch#sh run int gi 2/0/10
Building configuration...

Current configuration: 111 bytes
!
interface GigabitEthernet2/0/10
  description Connected_to_Core_Switch
  no switchport
  ip address 15.10.1.1 255.255.255.0
end
```

5. Installation of YDK

YDK can be installed on Cisco IOS XE platforms by using the command “sudo pip install ydk-models-cisco-ios-xe” on Mac or Linux. For more detailed information, please see the following links:

- https://github.com/CiscoDevNet/ydk-py
- https://ydk.cisco.com/py/docs/

6. Conclusion

The two apps shown in this white paper are simple examples of how to start using YDK. The real power of YDK is that it allows you to do this for any YANG model on the device, automatically generating Python classes that inherit the syntactic checks and requirements of the underlying model while also handling all the details of the underlying encoding and transport.

7. References

- ydk.cisco.com
- https://developer.cisco.com/site/ydk/
- http://github/CiscoDevNet/ydk-py-samples