YANG Development Kit Documentation

Release 0.2.0

Cisco Systems, Inc

June 19, 2017

Contents

CHAPTER 1

Contents:

About YDK

A Software Development Kit that provides API's that are modeled in YANG.

License

Version

Version : version-id

This ydk-py is generated using ydk-gen.

To check out the version of ydk-gen used to generate this ydk-py, use the below commands:

```
git clone repo-url
git checkout commit-id
```

Changelog

- Changelog

Getting Started

Table of Contents

Overview

The YANG Development Kit (YDK) is a Software Development Kit that provides API's that are modeled in YANG. The main goal of YDK is to reduce the learning curve of YANG data models by expressing the model semantics in an API and abstracting protocol/encoding details. YDK is composed of a core package that defines services and providers, plus one or more module bundles that are based on YANG models.

System Requirements

Linux

Ubuntu (Debian-based) - The following packages must be present in your system before installing YDK-Py:

```
$ sudo apt-get install python-pip zlib1g-dev python-lxml libxml2-dev libxslt1-dev python-dev
```

Centos (Fedora-based) - The following packages must be present in your system before installing YDK-Py:

```
$ sudo yum install epel-release
$ sudo yum install python-pip python-devel libxml2-devel libxslt-devel libcurl-devel libtool
```

macOS

It is required to install Xcode command line tools, homebrew and the following homebrew packages on your system before installing YDK-Py:

```
$ xcode-select --install
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
$ brew install python pkg-config libssh xml2 curl pcre
```

Windows

You must install the following requirements::

- Python Releases for Windows

- Visual C++ Build Tools

Python Requirements

Both Python 2 and 3 are supported. At least, Python2.7 or Python 3.4 must be installed in your system.

How to install

Quick Install

You can install the latest model packages from the Python package index. Note that, in some systems, you need to install the new package as root. You get a fully operational YDK environment by installing the `cisco-ios-xr` bundle which automatically installs all other YDK-related packages (`ydk`, `cisco-ios-xr`, `openconfig` and `ietf` packages):

```
$ pip install ydk-models-cisco-ios-xr
```

Alternatively, you can perform a partial installation. If you only want to install the `openconfig` bundle and its dependencies (`ydk` and `ietf` packages), execute:

```
$ pip install ydk-models-openconfig
```

If you only want to install the `ietf` bundle and its dependencies (`ydk` package), execute:

```
$ pip install ydk-models-ietf
```

Installing from Source

If you prefer not to use the YDK packages in the Python package index, you need to install manually the `ydk` core package and then the model bundles you plan to use. To install the `ydk` core package, execute:

```
$ cd core
core$ python setup.py sdist
core$ pip install dist/ydk*.gz
```

Once you have installed the `ydk` core package, you can install one more model bundles. Note that some bundles have dependencies on other bundles. Those dependencies are already captured in the bundle package. Make sure you install the desired bundles in the order below. To install the `ietf` bundle, execute:

```
core$ cd ../ietf
ietf$ python setup.py sdist
ietf$ pip install dist/ydk*.gz
```

To install the `openconfig` bundle, execute:

```
ietf$ cd ../openconfig
openconfig$ python setup.py sdist
openconfig$ pip install dist/ydk*.gz
```

To install the `cisco-ios-xr` bundle, execute:

```
openconfig$ cd ../cisco-ios-xr
cisco-ios-xr$ python setup.py sdist
cisco-ios-xr$ pip install dist/ydk*.gz
cisco-ios-xr$ cd ..
```

Using a Virtual Environment

You may want to perform the installation under a Python virtual environment (virtualenv/virtualenvwrapper). A virtual environment allows you to install multiple versions of YDK if needed. In addition, it prevents any potential conflicts between package dependencies in your system.

To install virtual environment support in your system, execute:

```
$ pip install virtualenv virtualenvwrapper
$ source /usr/local/bin/virtualenvwrapper.sh
```

In some systems (e.g. Debian-based Linux), you need to install support for Python virtual environments as root:

```
$ sudo pip install virtualenv virtualenvwrapper
$ source /usr/local/bin/virtualenvwrapper.sh
```

Create a new virtual environment:

```
$ mkvirtualenv -p python2.7 ydk-py
```

At this point, you can perform the quick install or the installation from source described above. Take into account that must not attempt to install YDK as root under a virtual environment.

Samples

To get started with using the YDK API, there are sample apps available in the YDK-Py repository. For example, to run the `bgp.py` sample, execute:

```
(ydk-py)ydk-py$ cd core/samples
(ydk-py)samples$ ./bgp.py -h
Usage: bgp.py [-h | --help] [options]

Options:
-h, --help            show this help message and exit
-v VERSION, --version=VERSION
                      force NETCONF version 1.0 or 1.1
-u USERNAME, --user=USERNAME
-p PASSWORD, --password=PASSWORD
                      password
--proto=PROTO         Which transport protocol to use, one of ssh or tcp
--host=HOST           NETCONF agent hostname
--port=PORT           NETCONF agent SSH port

(ydk-py)samples$ ./bgp.py --host <ip-address-of-netconf-server> -u <username> -p <password> --port <p
```

Documentation and Support

- Hundreds of samples can be found in the YDK-Py samples repository
- Join the YDK community to connect with other users and with the makers of YDK

Developer Guide

Introduction

Table of Contents

YDK consists of two main components: core library, which consists of services and providers, and python model API, which are APIs generated based on YANG models and packaged as bundles.

Core library consists of the below:

- **Service:** Provides simple API interface to be used with the bindings and providers
- **ServiceProvider:** Provides concrete implementation that abstracts underlying protocol details (e.g. `NetconfServiceProvider`, which is based on the NETCONF protocol)

Applications can be written using the python model API in conjunction with a service and a provider.

Writing an app

In this example, we set some BGP configuration using the OpenConfig model, the CRUD (Create/Read/Update/Delete) service and the NETCONF service provider. The example in this document is a simplified version of the more complete sample that is available in `samples/bgp.py`. That more complete sample can be run with the below steps:

```
(ydk-py)ydk-py$ cd core/samples
(ydk-py)samples$ ./bgp.py -h
Usage: bgp.py [-h | --help] [options]

Options:
-h, --help            show this help message and exit
-v VERSION, --version=VERSION
                      force NETCONF version 1.0 or 1.1
-u USERNAME, --user=USERNAME
-p PASSWORD, --password=PASSWORD
                      password
--proto=PROTO         Which transport protocol to use, one of ssh or tcp
--host=HOST           NETCONF agent hostname
--port=PORT           NETCONF agent SSH port

(ydk-py)samples$ ./bgp.py --host <ip-address-of-netconf-server> -u <username> -p <password> --port <p
```

What happens underneath

YDK performs the below actions when running this application:

1. Establish a session with the device
2. Encode python data objects to the protocol format (e.g. netconf XML payload)
3. Perform transport operation with the device and collect the response (e.g. netconf reply)
4. Decode response as python object and return the result to app

5. Raise python exceptions for any errors that occurred

Service Providers

The first step in any application is to create a service provider instance. In this case, the NETCONF service provider (defined in `ydk.providers.NetconfServiceProvider`) is responsible for mapping between the CRUD service API and the underlying manageability protocol (NETCONF RPCs).

We instantiate an instance of the service provider that creates a NETCONF session to the machine with address 10.0.0.1

```
from ydk.providers import NetconfServiceProvider

sp_instance = NetconfServiceProvider(address='10.0.0.1',
                                     port=830,
                                     username='test',
                                     password='test',
                                     protocol='ssh')
```

Using the model APIs

After establishing the connection, we instantiate the entities and set some data. First, we import the types from the OpenConfig BGP module:

```
from ydk.models.openconfig import openconfig_bgp
from ydk.models.openconfig import openconfig_bgp_types
```

Next, create a `BGP` configuration object and set the attributes:

```
# create BGP object
bgp_cfg = openconfig_bgp.Bgp()

# set the Global AS
bgp_cfg.global_.config.as_ = 65001

# Create an AFI SAFI config
ipv4_afsf = bgp_cfg.global_.afi_safis.AfiSafi()
ipv4_afsf.afi_safi_name = openconfig_bgp_types.Ipv4Unicast()
ipv4_afsf.config.afi_safi_name = openconfig_bgp_types.Ipv4Unicast()
ipv4_afsf.config.enabled = True

# Add the AFI SAFI config to the global AFI SAFI list
bgp_cfg.global_.afi_safis.afi_safi.append(ipv4_afsf)
```

Invoking the CRUD Service

The CRUD service provides methods to create, read, update and delete entities on a device making use of the session provided by a service provider (NETCONF in this case). In order to use the CRUD service, we need to import the *CRUDService* class:

```
from ydk.services import CRUDService
```

Next, we instantiate the CRUD service:

```
crud_service = CRUDService()
```

Finally, we invoke the create method of the `CRUDService` class passing in the service provider instance and our entity (bgp_cfg):

```
try:
    crud_service.create(sp_instance, bgp_cfg)
except YPYError:
```

Note if there were any errors the above API will raise a YPYError exception.

Logging

YDK uses common Python logging. All modules are based on the "ydk" log:

```
import logging
log = logging.getLogger('ydk')
log.setLevel(logging.DEBUG)
ch = logging.StreamHandler()
log.addHandler(ch)
```

Read Using Object Filter

In read operation, YDK object is used as read filter. This document explains how to use YDK object as a read filter. Examples below use ydktest.json profile file to generate YDK test package. Let's write some boilerplate code for device connection:

```
from ydk.services import CRUDService
from ydk.providers import NetconfServiceProvider
from ydk.models import ydktest_filterread as ysanity
ncc = NetconfServiceProvider(address='127.0.0.1',
                             username='admin',
                             password='admin',
                             protocol='ssh',
                             port=12022)
crud = CRUDService()
```

and *ydk.services.CodecService* to simplify payload comparison:

```
from ydk.services.codec_service import CodecService
from ydk.providers.codec_provider import CodecServiceProvider
codec = CodecService()
codec_provider = CodecServiceProvider(type='xml')
```

and configure the device with the initial data below:

```
a = ysanity.A()
a.a1, a.a2, a.a3 = "some value", "value of a2", "value of a3"
a.b.b1, a.b.b2, a.b.b3 = "some value", "value of b2", "value of b3"
a.b.f = a.b.F()
a.b.f.f1 = 'f'
a.b.c = a.b.C()
a.b.d.d1 = "some value d1"
a.b.d.d2 = "value of d2"
a.b.d.d3 = "value of d3"
a.b.d.e.e1, a.b.d.e.e2 = "some value e1", "value of e2"
l1, l2, l3 = a.Lst(), a.Lst(), a.Lst()
```

```
l1.number, l1.value = 1, "one"
l2.number, l2.value = 2, "two"
l3.number, l3.value = 3, "three"
a.lst.extend([l1, l2, l3])

crud.create(ncc, a)
```

The configuration above will config following data in device:

```
<a xmlns="http://cisco.com/ns/yang/ydk-filter">
  <a1>some value</a1>
  <a2>value of a2</a2>
  <a3>value of a3</a3>
  <b>
    <b1>some value</b1>
    <b2>value of b2</b2>
    <b3>value of b3</b3>
    <c/>
    <d>
      <d1>some value d1</d1>
      <d2>value of d2</d2>
      <d3>value of d3</d3>
      <e>
        <e1>some value e1</e1>
        <e2>value of e2</e2>
      </e>
    </d>
    <f>
      <f1>f</f1>
    </f>
  </b>
  <lst>
    <number>1</number>
    <value>one</value>
  </lst>
  <lst>
    <number>2</number>
    <value>two</value>
  </lst>
  <lst>
    <number>3</number>
    <value>three</value>
  </lst>
</a>
```

where <*c*> and <*f*> are presence container.

Read everything

The simplest filter is the top level container:

```
a_read = crud.read(ncc, ysanity.A())
print codec.encode(codec_provider, a_read)
```

the top level container filters nothing and return every data under current level:

```
<a xmlns="http://cisco.com/ns/yang/ydk-filter">
  <a1>some value</a1>
```

```
  <a2>value of a2</a2>
  <a3>value of a3</a3>
  <b>
    <b1>some value</b1>
    <b2>value of b2</b2>
    <b3>value of b3</b3>
    <c/>
    <d>
      <d1>some value d1</d1>
      <d2>value of d2</d2>
      <d3>value of d3</d3>
      <e>
        <e1>some value e1</e1>
        <e2>value of e2</e2>
      </e>
    </d>
    <f>
      <f1>f</f1>
    </f>
  </b>
  <lst>
    <number>1</number>
    <value>one</value>
  </lst>
  <lst>
    <number>2</number>
    <value>two</value>
  </lst>
  <lst>
    <number>3</number>
    <value>three</value>
  </lst>
</a>
```

Filter out more stuff

To make the filter more strict, you could assign more value to it. For example, if you are only interested in presence container *C*:

```
a = ysanity.A()
a.b.c = a.b.C()
a_read = crud.read(ncc, a)
print codec.encode(a_read)
```

```
<a xmlns="http://cisco.com/ns/yang/ydk-filter">
  <b>
    <c/>
  </b>
</a>
```

Content match nodes

According to NETCONF RFC, a "content match node" is used to select sibling nodes for filter output. Let's try this concept with the following example:

```
a = ysanity.A()
a.b.b1 = "some value"
a_read = crud.read(ncc, a)
print codec.encode(codec_provider, a_read)
```

In the example show above, the *a.b.b1* leaf serves as a content match node, therefore its siblings <*b2*> , <*b3*>, <*c*>, <*d*>, <*f*> and their children are all being kept.

```
<a xmlns="http://cisco.com/ns/yang/ydk-filter">
  <b>
    <b1>some value</b1>
    <b2>value of b2</b2>
    <b3>value of b3</b3>
    <c/>
    <d>
      <d1>some value d1</d1>
      <d2>value of d2</d2>
      <d3>value of d3</d3>
      <e>
        <e1>some value e1</e1>
        <e2>value of e2</e2>
      </e>
    </d>
    <f>
      <f1>f</f1>
    </f>
  </b>
</a>
```

Read on leaf

YDK also provides you with a *READ* class that could be used to read the value on a particular leaf. Let's use this *READ* class and import it from *ydk.types*:

```
from ydk.types import READ
a = ysanity.A()
a.a1 = READ()
a_read = crud.read(ncc, a)
print codec.encode(codec_provider, a_read)
```

```
<a xmlns="http://cisco.com/ns/yang/ydk-filter">
  <a1>some value</a1>
</a>
```

Delete operation

This document explains how to use YDK `delete` operation to delete nodes. Examples below use ydktest.json profile file to generate YDK test package.

Let's write some boilerplate code for device connection:

```
from ydk.services import CRUDService
from ydk.providers import NetconfServiceProvider
from ydk.models import ydktest_filterread as ysanity
ncc = NetconfServiceProvider(address='127.0.0.1',
                             username='admin',
```

```
                                password='admin',
                                protocol='ssh',
                                port=12022)
crud = CRUDService()
```

The delete operation can be executed on YANG containers and lists. Specific items in YANG list or leaf-list can also be deleted. To delete a container:

```
runner = ysanity.Runner()
crud.delete(ncc, runner)
```

To delete a list:

```
runner = ysanity.Runner()
runner.one.name = 'one'
foo = ysanity.Runner.OneList.Ldata()
bar = ysanity.Runner.OneList.Ldata()
foo.number = 1
foo.name = 'foo'
bar.number = 2
bar.name = 'bar'
baz.number = 1
baz.name = 'baz'
runner.one_list.ldata.extend([foo, bar, baz])
    crud.delete(ncc, runner.one_list.ldata)
```

To delete a slice of above list:

```
crud.delete(ncc, runner.one_list.ldata[1:])
```

The same syntax could be used to delete items in leaf-list.

Presence Classes

According to RFC 6020, YANG supports two styles of containers, one for organizing hierarchy, another for representing configuration data. For instance the existence of presence container ssh indicates the capability to log in to the device using ssh. Let's consider a container named conditions, with two sub container match-prefix-set(presence) and match-neighbor-set(non-presence), be generated as YDK class Conditions:

```
class Conditions(object):

    def __init__(self):
        self.match_prefix_set = None
        self.match_neighbor_set = Conditions.MatchNeighborSet()
```

Since the existence of sub container match-prefix-set its self representing configuration data, we should not assign an instance of class MatchPrefixSet to it when initializing, and the user need to manually assign it afterwards.

Using Types

This document will explain and give examples to using the ydk types. Types explained will include:

- Empty.
- Decimal64.
- FixedBitsDict.

- YList.

Example use of Empty type

- The leaf being configured (accept_route) under the module ydk.models.openconfig.openconfig_routing_policy:

accept_route: accepts the route into the routing table. **type:** Empty

```
from ydk.models.openconfig.openconfig_routing_policy import RoutingPolicy

# configure policy definition
routing_policy = RoutingPolicy()
policy_definition = routing_policy.policy_definitions.PolicyDefinition()
policy_definition.name = "POLICY2"
# community-set statement
statement = policy_definition.statements.Statement()
statement.actions.accept_route = Empty() # accept_route is of Empty type
```

Example use of Decimal64 type

- The leaf being configured (restart_timer) under the ydk.models.openconfig_bgp.bgp module:

restart_timer: Time interval in seconds after which the BGP session is re-established after being torn down due to exceeding the max-prefix limit. **type:** Decimal64

```
from ydk.models.openconfig.openconfig_bgp import Bgp

config = Bgp.Neighbors.Neighbor.AfiSafis.AfiSafi.Ipv4LabelledUnicast.PrefixLimit.Config()
config.restart_timer = Decimal64('3.343') # restart_timer is of Decimal64 type
```

Example use of FixedBitsDict type

- The leaf being configured (restart_timer) under the ydk.models.ietf.ietf_netconf_acm module:

access_operations: Access operations associated with this rule. This leaf matches if it has the value '*' or if the bit corresponding to the requested operation is set. **type:** str

```
from ydk.models.ietf.ietf_netconf_acm import Nacm

rule_list   = Nacm.RuleList()
rule        = rule_list.Rule()
rule.parent = rule_list
rule.rule_list.rule.access_operations['read'] = True # access_operations is of bits type
```

Example use of YList type

- The node being configured is afi_safi under the ydk.models.openconfig_bgp.bgp module:

```
from ydk.models.openconfig.openconfig_bgp import Bgp

bgp = Bgp()
afi_safi = bgp.global_.afi_safis.AfiSafi() # afi_safi is of YList type
afi_safi.afi_safi_name = oc_bgp_types.Ipv4UnicastIdentity()
afi_safi.config.afi_safi_name = oc_bgp_types.Ipv4UnicastIdentity()
```

```
afi_safi.config.enabled = True
bgp.global_.afi_safis.afi_safi.append(afi_safi)
```

Example use of YLeafList type

- The leaf being configured (ipv4_dscp) under the ydk.models.cisco_ios_xr.Cisco_IOS_XR_asr9k_policymgr_cfg module:

ipv4_dscp: An leaflist of Match IPv4 DSCP. **type:** YLeafList

```
from ydk.models.asr9k.Cisco_IOS_XR_asr9k_policymgr_cfg import PolicyManager

match = PolicyManager.ClassMaps.ClassMap.Match()
match.ipv4_dscp.extend(['15', '16', '17', '18', '19'])
even_elements = match.ipv4_dscp[::2]

# Note: YLeafList is associative array, attempt to add duplicated element will raise Exception.
match.ipv4_dscp.append('15')
# YPYDataValidationError will be raised.
```

Deviation

Overview

Not all devices faithfully support features defined in the standard yang module. For a particular device, it could support only part of features or the feature it supported varies from the standard module. In YANG, we use deviation statement to specify it. For example, in cisco-xr-bgp-deviations.yang,

```
deviation /bgp:bgp/bgp:global/bgp:afi-safis/bgp:afi-safi/bgp:apply-policy {
  deviate not-supported;
}
```

apply-policy is not supported.

How to use deviation with YDK

When using YDK to program a device which has some unsupported features, YDK will raise a validation error before sending payload to device. In the example below, the device has published a deviation `cisco-xr-bgp-deviations.yang` on a standard bgp module.

YDK will raise an error if an app tries to assign a value to this feature:

```
from ydk.models.bgp import bgp
from ydk.models.routing.routing_policy import DefaultPolicyTypeEnum, RoutingPolicy

bgp_cfg = bgp.Bgp()
ipv4_afsf = bgp_cfg.global_.afi_safis.AfiSafi()
ipv4_afsf.afi_safi_name = 'ipv4-unicast'
ipv4_afsf.config.afi_safi_name = 'ipv4-unicast'
ipv4_afsf.config.enabled = True
ipv4_afsf.apply_policy.config.default_export_policy = \
    DefaultPolicyTypeEnum.ACCEPT_ROUTE
bgp_cfg.global_.afi_safis.afi_safi.append(ipv4_afsf)
```

YDK will raise YPYDataValidationError when processing the above python object.

Behind the Scenes

YDK use pyang to compile yang module to intermediate tree structured python objects(pyang statements), and feed those objects to YDK's API module to generate python objects(YDK packages) suitable for language binding. If, compiled with deviation module, pyang will automatically trim the unsupported subtrees, and change the deviated feature. With subtree being trimmed, the original API will not being generated. However, we also need the deviation meta information at runtime.

Pyang's infrastructure provide way to insert additional phases between basic phases. So what YDK does is to capture deviation information before trim for deviation happens, and restore the information after that.

With those deviation messages being captured in pyang statements object, YDK's API module could use that information to print original module along with a central point for deviation.

Before send payload to the device, YDK will get active deviation from the device it is talking to, from ncclient, and use this information to trim/validate YDK-py object accordingly.

API Guide

ydk.errors module

errors

Contains types representing the Exception hierarchy in YDK.

exception ydk.errors.**YPYDataValidationError**
Bases: *ydk.errors.YPYError*

Exception for Client Side Data Validation.

Type Validation.

Any data validation error encountered that is related to type validation encountered does not raise an Exception right away.

To uncover as many client side issues as possible, an i_errors list is injected in the parent entity of any entity with issues. The items added to this i_errors list captures the object types that caused the error as well as an error message.

exception ydk.errors.**YPYError**
Bases: exceptions.Exception

Base Exception for YDK Errors.

ydk.providers module

providers.py

Service Providers module. Current implementation supports the NetconfServiceProvider which uses ncclient (a Netconf client library) to provide CRUD services.

class ydk.providers.**NetconfServiceProvider**(***kwargs*)
Bases: *ydk.providers.ServiceProvider*

NCClient based Netconf Service Provider.

Initialization parameter of NetconfServiceProvider

Parameters

- **address** – The address of the netconf server
- **port** – The port to use default is 830
- **username** – The name of the user
- **password** – The password to use
- **protocol** – One of either ssh or tcp

Timeout Default to 45

close()
Closes the netconf session.

class ydk.providers.**CodecServiceProvider**(***kwargs*)
Bases: *ydk.providers.ServiceProvider*

Codec Service Provider.

Initialization parameter of CodecServiceProvider

Parameters **type** – Type of encoding. Currently, 'xml' type is supported

class ydk.providers.**ServiceProvider**
Bases: object

Base class for Service Providers.

close()
Base method to close service provider instance session.

Exception for Client Side Data Validation.

Type Validation.

Any data validation error encountered that is related to type validation encountered does not raise an Exception right away.

To uncover as many client side issues as possible, an i_errors list is injected in the parent entity of any entity with issues. The items added to this i_errors list captures the object types that caused the error as well as an error message.

ydk.services module

services.py

The Services module. Supported services include

CRUDService: Provides Create/Read/Update/Delete API's

class ydk.services.**CRUDService**
Bases: ydk.services.Service

CRUD Service class for supporting CRUD operations on entities.

create(*provider*, *entity*)
Create the entity

Parameters

- **provider** – An instance of ydk.providers.ServiceProvider

- **entity** – An instance of an entity class defined under the ydk.models package or sub-packages.

Returns None

Raises

- ***ydk.errors.YPYDataValidationError*** – if validation error has occurred
- ***ydk.errors.YPYError*** – if other error has occurred

Possible Errors:

•a server side error

•there isn't enough information in the entity to prepare the message (eg. missing keys)

read(*provider*, *read_filter*, *only_config=False*)

Read the entity or entities.

Parameters

- **provider** – An instance of ydk.providers.ServiceProvider
- **read_filter** – A read_filter is an instance of an entity class. An entity class is a class defined under the ydk.models package that is not an Enum, Identity or a subclass of FixedBitsDict). Attributes of this entity class may contain values that act as match expressions or can be explicitly marked as to be read by assigning an instance of type *ydk.types.READ* to them.
- **only_config** – Flag that indicates that only the data that represents configuration data is to be fetched. Default is set to False i.e both oper and config data will be fetched.

Returns The entity or list of entities as identified by the *read_filter*

Raises

- ***ydk.errors.YPYDataValidationError*** – if validation error has occurred
- ***ydk.errors.YPYError*** – if other error has occurred

Possible errors could be

•a server side error

•if there isn't enough information in the entity to prepare the message (missing keys for example)

update(*provider*, *entity*)

Update the entity.

Note:

•An attribute of an entity class can be deleted by setting to an instance of *ydk.types.DELETE*.

•An entity can only be updated if it exists on the server. Otherwise a *ydk.errors.YPYError* will be raised.

Parameters

- **provider** – An instance of ydk.providers.ServiceProvider
- **entity** – An instance of an entity class defined under the ydk.models package or sub-packages.

Returns None

Raises

- ***ydk.errors.YPYDataValidationError*** – if validation error has occurred
- ***ydk.errors.YPYError*** – if other error has occurred

Possible errors could be

•a server side error

•if there isn't enough information in the entity to the message (missing keys for example)

delete (*provider*, *entity*)
Delete the entity

Parameters

- **provider** – An instance of ydk.providers.ServiceProvider
- **entity** – An instance of an entity class defined under the ydk.models package or sub-packages.

Returns None

Raises

- ***ydk.errors.YPYDataValidationError*** – if validation error has occurred
- ***ydk.errors.YPYError*** – if other error has occurred

Possible errors could be

•a server side error

•if there isn't enough information in the entity to the message (missing keys for example)

NetconfService: Provides API's to execute netconf operations

class ydk.services.**Datastore**
Bases: enum.Enum

Netconf datastore type

candidate = 1
Candidate

running = 2
Running

startup = 3
Startup

class ydk.services.**NetconfService**

Bases: ydk.services.Service.

Netconf Service class for executing netconf operations.

cancel_commit (*provider*, *persist_id=None*)
Execute an cancel-commit operation to cancel an ongoing confirmed commit.

Parameters

- **provider** (ydk.providers.ServiceProvider) – A provider instance.

- **persist_id** (*str*) – This parameter is given in order to cancel a persistent confirmed commit. The value must be equal to the value given in the 'persist' parameter to the commit operation. If it does not match, the operation fails with an 'invalid-value' error.

Returns An ok reply string if operation succeeds.

Return type str

Raises

- ***ydk.errors.YPYDataValidationError*** – If validation error has occurred.
- ***ydk.errors.YPYError*** – If other error has occurred. Possible errors could be:
  - A server side error
  - If there isn't enough information in the entity to the message (missing keys for example).

close_session (*provider*)

Execute a close-session operation to cancel an ongoing confirmed commit.

Parameters **provider** (ydk.providers.ServiceProvider) – A provider instance.

Returns An ok reply string if operation succeeds.

Return type str

Raises

- ***ydk.errors.YPYDataValidationError*** – if validation error has occurred
- ***ydk.errors.YPYError*** – if other error has occurred.

  Possible errors could be:
  - A server side error.
  - If there isn't enough information in the entity to the message (missing keys for example).

commit (*provider*, *confirmed=False*, *confirm_timeout=None*, *persist=False*, *persist_id=None*)

Execute a commit operation to commit the candidate configuration as the device's new current configuration.

Parameters

- **provider** (ydk.providers.ServiceProvider) – A provider instance.
- **confirmed** (*bool*) – Perform a confirmed commit operation.
- **confirm_timeout** (*int*) – The timeout interval for a confirmed commit.
- **persist** (*str*) – Make a confirmed commit persistent. A persistent confirmed commit is not aborted if the NETCONF session terminates. The only way to abort a persistent confirmed commit is to let the timer expire, or to use the <cancel-commit> operation. The value of this parameter is a token that must be given in the 'persist-id' parameter of <commit> or <cancel-commit> operations in order to confirm or cancel the persistent confirmed commit. The token should be a random string.
- **persist_id** (*str*) – This parameter is given in order to commit a persistent confirmed commit. The value must be equal to the value given in the 'persist' parameter to the <commit> operation. If it does not match, the operation fails with an 'invalid-value' error.

Returns An ok reply string if operation succeeds.

Return type str

Raises

- ***ydk.errors.YPYDataValidationError*** – if validation error has occurred.
- ***ydk.errors.YPYError*** – if other error has occurred.

  Possible errors could be:

  – A server side error.

  – If there isn't enough information in the entity to the message (missing keys for example).

copy_config(*provider*, *target*, *source*, *with_defaults_option=None*)

Execute a copy-config operation to create or replace an entire configuration datastore with the contents of another complete configuration datastore.

Parameters

- **provider** (ydk.providers.ServiceProvider) – A provider instance.
- **target** – Particular configuration to copy to. Valid options are `Datastore.candidate`, `Datastore.running`, `Datastore.startup` and url(`str`) if the device has such feature advertised in device capability.
- **source** – Particular configuration to copy from. Valid options are `Datastore.candidate`, `Datastore.running`, `Datastore.startup` and url(`str`) if the deivce has such feature advertised in capability. A YDK entity object can also be used for this parameter.
- **with_defaults** (`ietf_netconf.WithDefaultsModeEnum`) – The explicit defaults processing mode requested.

Returns An ok reply string if operation succeeds.

Return type str

Raises

- ***ydk.errors.YPYDataValidationError*** – if validation error has occurred.
- ***ydk.errors.YPYError*** – if other error has occurred.

  Possible errors could be

  – A server side error.

  – If there isn't enough information in the entity to the message (missing keys for example).

delete_config(*provider*, *target*)

Execute an delete-config operation to delete a configuration datastore.

Parameters

- **provider** (ydk.providers.ServiceProvider) – A provider instance.
- **target** – Particular configuration to delete. Valid options are `Datastore.startup` or url(`str`).

Returns An ok reply string if operation succeeds.

Return type str

Raises

- ***ydk.errors.YPYDataValidationError*** – if validation error has occurred.
- ***ydk.errors.YPYError*** – if other error has occurred.

  Possible errors could be:

  – A server side error.

  – If there isn't enough information in the entity to the message (missing keys for example).

discard_changes (*provider*)

Execute a discard-changes operation to revert the candidate configuration to the current running configuration.

Parameters **provider** (ydk.providers.ServiceProvider) – A provider instance.

Returns An ok reply string if operation succeeds.

Return type str

Raises

- ***ydk.errors.YPYDataValidationError*** – if validation error has occurred.
- ***ydk.errors.YPYError*** – if other error has occurred.

  Possible errors could be:

  – A server side error.

  – If there isn't enough information in the entity to the message (missing keys for example).

edit_config (*provider*, *target*, *config*, *default_operation=None*, *error_option=None*, *test_option=None*)

Execute an edit-config operation to load all or part of a specified configuration to the specified target configuration.

Parameters

- **provider** (ydk.providers.ServiceProvider) – A provider instance.
- **target** – Particular configuration to copy from. Valid options are Datastore.candidate, Datastore.running.
- **config** – A YDK entity object used as a config block.
- **default_operation** (EditConfigRpc.Input.DefaultOperationEnum) – Selects the default operation for this edit-config request.
- **error_option** (EditConfigRpc.Input.ErrorOptionEnum) – Selects the error option for this edit-config request.
- **test_option** (EditConfigRpc.Input.TestOptionEnum) – Selects the test option for this edit-config request.

Returns An ok reply string if operation succeeds.

Return type str

Raises

- ***ydk.errors.YPYDataValidationError*** – if validation error has occurred.

- ***ydk.errors.YPYError*** – if other error has occurred.

  Possible errors could be:

  - A server side error.
  - If there isn't enough information in the entity to the message (missing keys for example).

get_config (*provider*, *source*, *get_filter*, *with_defaults_option=None*)
Execute a get-config operation to retrieve all or part of a specified configuration.

Parameters

- **provider** (ydk.providers.ServiceProvider) – A provider instance.
- **get_filter** – A YDK entity object used as a subtree filter or XPath filter.
- **source** – Particular configuration to retrieve. Valid options are Datastore.candidate, Datastore.running, and Datastore.startup.
- **with_defaults** (ietf_netconf.WithDefaultsModeEnum) – The explicit defaults processing mode requested.

Returns A YDK entity object represents copy of the running datastore subset and/or state data that matched the filter criteria (if any). An empty data container indicates that the request did not produce any results.

Return type object

Raises

- ***ydk.errors.YPYDataValidationError*** – if validation error has occurred.
- ***ydk.errors.YPYError*** – if other error has occurred.

  Possible errors could be:

  - A server side error.
  - If there isn't enough information in the entity to the message (missing keys for example).

get (*provider*, *get_filter*, *with_defaults_option=None*)
Execute a get operation to retrieve running configuration and device state information.

Parameters

- **provider** (ydk.providers.ServiceProvider) – A provider instance.
- **get_filter** – This parameter specifies the portion of the system configuration and state data to retrieve.
- **with_defaults** (ietf_netconf.WithDefaultsModeEnum) – The explicit defaults processing mode requested.

Returns A YDK entity object represents copy of the running datastore subset and/or state data that matched the filter criteria (if any). An empty data container indicates that the request did not produce any results.

Return type object

Raises

- ***ydk.errors.YPYDataValidationError*** – if validation error has occurred.

- ***ydk.errors.YPYError*** – if other error has occurred.

  Possible errors could be:

  - A server side error.
  - If there isn't enough information in the entity to the message (missing keys for example).

kill_session (*provider*, *session_id*)

Execute a kill-session operation to force the termination of a NETCONF session.

Parameters

- **provider** (ydk.providers.ServiceProvider) – A provider instance.
- **session_id** (*int*) – Particular session to kill.

Returns An ok reply string if operation succeeds.

Return type str

Raises

- ***ydk.errors.YPYDataValidationError*** – if validation error has occurred.
- ***ydk.errors.YPYError*** – if other error has occurred.

  Possible errors could be:

  - A server side error.
  - If there isn't enough information in the entity to the message (missing keys for example).

lock (*provider*, *target*)

Execute a lock operation to allow the client to lock the configuration system of a device.

Parameters

- **provider** (ydk.providers.ServiceProvider) – A provider instance.
- **target** – Particular configuration to lock. Valid options are `Datastore.candidate`, `Datastore.running`, and `Datastore.startup` if the device has such feature advertised.

Returns An ok reply string if operation succeeds.

Return type str

Raises

- ***ydk.errors.YPYDataValidationError*** – if validation error has occurred.
- ***ydk.errors.YPYError*** – if other error has occurred.

  Possible errors could be:

  - A server side error.
  - If there isn't enough information in the entity to the message (missing keys for example).

unlock (*provider*, *target*)

Execute an unlock operation to release a configuration lock, previously obtained with the 'lock' operation.

Parameters

- **provider** (ydk.providers.ServiceProvider) – A provider instance.
- **target** – Particular configuration to unlock. Valid options are Datastore.candidate, Datastore.running, and Datastore.startup if the device has such feature advertised.

Returns An ok reply string if operation succeeds.

Return type str

Raises

- ***ydk.errors.YPYDataValidationError*** – if validation error has occurred
- ***ydk.errors.YPYError*** – if other error has occurred

  Possible errors could be:

  – A server side error.

  – If there isn't enough information in the entity to the message (missing keys for example).

validate (*provider*, *source=None*, *config=None*)

Execute a validate operation to validate the contents of the specified configuration.

Parameters

- **provider** (ydk.providers.ServiceProvider) – A provider instance.
- **source** – Particular configuration to validate. Valid options are Datastore.candidate, Datastore.running, Datastore.startup and url(str) if the deivce has such feature advertised in device capability. A YDK entity object can also be used for this parameter.
- **with_defaults** (ietf_netconf.WithDefaultsModeEnum) – The explicit defaults processing mode requested.

Returns An ok reply string if operation succeeds.

Return type str

Raises

- ***ydk.errors.YPYDataValidationError*** – if validation error has occurred.
- ***ydk.errors.YPYError*** – if other error has occurred.

  Possible errors could be:

  – A server side error.

  – If there isn't enough information in the entity to the message (missing keys for example).

CodecService: Provides encode/decode API's

class ydk.services.**CodecService**

Bases: ydk.services.Service

Codec Service class for supporting encoding entities and decoding payloads.

encode (*provider*, *entity*)

Encodes the python entity and returns the payload. Entity is either:

- an instance of an entity class defined under the ydk.models package or subpackages, or
- **a dictionary containing:**
  - module names as keys and
  - entity instances as values

Returns

encoded value can be: - an instance of an XML payload defined for a yang module, or - a dictionary containing:

- module names as keys and
- instances of XML payload defined for a yang module as values

Raises *ydk.errors.YPYDataValidationError* – if validation error has occurred

decode (*provider*, *payload*)

Decodes the the payload and returns the python entity. Payload is either:

- an instance of an XML payload defined for a yang module, or
- **a dictionary containing:**
  - module names as keys and
  - instances of XML payload defined for a yang module as values

Returns

decoded entity. Entity is either: - an instance of an entity class defined under the ydk.models package or subpackages, or - a dictionary containing:

- module names as keys and
- entity instances as values

Raises *ydk.errors.YPYDataValidationError* – if validation error has occurred

ExecutorService: Provides API to execute RPCs

class ydk.services.**ExecutorService**

Bases: ydk.services.Service

Executor Service class for supporting execution of RPCs.

execute_rpc(self, provider, rpc):

Create the entity

Parameters

- **provider** – An instance of ydk.providers.ServiceProvider
- **rpc** – An instance of an RPC class defined under the ydk.models package or subpackages

Returns None

Raises

- *ydk.errors.YPYDataValidationError* – if validation error has occurred
- *ydk.errors.YPYError* – if other error has occurred

Possible Errors:

- a server side error
- there isn't enough information in the entity to prepare the message (eg. missing keys)

ydk.types module

types.py

Contains type definitions.

class ydk.types.**DELETE**
Bases: object

Marker class used to mark nodes that are to be deleted.

Assign DELETE object to a mark a leaf for deletion. A CRUD update operation will delete the leaf from the device it is on.

class ydk.types.**Decimal64**(*str_val*)
Bases: object

Represents the decimal64 YANG type. The decimal64 type represents a subset of the real numbers, which can be represented by decimal numerals.

The value space of decimal64 is the set of numbers that can be obtained by multiplying a 64-bit signed integer by a negative power of ten, i.e., expressible as "i x 10^-n" where i is an integer64 and n is an integer between 1 and 18, inclusively.

str_val String representation of the decimal64 number.

class ydk.types.**Empty**
Bases: object

Represents the empty type in YANG. The empty built-in type represents a leaf that does not have any value, it conveys information by its presence or absence.

class ydk.types.**FixedBitsDict**(*dictionary*, *pos_map*)
Bases: object

Super class of all classes that represents the bits type in YANG

A concrete implementation of this class has a dictionary. The bits built-in type represents a bit set. That is, a bits value is a set of flags identified by small integer position numbers starting at 0. Each bit number has an assigned name. To set a bit use the name of the bit as a key into the dictionary and set the value to True (False to unset).

class ydk.types.**READ**
Bases: object

Marker class used to mark nodes that are to be read.

class ydk.types.**YList**
Bases: list

Represents a list with support for hanging a parent.

All YANG based entity classes that have lists in them use YList to represent the list.

class ydk.types.**YLeafList**
Bases: *ydk.types.YList*

Represents a leaf-list with support for hanging a parent.

All YANG leaf-list is represented as YLeafList. YLeafList is associative array, it contains unique elemenets.

C

D

E

G

K

L

R

U

V

Y