Send document comments to nexus3k-docfeedback@cisco.com.

Cisco Nexus 3000 Series NX-OS Python API Reference Guide, Release 5.0(3)U4(1)

September 2012

Cisco Systems, Inc.
www.cisco.com

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco website at www.cisco.com/go/offices.

Text Part Number: OL-26601-02

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: www.cisco.com/go/trademarks. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

Cisco Nexus 3000 Series NX-OS Python API Reference Guide, Release 5.0(3)U4(1)
© 2012 Cisco Systems, Inc. All rights reserved.

CHAPTER 1 **Overview** 1-1

Information About the Python API 1-1

Installing Python 1-2

Installing Third Party Pure Python Packages 1-2

Using Python 1-2

Entering the Python Shell 1-2

Executing Scripts 1-3

Passing Parameters to the Script 1-3

Embedded Event Manager Support 1-3

Calculating MD5 Checksum 1-4

CHAPTER 2 **API Functions** 2-5

Classes 2-5

API Functions 2-7

class BGPSession(*ASN*=None, *vrf*='default') 2-7

class BGPNeighbor(*ip_address*, vrf= 'default', **kwargs) 2-20

class BufferDepthMonitor() 2-30

class CLI(*command*='', *do_print*=TRUE) 2-31

class CheckPortDiscards(*port*) 2-32

class CiscoSecret(*key*, *type*=0) 2-33

class CiscoSocket([*family*[, *type*[, *proto*]]]) 2-34

class FTPTransfer(*source*, *dest*='', *host*=(None, None), *vrf*='management', *login_timeout*=10, *user*='', *password*='') 2-38

class Feature(__builtin__.object) 2-40

class HTTPTransfer(*source*, *dest*='', *host*=(None, None), *vrf*='management', *login_timeout*=10) 2-41

class History() 2-43

class IPv4ACL(*name*) 2-47

class IPv6ACL(*name*) 2-50

class Interface(*intf*) 2-53

class Key(*key*='', *start*='', end='') 2-54

class MacAddressTable() 2-54

class PasswordProtoTransfer(*source*, *dest*='', *host=(None, None)*, *vrf*='management', *login_timeout*=10, *user*='', *password*='') 2-55

Send document comments to nexus3k-docfeedback@cisco.com.

class RouteMap(*name*, *sequence*, *type*='permit') **2-57**
class Routes() **2-64**
class SCPTransfer(*source*, *dest*='', *host*=(None, None), vrf='management', login_timeout=10, user='', password='') **2-65**
class SFTPTransfer(*source*, dest='', *host*=(None, None), *vrf*='management', login_timeout=10, user='', password='') **2-68**
class SSH() **2-70**
class ShowInterface(*ifname*) **2-71**
class System(__builtin__.object) **2-72**
class TFTPTransfer(*source*, dest='', host=(None, None), vrf='management', *login_timeout*=10) **2-74**
class Tacacs() **2-76**
class Transfer(source, dest='', host=(None, None), vrf='management', login_timeout=10) **2-80**
class VRF(*vrf*) **2-82**
class Vlan() **2-84**
class bfd(*Feature*) **2-85**
class dhcp(*Feature*) **2-86**
class eigrp(*Feature*) **2-87**
class hsrp(*Feature*) **2-88**
class interface-vlan(*Feature*) **2-89**
class lacp(*Feature*) **2-90**
class msdp(*Feature*) **2-91**
class ospf(*Feature*) **2-92**
class ospfv3(*Feature*) **2-93**
class pim(*Feature*) **2-94**
class private-vlan(*Feature*) **2-95**
class ptp(*Feature*) **2-96**
class rip(*Feature*) **2-97**
class scheduler(*Feature*) **2-98**
class telnet(*Feature*) **2-99**
class udld(*Feature*) **2-100**
class vpc(*Feature*) **2-101**
class vrrp(*Feature*) **2-102**
class vtp(*Feature*) **2-103**

Preface

This document lists and describes the Python application programming interface (API) for Cisco Nexus 3000 Series switches.

This chapter includes the following sections:

- Audience, page v
- Document Conventions, page v
- Related Documentation, page vi
- Documentation Feedback, page vi
- Obtaining Documentation and Submitting a Service Request, page vii

Audience

This publication is for system administrators who are responsible for installing and managing Cisco Nexus 3000 Series switches.

Document Conventions

Function descriptions use these conventions:

Conventions	Description
boldface font	Names of the functions are in boldface.
italic font	Arguments for the functions are in italics.
{ }	Elements in braces are required choices.
[ ]	Elements in square brackets are optional.
x | y | z	Alternative, mutually exclusive elements are separated by vertical bars.

Note Means reader *take note*. Notes contain helpful suggestions or references to material not covered in the manual.

Caution Means *reader be careful*. In this situation, you might do something that could result in equipment damage or loss of data.

Related Documentation

Documentation for the Cisco Nexus 3000 Series switch is available at the following URL:

http://www.cisco.com/en/US/products/ps11541/tsd_products_support_series_home.html

The documentation set is divided into the following categories:

Release Notes

The release notes are available at the follwing URL:

http://www.cisco.com/en/US/products/ps11541/prod_release_notes_list.html

Installation and Upgrade Guides

The installation and upgrade guides are available at the following URL:

http://www.cisco.com/en/US/products/ps11541/prod_installation_guides_list.html

Command References

The command references are available at the following URL:

http://www.cisco.com/en/US/products/ps11541/prod_command_reference_list.html

Technical References

The technical references are available at the following URL:

http://www.cisco.com/en/US/products/ps11541/prod_technical_reference_list.html

Configuration Guides

The configuration guides are available at the following URL:

http://www.cisco.com/en/US/products/ps11541/products_installation_and_configuration_guides_list.html

Error and System Messages

The system message reference guide is available at the following URL:

http://www.cisco.com/en/US/products/ps11541/products_system_message_guides_list.html

Documentation Feedback

To provide technical feedback on this document, or to report an error or omission, please send your comments to nexus3k-docfeedback@cisco.com. We appreciate your feedback.

Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, submitting a service request, and gathering additional information, see the monthly *What's New in Cisco Product Documentation*, which also lists all new and revised Cisco technical documentation, at:

http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html

Subscribe to the *What's New in Cisco Product Documentation* as a Really Simple Syndication (RSS) feed and set content to be delivered directly to your desktop using a reader application. The RSS feeds are a free service and Cisco currently supports RSS version 2.0.

Overview

This chapter provides an overview and installation information needed to use the Python application programming interface (API) support on Cisco Nexus 3000 Series switches.

- Information About the Python API, page 1-1
- Installing Python, page 1-2
- Installing Third Party Pure Python Packages, page 1-2
- Using Python, page 1-2

Information About the Python API

Python is an easy-to-learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python website:

http://www.python.org/

The same site also distributes and points to many free third-party Python modules, programs and tools, and additional documentation.

The Cisco Nexus 3000 Series switches support all the features available in Python v2.7.2.

The Python scripting capability on the Cisco Nexus 3000 Series switches allows you to perform the following tasks:

- Running a script to verify configuration on switch bootup.
- Backing up a configuration.
- Proactive congestion management by monitoring and responding to buffer utilization characteristics.
- Integration with the Power-On Auto Provisioning or EEM modules.
- Ability to perform a job at a specific time interval (such as Port Auto Description).
- Programmatic access to the switch command-line interface (CLI) to perform various tasks.

Send document comments to nexus3k-docfeedback@cisco.com.

Installing Python

The Python interpreter is available by default in the Cisco NX-OS software.You can invoke Python by entering the **python** command, and you can write scripts to access Cisco NX-OS APIs by importing the cisco.py module using the **import cisco** command.

Installing Third Party Pure Python Packages

You can install the third-party pure Python package by copying mypkg.tgz on your server. Perform the following steps to extract and install the third-party package:

- Secure-copy the tar file by executing the **copy scp://user@server/path/to/mypkg.tgz bootflash:mypkg.tgz vrf management** command.
- Untar the mypkg.tgz file by using the **tar extract bootflash:mypkg.tgz** command.
- Move the extracted file to bootflash by using the **move bootflash:mypkg-1.2/* bootflash:** command.
- You can install the package by using the **python setup.py install** command.
- Remove the copied file from bootflash.
- You can use the third-party package in scripts or in the Python shell.

```
switch# python
>>> import mypkg
```

Note You will be able to install the third-party packages using the **easy_install** command in future releases.

Using Python

This section describes how to write and execute Python scripts by passing parameters and includes the following topics:

- Entering the Python Shell, page 1-2
- Executing Scripts, page 1-3
- Passing Parameters to the Script, page 1-3
- Embedded Event Manager Support, page 1-3

Entering the Python Shell

You can enter the Python shell by using the **python** command without any parameters.

```
switch# python
Python 2.7.2 (default, Oct 11 2011, 13:55:49)
[GCC 3.4.3 (MontaVista 3.4.3-25.0.143.0800417 2008-02-22)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Loaded cisco NX-OS lib!
>>> print 'helo world!'
helo world!
>>>exit()
switch#
```

Send document comments to nexus3k-docfeedback@cisco.com.

Executing Scripts

You can execute a Python script by using the **python** *filename* command.

```
switch# python test.py
['/bootflash/test.py']
doing 0/1
doing 0/2
doing 1/2
switch#
```

Passing Parameters to the Script

You can execute a Python script by using the **python** *filename* [*arg1, arg2, arg3, ...*] command.

```
switch# python test.py abc xyz 1 2
['/bootflash/test.py', 'abc', 'xyz', '1', '2']
doing 0/1
doing 0/2
doing 1/2
switch#
```

Embedded Event Manager Support

Embedded Event Manager (EEM) supports invocation of Python scripts based on events. Syslog for events can be passed to a Python script by using the **variable $** command.

The following example shows EEM invoking a python script for an IF_DOWN event.

EEM configuration:

```
switch(config)# event manager applet if-mon
switch(config-applet)# event syslog pattern "*IF_DOWN.*"
Configuration accepted successfully
switch(config-applet)# action 1.0 cli python if-mon.py eth1/1 $command
switch(config-applet)# end
```

Python script:

```
import re
import sys
from cisco import *

def findIf ():
x = re.compile ('[Ee]thernet\d+\/\d+')
for a in sys.argv[1:]:
if x.match (a):
print a
return a
return None

print 'Starting my script.. args:'
print sys.argv
intf = findIf ()

if not intf:
intf = 'eth1/1'

print 'Detected shut on interface %s' % intf
i = Interface (intf)
```

Send document comments to nexus3k-docfeedback@cisco.com.

```
print ('---------show run int %s---------\n%s\n' % (intf, i.show().raw_output))
print ('----------------------\n%s\n' % o)
print 'Restoring interface %s' % intf
i.set_state ('up')
i.set_description ('++dont shut++')
print ('----------------------\n%s\n' % o)
print ('---------show run int %s---------\n%s\n' % (intf, i.show().raw_output))

print ('\nbye\n')
```

Calculating MD5 Checksum

Every time you make a change to the configuration script, ensure that you recalculate the MD5 checksum. The following example shows the script to be run in a bash shell to recalculate the MD5 checksum:

```
#!/bin/env python
#md5sum="13f71ffebe16cc6143a6af2186b40948"
# If any changes to this script file are made, please run the below command
# in bash after modifications.
# The above is the (embedded) md5sum of this file taken without this line,
# can be # created this way if using a bash shell:
# f=poap_fabric.py ; cat $f | sed '/^#md5sum/d' > $f.md5 ; sed -i
"s/^#md5sum=.*/#md5sum=\"$(md5sum $f.md5 | sed 's/ .*//')\"/" $f
# This way this script's integrity can be checked in case you do not trust
# tftp's ip checksum. This integrity check is done by /isan/bin/poap.bin).
# The integrity of the files downloaded later (images, config) is checked
# by downloading the corresponding file with the .md5 extension and is
# done by this script itself.
```

API Functions

This chapter provides information about Python application programming interface (API) functions and includes the following sections:

- Classes, page 2-5
- API Functions, page 2-7

Classes

CLASSES

History

ACL

IPv4ACL

IPv6ACL

CLI

BufferDepthMonitor

CheckPortDiscards

ShACL

ShFeature

ShTacas

ShTacasServer

ShowInterface

ShowVlan

shSshKey

CiscoSecret

Feature

BGPSession

SSH

Tacacs

bfd

Send document comments to nexus3k-docfeedback@cisco.com.

```
    dhcp
    eigrp
    hsrp
    interface-vlan
    lacp
    msdp
    ospf
    ospfv3
    pim
    private-vlan
    ptp
    rip
    scheduler
    telnet
    udld
    vpc
    vrrp
    vtp
```

Interface

Key

MacAddressTable

RouteMap

Routes

System

Transfer

```
    HTTPTransfer
    PasswordProtoTransfer
      FTPTransfer
      SCPTransfer
      SFTPTransfer
    TFTPTransfer
```

VRF

Vlan

```
 __builtin__.type(__builtin__.object)
  FeatureFactory
socket._socketobject(__builtin__.object)
  CiscoSocket
```

Send document comments to nexus3k-docfeedback@cisco.com.

API Functions

class BGPSession(*ASN*=None, *vrf*='default')

Use this class to configure the BGP feature.

Method resolution order:

`BGPSession`

`Feature`

`__builtin__.object`

Methods defined here:

add_network(*network*, ***kwargs*)

Configure an IP prefix to advertise. To remove the IP prefix to advertise, set the optional *no* argument to `TRUE`.

Arguments

network: A string representing the IP prefix in either Classless Inter-Domain Routing (CIDR) notation or dotted quad. For example, '192.0.2.0/24' or '192.0.2.0/255.255.255.0'.

Optional Arguments

route-map: Specifies the name of the route map to modify attributes.

no: Set to `TRUE` to remove a network.

Returns

`TRUE` on success.

Usage Guidelines

The IP prefix to advertise is considered for bestpath purposes and advertisement to peers only if a route of equal or more specificity is present in the routing table.

cfg_aggregate_address(*address*, ***kwargs*)

Create a summary address in a Border Gateway Protocol (BGP) routing table. To remove the summary address, set the optional *no* argument to `TRUE`.

Arguments

address: A string representing the aggregate IP address and mask in either CIDR notation or dotted quad. For example, '192.0.2.0/24' or '192.0.2.0/255.255.255.0'.

Optional Arguments

advertise_map: A string that specifies the name of the route map used to select attribute information from specific routes. Should be an alphanumeric string up to 63 characters long.

as_set: A Boolean. Set to `TRUE` to generate the autonomous system set path information and community information from the contributing paths.

attribute_map: A string that specifies the name of the route map used to set the attribute information for specific routes. Should be an alphanumeric string up to 63 characters long.

Send document comments to nexus3k-docfeedback@cisco.com.

summary_only: A Boolean. Set to TRUE to filter all more specific routes from updates.

suppress_map: A string that specifies the name of the route map used to conditionally filter more specific routes. Should be an alphanumeric string up to 63 characters.

no: A Boolean. Set to TRUE to remove a summary address.

Returns

TRUE on success.

cfg_bestpath(**kwargs)

Modify the default best-path selection algorithm.

Optional Arguments

always_compare_med: A Boolean. Set to TRUE to compare the Multi-Exit Discriminator (MED) on paths from a different autonomous system (AS). Set to FALSE to remove this configuration.

as_path_multipath_relax: A Boolean. Set to TRUE to configure a BGP routing process to consider the different AS paths and load-balance multiple paths during best-path route selection. Set to FALSE to remove this configuration.

compare_router_id: A Boolean. Set to TRUE to configure a Border Gateway Protocol (BGP) routing process to compare identical routes received from different external peers during the best-path selection process and to select the route with the lowest router ID as the best-path. Set to FALSE to remove this configuration.

med_missing_as_worst: A Boolean. Set to TRUE to assign the value of infinity to received routes that do not carry the MED attribute, making these routes the least desirable. Set to FALSE to remove this configuration.

med_non_deterministic: A Boolean. Set to TRUE to specify that the best-MED-path among paths is not picked from the same AS. Set to FALSE to remove this configuration.

Returns

TRUE on success.

cfg_cli_event_history(**kwargs)

Enable the CLI event history to be saved in a buffer. It is enabled by default. The default buffer size is ‘small’. Possible sizes are ‘small’, ‘medium’, and ‘large’.

Optional Arguments

size: A string. Possible values are: ‘small’, ‘medium’, ‘large’, or ‘disable’.

no: A Boolean. Set to TRUE to stop saving the CLI event history.

Returns

TRUE on success.

cfg_cluster_id(*cluster_id*, **kwargs)

Set the cluster ID on a route reflector in a route reflector cluster. To remove the cluster ID, set the optional *no* argument to TRUE.

Send document comments to nexus3k-docfeedback@cisco.com.

Arguments

cluster_id: Cluster ID of this router acting as a route reflector. Can be specified as an integer ranging from 1 to 4294967295 or as a string in dotted format: 'A.B.C.D'.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove the specified cluster ID.

Returns

`TRUE` on success.

cfg_confederation_identifier(*id*, *******kwargs*)

Specify a BGP confederation identifier. To remove the confederation identifier, set the optional *no* argument to `TRUE`.

Arguments

id: The AS number. Can be specified as an integer ranging from 1 to 4294967295 or as a string in the following format: <1-65535>[.<0-65535>].

Optional Arguments

no: A Boolean. Set to `TRUE` to remove the specified confederation ID.

Returns

`TRUE` on success.

Usage Guidelines

The BGP confederation identifier is used to configure a single AS number to identify a group of smaller ASes as a single confederation.

A confederation can be used to reduce the internal BGP (iBGP) mesh by dividing a large single AS into multiple subASes and then grouping them into a single confederation. The subASes within the confederation exchange routing information like iBGP peers. External peers interact with the confederation as if it were a single AS.

Each subAS is fully meshed within itself and has a few connections to other ASes within the confederation. Next-hop, MED, and local preference information is preserved throughout the confederation, allowing you to retain a single Interior Gateway Protocol (IGP) for all ASes.

cfg_confederation_peers(*peers*, *******kwargs*)

Configures subASes to belong to a single confederation. To remove an AS from the confederation, set the optional *no* argument to `TRUE`.

Arguments

peers: A string of space-separated AS numbers where each `ASN=<1-4294967295>|<1-65535>[.<0-65535>]`.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove the specified AS numbers from the list of confederation peers.

Returns

`TRUE` on success.

Send document comments to nexus3k-docfeedback@cisco.com.

Usage Guidelines

The **bgp confederation peers** command is used to configure multiple ASes as a single confederation. The ellipsis (...) in the command syntax indicates that your command input can include multiple values for the *as-number* argument.

The ASes specified in this command are visible internally to the confederation. Each AS is fully meshed within itself. The **bgp confederation identifier** command specifies the confederation to which the ASes belong.

cfg_dampening(**kwargs)

Enable BGP route dampening or change various BGP route dampening factors. To disable the function, set the optional *no* argument to `TRUE`.

Optional Arguments

half_life: Time (in minutes) after which a penalty is decreased. Once the route has been assigned a penalty, the penalty is decreased by half after the half-life period (which is 15 minutes by default). The process of reducing the penalty happens every 5 seconds. The range of the half-life period is 1 to 45 minutes. The default is 15 minutes.

reuse_limit: Value to start reusing a route. The range is from 1 to 20000.

suppress_limit: Value to start suppressing a route. The range is from 1 to 20000.

max_suppress_time: Maximum suppress time for a stable route. The range is from 1 to 255.

route_map: Name of a route map that specifies dampening criteria. The name can be any alphanumeric string up to 63 characters.

no: Set to `TRUE` to disable the dampening feature.

Returns

`TRUE` on success.

Usage Guidelines

You can configure route dampening to minimize route flaps propagating through your iBGP network.

cfg_distance(*ebgp_dist*, *ibgp_dist*, *local_dist*, **kwargs)

Configure administrative distance for external BGP, internal BGP, and local routes. Default values are: eBGP 20, iBGP 200, and local 220. To set the distances back to the default, set the optional *no* argument to `TRUE`.

Arguments

ebgp_dist: Distance for eBGP routes. An integer ranging from 1 to 255.

ibgp_dist: Distance for iBGP routes. An integer ranging from 1 to 255.

local_dist: Distance for local routes. An integer ranging from 1 to 255.

Optional Arguments

no: Set to `TRUE` to set the distances back to the default values.

Returns

`TRUE` on success.

Send document comments to nexus3k-docfeedback@cisco.com.

cfg_events_event_history(**kwargs)

Enable the event history to be saved in a buffer. It is enabled by default. The default buffer size is ‘small’. Possible sizes are ‘small’, ‘medium’, and ‘large’.

Optional Arguments

size: A string. Possible values are: ‘small’, ‘medium’, ‘large’, or ‘disable’.

no: A Boolean. Set to `TRUE` to stop saving the event history.

Returns

`TRUE` on success.

cfg_graceful_restart_restart_time(time, **kwargs)

Configure the maximum time for restart that is advertised to peers. Default value is 120 seconds. To remove a previously configured value, set the optional *no* argument to `TRUE`.

Arguments

time: An integer ranging from 1 to 3600 representing the restart time in seconds.

Optional Arguments

no: A Boolean. Set to `TRUE` to delete the existing configuration and revert to the default.

Returns

`TRUE` on success.

cfg_graceful_restart_stalepath_time(time, **kwargs)

Configure the maximum time to keep a restarting peer’s stale routes. Default value is 300 seconds. To remove a previously configured value, set the optional *no* argument to `TRUE`.

Arguments

time: An integer ranging from 1 to 3600 representing the stale path time in seconds.

Optional Arguments

no: A Boolean. Set to `TRUE` to delete the existing configuration and revert to the default.

Returns

`TRUE` on success.

cfg_ibgp_maximum_paths(max, **kwargs)

Configure the maximum number of parallel routes that the iBGP can support. To restore the default number of parallel routes, set the optional *no* argument to `TRUE`.

Arguments

max: Maximum number of parallel routes that an IP routing protocol installs in a routing table. The range is from 1 to 64.

Optional Arguments

no: Set to `TRUE` to restore the default number of parallel routes.

Send document comments to nexus3k-docfeedback@cisco.com.

Returns

TRUE on success.

cfg_maximum_paths(*max*, *******kwargs*)

Configure the maximum number of parallel routes that the BGP can support. To restore the default number of parallel routes, set the optional *no* argument to TRUE.

Arguments

max: Maximum number of parallel routes that an IP routing protocol installs in a routing table. The range is from 1 to 64.

Optional Arguments

no: Set to TRUE to restore the default number of parallel routes.

Returns

TRUE on success.

cfg_nexthop_routemap(*route_map*, *******kwargs*)

Specify that BGP routes are resolved using only next-hops whose routes match specific characteristics. To remove the route map, set the optional *no* argument to TRUE.

Arguments

route_map: Route map name. The name can be any alphanumeric string up to 63 characters.

Optional Arguments

no: A Boolean. Set to TRUE to remove a route map.

Returns

TRUE on success.

Usage Guidelines

Use this function to configure route policy filtering for next-hops. BGP next-hop filtering allows you to specify that when a next-hop address is checked with the Routing Information Base (RIB), the underlying route for that next-hop address is passed through the route map. If the route map rejects the route, the next-hop address is treated as unreachable. BGP marks all next-hops that are rejected by the route policy as invalid and does not calculate the best-path for the routes that use the invalid next-hop address.

cfg_nexthop_trigger_delay(*critical_delay*=3000, *non_critical_delay*=10000, *******kwargs*)

Configure BGP to delay for triggering next-hop calculations. To set the trigger delay to the default value, set the optional *no* argument to TRUE.

Arguments

critical_delay: An integer specifying the critical next-hop trigger delay in milliseconds. The range is from 1 to 4294967295. The default is 3000.

non_critical_delay: An integer specifying the noncritical next-hop trigger delay in milliseconds. The range is from 1 to 4294967295. The default is 10000.

Send document comments to nexus3k-docfeedback@cisco.com.

Optional Arguments

no: Set to TRUE to set the trigger delay to the default value.

Returns

TRUE on success.

Usage Guidelines

Use this function to modify when BGP processes next-hop address tracking events. The non-critical delay value must always be equal to or greater than the critical delay value. The delay should be slightly higher than the time it takes for the Interior Gateway Protocol (IGP) to settle into a steady state after some event (IGP convergence time).

cfg_periodic_event_history(**kwargs)

Enable periodic event history to be saved in a buffer. It is enabled by default. The default buffer size is 'small'. Possible sizes are 'small', 'medium', and 'large'.

Optional Arguments

size: A string. Possible values are: 'small', 'medium', 'large', or 'disable'.

no: A Boolean. Set to TRUE to stop saving the periodic event history.

Returns

TRUE on success.

cfg_router_id(*router_id*, **kwargs)

Specify the IP address to use as the router ID. To remove this configuration, set the optional *no* argument to TRUE.

Arguments

router_id: A string in dotted quad format ('A.B.C.D') representing the IP address of the router.

Optional Arguments

no: A Boolean. Set to TRUE to remove the *router-id*.

Returns

TRUE on success.

client_to_client_route_reflection(**kwargs)

Configure client-to-client route reflection. This feature is enabled by default. This command triggers an automatic soft-clear or refresh of BGP neighbor sessions. To disable this feature, set the optional *no* argument to TRUE.

Optional Arguments

no: A Boolean. Set to TRUE to disable client-to-client route reflection.

Returns

TRUE on success.

Send document comments to nexus3k-docfeedback@cisco.com.

enforce_first_as(**kwargs)

Configure a router to deny an update received from an external BGP (eBGP) peer that does not list its AS number at the beginning of the AS_PATH in the incoming update. To disable this behavior, set the optional *no* argument to TRUE.

Optional Arguments

no: A Boolean. Set to TRUE to disable this feature.

Returns

TRUE on success.

Usage Guidelines

The **bgp enforce-first-as** command is used to deny incoming updates received from eBGP peers that do not list their AS number as the first segment in the AS_PATH attribute. Enabling this command prevents a misconfigured or unauthorized peer from misdirecting traffic (spoofing the local router) by advertising a route as if it were sourced from another AS.

fast_external_fallover(**kwargs)

Configure the BGP process to immediately reset the session if the link to a directly connected BGP peer goes down. This is enabled by default. To disable this behavior, set the optional *no* argument to TRUE.

Optional Arguments

no: A Boolean. Set to TRUE to disable this feature.

Returns

TRUE on success.

flush_routes(**kwargs)

Flush routes in the RIB upon a controlled restart. To disable this behavior, set the optional *no* argument to TRUE.

Optional Arguments

no: A Boolean. Set to TRUE to disable this feature.

Returns

TRUE on success.

graceful_restart(**kwargs)

Enable graceful restart and graceful restart helper functionality. This is enabled by default. To disable this behavior, set the optional *no* argument to TRUE.

Optional Arguments

no: A Boolean. Set to TRUE to disable this feature.

Returns

TRUE on success.

Send document comments to nexus3k-docfeedback@cisco.com.

Usage Guidelines

The **graceful-restart** command is used to configure or disable the graceful restart capability on a router in a BGP network. If the graceful restart capability is enabled after a BGP session has been established, you need to restart the session with a soft or hard reset.

The default timer values for this feature are optimal for most network deployments. We recommend that they are adjusted only by experienced network operators. When adjusting the timer values, the restart timer should not be set to a value greater than the hold time that is carried in the `OPEN` message. If consecutive restart operations occur, routes (from a restarting router) that were previously marked as stale are deleted.

graceful_restart_helper(**_kwargs_)

Configure graceful restart helper mode functionality. To disable this behavior, set the optional *no* argument to `TRUE`.

Optional Arguments

no: A Boolean. Set to `TRUE` to disable this feature.

Returns

`TRUE` on success.

Usage Guidelines

The **graceful-restart-helper** command is used to configure the local BGP router to support the graceful restart of a remote BGP peer.

is_shutdown()

Check if the BGP routing process is shut down.

Arguments

None

Returns

`TRUE` if the BGP process is enabled and shut down.

`FALSE` if the BGP process is running or if BGP is not enabled.

log_neighbor_changes(**_kwargs_)

Log a message for a neighbor up/down event. To disable this behavior, set the optional *no* argument to `TRUE`.

Optional Arguments

no: A Boolean. Set to `TRUE` to disable this feature.

Returns

`TRUE` on success.

Send document comments to nexus3k-docfeedback@cisco.com.

max_as_limit(*limit*, ******kwargs*)

Allow the AS_PATH attribute from an eBGP neighbor, imposing a limit on the number of ASes. To disable this behavior, set the optional *no* argument to TRUE.

Arguments

limit: An integer ranging from 1 to 512 representing the number of ASes in the AS_PATH attribute.

Optional Arguments

no: A Boolean. Set to TRUE to disable this feature.

Returns

TRUE on success.

redistribute_direct_routes(*route_map*, ******kwargs*)

Inject routes that are directly connected on an interface into the BGP. To restore the system to its default condition in which the software does not redistribute routes, set the optional *no* argument to TRUE.

Arguments

route_map: An alphanumeric string up to 63 characters specifying the identifier of a configured route map. Use a route map to filter which routes are redistributed into BGP.

Optional Arguments

no: A Boolean. Set to TRUE to remove the configuration.

Returns

TRUE on success.

redistribute_eigrp_routes(*instance_tag*, *route_map*, ******kwargs*)

Inject routes from the Enhanced Interior Gateway Routing Protocol (EIGRP) into the BGP. To restore the system to its default condition in which the software does not redistribute routes, set the optional *no* argument to TRUE.

Arguments

instance_tag: Any case-sensitive, alphanumeric string up to 64 characters.

route_map: Specifies the identifier of a configured route map. Use a route map to filter which routes are redistributed into BGP.

Optional Arguments

no: A Boolean. Set to TRUE to remove the configuration.

Returns

TRUE on success.

redistribute_isis_routes(*instance_tag*, *route_map*, ******kwargs*)

Inject routes from the Intermediate System to Intermediate System (IS-IS) protocol into the BGP. To restore the system to its default condition in which the software does not redistribute routes, set the optional *no* argument to TRUE.

Send document comments to nexus3k-docfeedback@cisco.com.

Arguments

instance_tag: Any case-sensitive, alphanumeric string up to 64 characters.

route_map: Specifies the identifier of a configured route map. Use a route map to filter which routes are redistributed into BGP.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove the configuration.

Returns

`TRUE` on success.

redistribute_ospf_routes(*protocol_tag*, *route_map*, ***kwargs*)

Inject routes from the Open Shortest Path First (OSPF) protocol into the BGP. To restore the system to its default condition in which the software does not redistribute routes, set the optional *no* argument to `TRUE`.

Arguments

protocol_tag: Any case-sensitive, alphanumeric string up to 64 characters.

route_map: Specifies the identifier of a configured route map. Use a route map to filter which routes are redistributed into BGP.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove the configuration.

Returns

`TRUE` on success.

redistribute_rip_routes(*instance_tag*, *route_map*, ***kwargs*)

Inject routes from the Routing Information Protocol (RIP) protocol into the BGP. To restore the system to its default condition in which the software does not redistribute routes, set the optional *no* argument to `TRUE`.

Arguments

instance_tag: Any case-sensitive, alphanumeric string up to 64 characters.

route_map: Specifies the identifier of a configured route map. Use a route map to filter which routes are redistributed into BGP.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove the configuration.

Returns

`TRUE` on success.

redistribute_static_routes(*route_map*, ***kwargs*)

Inject static routes into the BGP. To restore the system to its default condition in which the software does not redistribute routes, set the optional *no* argument to `TRUE`.

Send document comments to nexus3k-docfeedback@cisco.com.

Arguments

route_map: Specifies the identifier of a configured route map. Use a route map to filter which routes are redistributed into BGP.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove the configuration.

Returns

`TRUE` on success.

set_addr_family(*ip_version*, *transmission_type*)

Enable an address family for use with BGP. Possible address families are ipv4 unicast, ipv4 multicast, ipv6 unicast, and ipv6 multicast. This function sets the current address-family for use with functions that configure address-family attributes.

Arguments

ip_version: A string. Possible values are ‘ipv4’ and ‘ipv6’.

transmission_type: A string. Possible values are ‘unicast’ and ‘multicast’.

Returns

`ValueError` if an invalid *ip_version* or *transmission_type* is specified.

`TRUE` on success.

set_default_metric(*metric*, ****kwargs*)

Set the metric of redistributed routes. To remove this configuration, set the optional *no* argument to `TRUE`.

Arguments

metric: The metric. An integer ranging from 0 to 4294967295.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove any existing configuration.

Returns

`TRUE` on success.

set_vrf(*vrf*)

Set the VRF (Virtual Routing and Forwarding) context for subsequent API calls on this BGPSession object. Any configuration done on this BGPSession object is applied to this VRF.

Arguments

vrf: VRF name (string) or the VRF ID (integer).

Returns

None

Send document comments to nexus3k-docfeedback@cisco.com.

shutdown()

Shut down the BGP routing process. All existing BGP configurations are preserved.

Arguments

None

Returns

`TRUE` on success.

start()

Start the BGP routing process. If BGP is shut down, this restarts it. If the BGP feature is disabled, this enables the feature in addition to starting the process.

Arguments

None

Returns

`TRUE` on success.

suppress_inactive(**_kwargs_)

Configure the BGP process to advertise active routes to a BGP peer only. To remove the restriction, set the optional *no* argument to `TRUE`.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove the configuration.

Returns

`TRUE` on success.

Static methods defined here:

is_valid_asn(*asn*)

Data and other attributes defined here:

`BGPNeighbor = <class 'cisco.BGPNeighbor'>`

Use this class to configure a BGP neighbor.

Methods inherited from `Feature`:

disable()

Disable feature.

Send document comments to nexus3k-docfeedback@cisco.com.

enable(**kwargs)

Start feature.

Arguments

no=TRUE: Stops the Terminal Access Controller Access-Control System (TACACS).

Returns

TRUE on success.

is_enabled()

Returns

TRUE if the feature is enabled.

name()

Feature name—as used in the **configure terminal** command.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return the state (or states if multiple instances) of Feature.

Static methods inherited from Feature:

__new__(*typ*, **args*, ***kwargs*)

Create a single instance of an object per each derived class.

class BGPNeighbor(*ip_address*, *vrf*= 'default', ***kwargs*)

Use this class to configure a BGP neighbor. This class is an attribute of the BGPSession class.

Methods defined here:

add()

Add a BGP neighbor to the BGP configuration.

Arguments

None

Returns

TRUE on success.

Send document comments to nexus3k-docfeedback@cisco.com.

allow_as_in(**kwargs)

Configure BGP to accept AS paths with this neighbor's AS present.

To remove this configuration, set the optional *no* argument to `TRUE`.

Optional Arguments

num_occurrences: Number of occurrences of AS number., an integer ranging from 1 to 10.

no: A Boolean. Set to `TRUE` to remove this configuration.

Returns

`TRUE` on success.

cfg_advertise_map(*map*, **kwargs)

Configure Border Gateway Protocol (BGP) conditional advertisement. To remove a BGP conditional advertisement, set the optional *no* argument to `TRUE`.

Arguments

map: Route map with match statements that the route must pass before BGP passes the route to the next route map. The map is a case-sensitive, alphanumeric string up to 63 characters.

Optional Arguments

exist_map: Specifies a route map with match statements for a prefix list. A prefix in the BGP table must match a prefix in the prefix list before BGP advertises the route. *exist_map* is a case-sensitive, alphanumeric string up to 63 characters.

non_exist_map: Specifies a route map without match statements for a prefix list. A prefix in the BGP table must not match a prefix in the prefix list before BGP advertises the route. *non_exist_map* is a case-sensitive, alphanumeric string up to 63 characters.

no: A Boolean. Set to `TRUE` to remove a BGP conditional advertisement.

Returns

`TRUE` on success.

cfg_ebgp_multihop(*ebgp_ttl*=None, **kwargs)

Accepts and attempts BGP connections to external peers that reside on networks that are not directly connected.

Arguments

ebgp_ttl: An integer. The multihop TTL value. Acceptable values are 2 to 255.

Optional Arguments

no: A Boolean. Set to true to remove this configuration.

Returns

`TRUE` on success.

cfg_filter_list_in(*filter*, **kwargs)

Apply the `AS_PATH` filter list to incoming routes. To remove this configuration, set the optional *no* argument to `TRUE`.

Send document comments to nexus3k-docfeedback@cisco.com.

Arguments

filter: Name of the filter list. An alphanumeric string up to 63 characters.

Optional Arguments

no: A Boolean. Set to TRUE to remove this configuration.

Returns

TRUE on success.

cfg_filter_list_out(*filter*, *******kwargs*)

Apply AS_PATH filter list to outgoing routes. To remove this configuration, set the optional *no* argument to TRUE.

Arguments

filter: Name of filter list. An alphanumeric string up to 63 characters.

Optional Arguments

no: A Boolean. Set to TRUE to remove this configuration.

Returns

TRUE on success.

cfg_local_as(*ASN*=None, *******kwargs*)

Configure a router to appear as a member of a second AS in addition to the real AS of the device. To remove the local AS configuration, set the optional *no* argument to TRUE.

Arguments

ASN: A string or integer representing the Autonomous System Number. If an integer, range is from 1 to 4294967295. If a string, it should be in this format <1-4294967295>|<1-65535>[.<0-65535>].

Optional Arguments

no_prepend: A Boolean. Set to TRUE to prevent prepending the local AS number to any routes received from the eBGP neighbor.

replace_as: A Boolean. Set to TRUE to prepend only the local AS number to updates to the eBGP neighbor.

dual_as: A Boolean. Set to TRUE to configure the eBGP neighbor to establish a peering session using the real ASN (from the local BGP routing process) or by using the ASN.

no: A Boolean. Set to TRUE to remove the local AS configuration.

Returns

TRUE on success.

cfg_maximum_prefix(*limit*=None, *******kwargs*)

Configure the maximum number of prefixes from this neighbor. To remove this configuration, set the optional *no* argument to TRUE.

Arguments

limit: Max prefix limit. An integer ranging from 1 to 300,000.

Send document comments to nexus3k-docfeedback@cisco.com.

Optional Arguments

threshold: Threshold percentage at which to generate a warning. An integer ranging from 1 to 100.

restart_interval: Restart the BGP connection after the limit is exceeded. An integer ranging from 1 to 65535.

warning_only: A Boolean. Set to TRUE to only give a warning message when the limit is exceeded.

no: A Boolean. Set to `TRUE` to remove this configuration.

Returns

`TRUE` on success.

cfg_password(*password*=None, ***kwargs*)

Configure a password for this BGP neighbor. To remove a password, set the optional *no* argument to `TRUE`.

Arguments

password: A `CiscoSecret` object.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove an existing password.

Returns

`TRUE` on success.

cfg_prefix_list_in(*list*, ***kwargs*)

Apply a prefix-list to incoming routes. Set the optional *no* argument to `TRUE` to remove this configuration.

Arguments

list: Name of the prefix-list. An alphanumeric string up to 63 characters.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove this configuration.

Returns

`TRUE` on success.

cfg_prefix_list_out(*list*, ***kwargs*)

Apply a prefix-list to outgoing routes. Set the optional *no* argument to `TRUE` to remove this configuration.

Arguments

list: Name of prefix-list. An alphanumeric string up to 63 characters.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove this configuration.

Returns

`TRUE` on success.

Send document comments to nexus3k-docfeedback@cisco.com.

cfg_remote_as(*ASN*=None, **kwargs)

Specify the AS number for this neighbor. To remove an AS number set the optional *no* argument to TRUE.

Arguments

ASN: A string or integer representing the ASN. If integer, range is from 1 to 4294967295. If string, it should be in this format: '<1-4294967295>|<1-65535>[.<0-65535>]'.

Optional Arguments

no: A Boolean. Set to TRUE to remove an AS number.

Returns

TRUE on success.

cfg_route_map_in(*map*, **kwargs)

Apply a route-map to incoming routes. Set the optional *no* argument to TRUE to remove this configuration.

Arguments

map: Name of the route-map. An alphanumeric string up to 63 characters.

Optional Arguments

no: A Boolean. Set to TRUE to remove this configuration.

Returns

TRUE on success.

cfg_route_map_out(*map*, **kwargs)

Apply a route-map to outgoing routes. Set the optional *no* argument to TRUE to remove this configuration.

Arguments

map: Name of the route-map. An alphanumeric string up to 63 characters.

Optional Arguments

no: A Boolean. Set to TRUE to remove this configuration.

Returns

TRUE on success.

cfg_timers(*keepalive_interval*=60, *holdtime*=180, **kwargs)

Configure keepalive and hold timers in seconds. Default values are 60 seconds for keepalive and 180 seconds for hold time. To set back to the default values, set the optional *no* argument to TRUE.

Arguments

keepalive_interval: An integer ranging from 0 to 3600.

holdtime: An integer ranging from 0 3600.

Send document comments to nexus3k-docfeedback@cisco.com.

Optional Arguments

no: A Boolean. Set to TRUE to set the timers back to the default values.

Returns

TRUE on success.

cfg_update_source(*interface*=None, **kwargs)

Specify the source of the BGP session and updates. The Cisco NX-OS software allows BGP sessions to use any operational interface for TCP connections. To restore the interface assignment to the closest interface, which is called the best local address, set the optional *no* argument to TRUE.

Arguments

interface: A string representing an interface on the switch. Can be Ethernet, loopback, port-channel or a VLAN in either expanded or short forms. For example, ‘e1/1.2’, ‘Ethernet1/1.2’.

Optional Arguments

no: A Boolean. Set to TRUE to restore the interface assignment to the best local address.

Returns

TRUE on success.

default_originate(**kwargs)

Originate a default toward this neighbor. To remove this configuration, set the optional *no* argument to TRUE.

Optional Arguments

route_map: Name of route-map that specifies criteria for the originating default.

no: A Boolean. Set to TRUE to remove this configuration.

Returns

TRUE on success.

disable_connected_check(**kwargs)

Disable the connection verification for the directly connected peer.

Arguments

None

Optional Arguments

no: A Boolean. Set to TRUE to enable the connection verification for the directly connected peer.

Returns

TRUE on success.

Send document comments to nexus3k-docfeedback@cisco.com.

Usage Guidelines

Use this function to disable a check for an eBGP peer that is directly connected to the local router. BGP triggers a connection check automatically for all eBGP peers that are known to be a single hop away, unless you disable this check with this function. BGP does not bring up sessions if the check fails. BGP considers an eBGP peer as a single hop away if the eBGP peer does not have the ebgp-multihop option configured (that is, the time-to-live (TTL) value is one).

dont_capability_negotiate(**_kwargs_)

Turn off the negotiate capability with this neighbor.

Arguments

None

Optional Arguments

no: A Boolean. Set to `TRUE` to turn on the negotiate capability with this neighbor.

Returns

`TRUE` on success.

dynamic_capability(**_kwargs_)

Enable the dynamic capability.

Arguments

None

Optional Arguments

no: A Boolean. Set to `TRUE` to disable the dynamic capability.

Returns

`TRUE` on success.

exists()

Check if this BGP neighbor exists.

Arguments

None

Returns

`TRUE` if this BGP neighbor exists.

`FALSE` if this BGP neighbor does not exist.

low_memory_exempt(**_kwargs_)

Exempt this BGP neighbor from a low-memory shutdown. To make this BGP neighbor eligible for a low-memory shutdown, set the optional *no* argument to `TRUE`.

Arguments

None

Send document comments to nexus3k-docfeedback@cisco.com.

Optional Arguments

no: A Boolean. Set to `TRUE` to make this BGP neighbor eligible for a low-memory shutdown.

Returns

`TRUE` on success.

next_hop_self(**_kwargs_)

Set our peering address as nexthop. To remove this configuration, set the optional *no* argument to `TRUE`.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove this configuration.

Returns

`TRUE` on success.

next_hop_third_party(**_kwargs_)

Compute a third-party nexthop if possible. To remove this configuration, set the optional *no* argument to `TRUE`.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove this configuration.

Returns

`TRUE` on success.

remove()

Remove the BGP neighbor from the BGP configuration.

Arguments

None

Returns

`TRUE` on success.

`ValueError` if the neighbor does not exist.

remove_private_as(**_kwargs_)

Remove the private AS number from outbound updates. To include the private AS number in outbound updates, set the optional *no* argument to `TRUE`.

Arguments

None

Optional Arguments

no: A Boolean. Set to `TRUE` to include the private AS number in outbound updates.

Returns

`TRUE` on success.

Send document comments to nexus3k-docfeedback@cisco.com.

route_reflector_client(**kwargs)

Configure this neighbor as a route reflector client. To remove this configuration, set the optional *no* argument to TRUE.

Optional Arguments

no: A Boolean. Set to TRUE to remove this configuration.

Returns

TRUE on success.

send_community(**kwargs)

Send the BGP community attribute to this neighbor. To revert back to the defaults, set the optional *no* argument to TRUE.

Optional Arguments

no: A Boolean. Set to TRUE to revert back to the defaults.

Returns

TRUE on success.

send_community_extended(**kwargs)

Send the BGP extended community attribute to this neighbor. To revert back to the defaults, set the optional *no* argument to TRUE.

Optional Arguments

no: A Boolean. Set to TRUE to revert back to the defaults.

Returns

TRUE on success.

set_addr_family(*ip_version*, *transmission_type*)

set_description(*description*=None, **kwargs)

Set a descriptive string for this BGP neighbor.

Arguments

description: An alphanumeric string up to 80 characters long.

Optional Arguments

no: A Boolean. Set to TRUE to remove any existing description.

Returns

TRUE on success.

Send document comments to nexus3k-docfeedback@cisco.com.

shutdown(**kwargs)

Administratively shut down this neighbor. All existing configurations are preserved. To enable this neighbor, set the optional *no* argument to TRUE.

Arguments

None

Optional Arguments

no: A Boolean. Set to TRUE to enable this neighbor.

Returns

TRUE on success.

soft_reconfiguration_inbound(**kwargs)

Configure the switch software to start storing BGP peer updates. To not store received updates, set the optional *no* argument to TRUE.

Optional Arguments

no: A Boolean. Set to TRUE to stop storing received updates.

Returns

TRUE on success.

Usage Guidelines

Entering this command starts the storage of updates, which is required to do inbound soft reconfiguration. To use soft reconfiguration, or soft reset, without preconfiguration, both BGP peers must support the soft route refresh capability.

suppress_inactive(**kwargs)

Advertise the active routes to a BGP peer only. To remove the restriction, set the optional *no* argument to TRUE.

Optional Arguments

no: A Boolean. Set to TRUE to remove this restriction.

Returns

TRUE on success.

transport_connection_mode_passive(**kwargs)

Allows a passive connection setup only. To remove this restriction, set the optional *no* argument to TRUE.

Arguments

None

Optional Arguments

no: A Boolean. Set to TRUE to allow all connection setups.

Returns

Send document comments to nexus3k-docfeedback@cisco.com.

TRUE on success.

Data descriptors defined here:

__dict__

Dictionary for instance variables (if defined).

__weakref__

List of weak references to the object (if defined).

class BufferDepthMonitor()

Method resolution order:

BufferDepthMonitor

CLI

__builtin__.object

Methods defined here:

dumps()

get_max_cell_usage()

get_remaining_instant_usage()

get_status()

get_switch_cell_count()

get_total_instant_usage()

parse_specific()

Methods inherited from CLI:

get_command()

get_output()

Send document comments to nexus3k-docfeedback@cisco.com.

get_raw_output()

get_timestamp()

get_xml_dom_from_cli_output(*text*)

key_map(*key*)

key_value_colon_parser(*line*)

key_value_xml_parser(*element*)

rerun()

Data descriptors inherited from `CLI`:

`__dict__`

Dictionary for instance variables (if defined).

`__weakref__`

List of weak references to the object (if defined).

class CLI(*command*='', *do_print*=TRUE)

Generic CLI base class with useful utilities.

Methods defined here:

get_command()

get_output()

get_raw_output()

get_status()

get_timestamp()

Send document comments to nexus3k-docfeedback@cisco.com.

get_xml_dom_from_cli_output(*text*)

key_map(*key*)

key_value_colon_parser(*line*)

key_value_xml_parser(*element*)

parse_specific()

rerun()

Data descriptors defined here:

`__dict__`

Dictionary for instance variables (if defined).

`__weakref__`

List of weak references to the object (if defined).

class CheckPortDiscards(*port*)

Method resolution order:

`CheckPortDiscards`

CLI

`__builtin__.object`

Methods defined here:

parse_specific()

Methods inherited from `CLI`:

get_command()

get_output()

Send document comments to nexus3k-docfeedback@cisco.com.

get_raw_output()

get_status()

get_timestamp()

get_xml_dom_from_cli_output(*text*)

key_map(*key*)

key_value_colon_parser(*line*)

key_value_xml_parser(*element*)

rerun()

Data descriptors inherited from `CLI`:

`__dict__`

Dictionary for instance variables (if defined).

`__weakref__`

List of weak references to the object (if defined).

class CiscoSecret(*key*, *type*=0)

Cisco password container keytype:

0—cleartext.

5—encrypted (stronger, not all CLIs support it).

7—encrypted.

Methods defined here:

get_key()

get_key_type()

Send document comments to nexus3k-docfeedback@cisco.com.

set(*key*, *type*=0)

Data descriptors defined here:

`__dict__`

Dictionary for instance variables (if defined).

`__weakref__`

List of weak references to the object (if defined).

class CiscoSocket([*family*[, *type*[, *proto*]]])

Extend the `socket.socket` class in order to provide a way to set/get the Virtual Routing and Forwarding (VRF) associated with a socket. The default VRF for a `CiscoSocket` is 'management' (2).

Method resolution order:

`CiscoSocket`

`socket._socketobject`

`__builtin__.object`

Methods defined here:

get_vrf()

Get the VRF associated with a `CiscoSocket`.

Arguments

None

Returns

VRF name as a string.

Example

```
s = CiscoSocket(socket.AF_INET, socket.SOCK_STREAM)
s.get_vrf()
```

set_vrf(*vrf*)

Set the VRF on a `CiscoSocket`. The default VRF for a `CiscoSocket` is 'management' (2).

Arguments

vrf: VRF name (string) or the VRF ID (int).

Returns

`TRUE` on success.

Example

```
a. s = CiscoSocket(socket.AF_INET, socket.SOCK_STREAM)
   s.set_vrf(3)
b. s = CiscoSocket(socket.AF_INET, socket.SOCK_STREAM)
```

Send document comments to nexus3k-docfeedback@cisco.com.

```
   s.set_vrf('floor1')
----------------------------------------------------------------------
```

Data descriptors defined here:

`__dict__`

Dictionary for instance variables (if defined).

Methods inherited from `socket._socketobject`:

accept()

```
accept() -> (socket object, address info)
```

Wait for an incoming connection. Return a new socket representing the connection and the address of the client. For IP sockets, the address info is a pair (*hostaddr*, *port*).

bind(...)

`bind(`*address*`)`

Bind the socket to a local address. For IP sockets, the address is a pair (*host*, *port*); the host must refer to the local host. For raw packet sockets, the address is a tuple `(`*ifname*`, `*proto*` [,`*pkttype*` [,`*hatype*`]])`.

close(*_closedsocket*=<class 'socket._closedsocket'>, *_delegate_methods*=('recv', 'recvfrom', 'recv_into', 'recvfrom_into', 'send', 'sendto'), *setattr*=<built-in function setattr>)

close()

Close the socket. It cannot be used after this call.

connect(...)

`connect(`*address*`)`

Connect the socket to a remote address. For IP sockets, the address is a pair (*host*, *port*).

connect_ex(...)

`connect_ex(`*address*`) -> errno`

This is like `connect(`*address*`)`, but returns an error code (the `errno` value) instead of raising an exception when an error occurs.

dup()

```
dup() -> socket object
```

Send document comments to nexus3k-docfeedback@cisco.com.

Return a new socket object connected to the same system resource.

fileno(...)

`fileno() -> integer`

Return the integer file descriptor of the socket.

getpeername(...)

`getpeername() -> address info`

Return the address of the remote endpoint. For IP sockets, the address info is a pair (*hostaddr*, *port*).

getsockname(...)

`getsockname() -> address info`

Return the address of the local endpoint. For IP sockets, the address info is a pair (*hostaddr*, *port*).

getsockopt(...)

`getsockopt(`*level*`, `*option*`[, `*buffersize*`]) -> value`

Get a socket option. See the Unix manual for *level* and *option*. If a nonzero *buffersize* argument is given, the return value is a string of that length; otherwise it is an integer.

gettimeout(...)

`gettimeout() -> timeout`

Return the timeout in floating seconds associated with socket operations. A timeout of `None` indicates that timeouts on socket operations are disabled.

listen(...)

`listen(`*backlog*`)`

Enable a server to accept connections. The backlog argument must be at least 0 (if it is lower, it is set to 0); it specifies the number of unaccepted connections that the system allows before refusing new connections.

makefile(*mode*='r', *bufsize*=-1)

`makefile([`*mode*`[, `*bufsize*`]]) -> file object`

Return a regular file object corresponding to the socket. The *mode* and *bufsize* arguments are as for the `built-in open()` function.

sendall(...)

`sendall(`*data*`, [`*flags*`])`

Send document comments to nexus3k-docfeedback@cisco.com.

Send a data string to the socket. For the optional *flags* argument, see the Unix manual. This function calls `send()` repeatedly until all data is sent. If an error occurs, it is impossible to tell how much data has been sent.

setblocking(...)

`setblocking(`*flag*`)`

Set the socket to blocking (*flag* is `TRUE`) or non-blocking (`FALSE`). `setblocking(TRUE)` is equivalent to `settimeout(None)`; `setblocking(FALSE)` is equivalent to `settimeout(0.0)`.

setsockopt(...)

`setsockopt(`*level*`, `*option*`, `*value*`)`

Set a socket option. See the Unix manual for *level* and *option*. The value argument can either be an integer or a string.

settimeout(...)

`settimeout(`*timeout*`)`

Set a timeout on socket operations. *timeout* can be a float, giving in seconds, or `None`. Setting a timeout of `None` disables the timeout feature and is equivalent to `setblocking(1)`. Setting a timeout of zero is the same as `setblocking(0)`.

shutdown(...)

`shutdown(`*flag*`)`

Shut down the reading side of the socket (*flag* == `SHUT_RD`), the writing side of the socket (*flag* == `SHUT_WR`), or both ends (*flag* == `SHUT_RDWR`).

Data descriptors inherited from `socket._socketobject`:

`__weakref__`

List of weak references to the object (if defined).

`family`

The socket family

`proto`

The socket protocol

`recv`

`recv_into`

`recvfrom`

`recvfrom_into`

`send`

`sendto`

`type`

Send document comments to nexus3k-docfeedback@cisco.com.

The socket type

class FTPTransfer(*source*, *dest*='', *host*=(None, None), *vrf*='management', *login_timeout*=10, *user*='', *password*='')

Method resolution order:

`FTPTransfer`

`PasswordProtoTransfer`

`Transfer`

`__builtin__.object`

Methods defined here:

protoprocessresponse(*i*)

protosetup()

Methods inherited from `PasswordProtoTransfer`:

baseprocessresponse(*i*)

geturi()

inputvalidation()

processresponse(*i*)

setusercredentials(*user*, *password*)

Methods inherited from `Transfer`:

find_local_filename(*filename*='')

getdest()

Send document comments to nexus3k-docfeedback@cisco.com.

gethost()

getsource()

getstatus()

getswitchname()

gettimeout()

getvrf()

local_file_exist(*filename=''***)**

postvalidation()

run()

setdest(*dest***)**

sethost(*host***)**

setsource(*source***)**

settimeout(*login_timeout***)**

setup(*source***, *dest*, *host*, *vrf*, *login_timeout*)**

setvrf(*vrf***)**

transferstatus()

Send document comments to nexus3k-docfeedback@cisco.com.

Static methods inherited from `Transfer`:

gettransferobj(*protocol*='', *host*='', *source*='', *dest*='bootflash:', *vrf*='management', *login_timeout*=10, *user*='', *password*='')

Data descriptors inherited from `Transfer`:

`__dict__`

Dictionary for instance variables (if defined).

`__weakref__`

List of weak references to the object (if defined).

class Feature(__builtin__.object)

An abstract base class for Nexus OS Features. It has one get() class method that returns a singleton object per feature.

Arguments to `cisco.Feature.get()` can be one of the following strings:

```
bfd
bgp
dhcp
eigrp
hsrp
interface-vlan
lacp
msdp
ospf
ospfv3
pim
private-vlan
ptp
rip
scheduler
ssh
tacacs+
telnet
udld
vpc
vrrp
vtp
```

Send document comments to nexus3k-docfeedback@cisco.com.

If called multiple times with the same feature name, `cisco.feature.get()` returns the same object.

class HTTPTransfer(*source*, *dest*='', *host*=(None, None), *vrf*='management', *login_timeout*=10)

Method resolution order:

`HTTPTransfer`

`Transfer`

`__builtin__.object`

Methods defined here:

inputvalidation()

protoprocessresponse(*i*)

protosetup()

Methods inherited from `Transfer`:

baseprocessresponse(*i*)

find_local_filename(*filename*='')

getdest()

gethost()

getsource()

getstatus()

getswitchname()

gettimeout()

Send document comments to nexus3k-docfeedback@cisco.com.

geturi()

getvrf()

local_file_exist(*filename=''***)**

postvalidation()

processresponse(*i***)**

run()

setdest(*dest***)**

sethost(*host***)**

setsource(*source***)**

settimeout(*login_timeout***)**

setup(*source***, *dest*, *host*, *vrf*, *login_timeout*)**

setvrf(*vrf***)**

transferstatus()

Static methods inherited from `Transfer`:

gettransferobj(*protoco=''***, *host=''*, *source=''*, *dest=*'bootflash:', *vrf=*'management', *login_timeout=*10, *user=''*, *password=''*)**

Data descriptors inherited from `Transfer`:

`__dict__`

Send document comments to nexus3k-docfeedback@cisco.com.

dictionary for instance variables (if defined).

__weakref__

list of weak references to the object (if defined).

class History()

Method resolution order:

History

__builtin__.dict

__builtin__.object

Methods defined here:

__del__(*type*)

add_command(*command=''*)

clear_history()

get_history(*do_print*=TRUE)

Static methods defined here:

__new__(*type*)

Data descriptors defined here:

__dict__

Dictionary for instance variables (if defined).

__weakref__

List of weak references to the object (if defined).

Data and other attributes defined here:

cmd_history = deque([])

length = 256

Methods inherited from __builtin__.dict:

Send document comments to nexus3k-docfeedback@cisco.com.

__cmp__(...)

x.__cmp__(y) <==> cmp(x,y)

__contains__(...)

D.__contains__(k) -> TRUE if D has a key k, else FALSE.

__delitem__(...)

x.__delitem__(y) <==> del x[y]

__eq__(...)

x.__eq__(y) <==> x==y

__ge__(...)

x.__ge__(y) <==> x>=y

__getattribute__(...)

x.__getattribute__('name') <==> x.name

__getitem__(...)

x.__getitem__(y) <==> x[y]

__gt__(...)

x.__gt__(y) <==> x>y

__iter__(...)

x.__iter__() <==> iter(x)

__le__(...)

x.__le__(y) <==> x<=y

__len__(...)

x.__len__() <==> len(x)

__lt__(...)

Send document comments to nexus3k-docfeedback@cisco.com.

`x.__lt__(y) <==> x<y`

__ne__(...)

`x.__ne__(y) <==> x!=y`

__repr__(...)

`x.__repr__() <==> repr(x)`

__setitem__(...)

`x.__setitem__(i, y) <==> x[i]=y`

__sizeof__(...)

`D.__sizeof__() ->` size of D in memory, in bytes.

clear(...)

`D.clear() -> None.`

Remove all items from D.

copy(...)

`D.copy() ->` a shallow copy of D.

get(...)

`D.get(`*k*`[,`*d*`]) -> D[`*k*`]` if *k* in D, else *d*. *d* defaults to `None`.

has_key(...)

`D.has_key(`*k*`) -> TRUE` if `D` has a key *k*, else `FALSE`.

items(...)

`D.items() ->` list of D's `(`*key*`, `*value*`)` pairs, as 2-tuples.

iteritems(...)

`D.iteritems() ->` an iterator over the `(`*key*`, `*value*`)` items of `D`.

iterkeys(...)

Send document comments to nexus3k-docfeedback@cisco.com.

`D.iterkeys()` -> an iterator over the keys of `D`.

itervalues(...)

`D.itervalues()` -> an iterator over the values of `D`.

keys(...)

`D.keys()` -> list of `D`'s keys.

pop(...)

`D.pop(k[,d])` -> v.

Remove specified key and return the corresponding value. If key is not found, *d* is returned, if given. Otherwise, `KeyError` is raised.

popitem(...)

`D.popitem()` -> (*k*, *v*).

Remove and return some (*key*, *value*) pair as a 2-tuple; but raise `KeyError` if `D` is empty.

setdefault(...)

`D.setdefault(k[,d])` -> `D.get(k,d)`, also set `D[k]=d` if k not in `D`.

update(...)

`D.update(E, **F)` -> None.

Update `D` from dict/iterable *E* and *F*. If *E* has a `.keys()` method, this function does: for *k* in *E*: `D[k]=E[k]`. If *E* lacks `.keys()` method, this function does: for (*k*, *v*) in *E*: `D[k] = v`. In either case, this is followed by: for *k* in *F*: `D[k] = F[k]`.

values(...)

`D.values()` -> list of D's values.

viewitems(...)

`D.viewitems()` -> a set-like object providing a view on D's items.

viewkeys(...)

`D.viewkeys()` -> a set-like object providing a view on D's keys.

Send document comments to nexus3k-docfeedback@cisco.com.

viewvalues(...)

D.viewvalues() -> an object providing a view on D's values.

Data and other attributes inherited from __builtin__.dict:

__hash__ = None

fromkeys = <built-in method fromkeys of type object>

dict.fromkeys(*S*[,*v*]) -> New dict with keys from *S* and values equal to *v*. *v* defaults to None.

class IPv4ACL(*name*)

Use this class to configure the IPv4 ACL.

Method resolution order:

IPv4ACL

ACL

__builtin__.object

Methods defined here:

deny(*protocol*, *source*, *destination*, ***kwargs*)

Specify packets to reject. To stop rejecting particular packet types, set the optional *no* argument to TRUE.

Arguments

protocol: An integer ranging from <0-255> representing the protocol number, or a string representing the protocol name.

source: A string representing the source IP address or network in either CIDR notation or dotted quad. For example, '192.0.2.0', '192.0.2.0/24', '192.0.2.0/255.255.255.0'. For a network, can also specify wildcard bits. For example, '192.0.2.0/255.0.7.255'.

destination: A string representing the source IP address or network in either CIDR notation or dotted quad. For example, '192.0.2.0', '192.0.2.0/24', '192.0.2.0/255.255.255.0'. For a network, can also specify wildcard bits. For example, '192.0.2.0/255.0.7.255''.

Optional Arguments

sequence: An integer ranging from <1-4294967295> where this rule is placed.

dscp: An integer ranging from <0-63> or a string representing the type of Differentiated Services Code Point (DSCP). Use this to match packets with a particular DSCP value.

fragments: A Boolean. Set to TRUE to check non-initial fragments.

precedence: An integer ranging from <0-7> or a string representing the precedence type. Use this to match packets with a particular precedence value.

no: A Boolean. Set to TRUE to stop rejecting particular packet types.

Returns

TRUE on success.

Send document comments to nexus3k-docfeedback@cisco.com.

permit(*protocol*, *source*, *destination*, ******kwargs*)

Specify packets to forward. To stop forwarding particular packet types, set the optional *no* argument to TRUE.

Arguments

protocol: An integer ranging from <0-255> representing the protocol number, or a string representing the protocol name.

source: A string representing the source IP address or network in either CIDR notation or dotted quad. For example, '192.0.2.0', '192.0.2.0/24', '192.0.2.0/255.255.255.0'. For a network, can also specify wildcard bits. For example, '192.0.2.0/255.0.7.255''.

destination: A string representing the source IP address or network in either CIDR notation or dotted quad. For example, '192.0.2.0', '192.0.2.0/24', '192.0.2.0/255.255.255.0'. For a network, can also specify wildcard bits. For example, '192.0.2.0/255.0.7.255''.

Optional Arguments

sequence: An integer ranging from <1-4294967295> where this rule is placed.

dscp: An integer ranging from <0-63> or a string representing the type of DSCP. Use this to match packets with a particular DSCP value.

fragments: A Boolean. Set to TRUE to check non-initial fragments.

precedence: An integer ranging from <0-7> or a string representing the precedence type. Use this to match packets with a particular precedence value.

no: A Boolean. Set to TRUE to stop forwarding particular packet types.

Data and other attributes defined here:

```
__abstractmethods__ = frozenset([])
```

Methods inherited from ACL:

delete()

Delete the ACL associated with this object.

Arguments

None

Returns

TRUE on success.

delete_entry(sequence)

Delete a particular entry in this ACL by specifying the sequence number.

Arguments

sequence: An integer ranging from <1-4294967295>.

Returns

Send document comments to nexus3k-docfeedback@cisco.com.

TRUE on success.

set_per_entry_statistic(**kwargs)

Set the per-entry statistics in this ACL. To remove this configuration, set the optional *no* argument to TRUE.

Arguments

None

Optional Arguments

no: A Boolean. Set to TRUE to remove the per-entry statistics.

Returns

TRUE on success.

set_remark(remark, **kwargs)

Set a remark. To remove a remark, set the optional *no* argument to TRUE.

Arguments

remark: A string containing the remark.

Optional Arguments

no: A Boolean. Set to TRUE to remove a particular remark.

sequence: An integer sequence number where the remark is placed.

Returns

TRUE on success.

show()

Show the currently configured entries in this ACL.

Arguments

None

Returns

Outputs the ACL entries

Data descriptors inherited from ACL:

__dict__

Dictionary for instance variables (if defined).

__weakref__

List of weak references to the object (if defined).

Data and other attributes inherited from ACL:

__metaclass__ = <class 'abc.ABCMeta'>

Send document comments to nexus3k-docfeedback@cisco.com.

Metaclass for defining Abstract Base Classes (ABCs).

Use this metaclass to create an ABC. An ABC can be subclassed directly and then acts as a mix-in class. You can also register unrelated concrete classes (even built-in classes) and unrelated ABCs as ‘virtual subclasses’ -- these and their descendants are considered subclasses of the registering ABC by the built-in issubclass() function, but the registering ABC do not show up in their Method Resolution Order (MRO) nor are method implementations defined by the registering ABC callable (not even via super()).

```
ace_pat = <_sre.SRE_Pattern object>
```

class IPv6ACL(*name*)

Use this class to configure the IPv6 ACL.

Method resolution order:

IPv6ACL

ACL

__builtin__.object

Methods defined here:

deny(*protocol*, *source*, *destination*, *******kwargs*)

Specify packets to reject. To stop rejecting particular packet types, set the optional *no* argument to TRUE.

Arguments

protocol: An integer ranging from <0-255> representing the protocol number, or a string representing the protocol name.

source: A string representing the source IP network in Classless Inter-Domain Routing (CIDR) notation. For example, ‘1:1::1:1/32’.

destination: A string representing the destination IP network in CIDR notation. For example, ‘1:1::1:1/32’.

Optional Arguments

sequence: an integer ranging from <1-4294967295> where this rule is placed.

dscp: An integer ranging from <0-63> or a string representing the type of DSCP. Use this to match packets with a particular DSCP value.

fragments: A Boolean. Set to TRUE to check non-initial fragments.

no: A Boolean. Set to TRUE to stop rejecting particular packet types.

Returns

TRUE on success.

permit(*protocol*, *source*, *destination*, *******kwargs*)

Specify packets to forward. To stop forwarding particular packet types, set the optional *no* argument to TRUE.

Send document comments to nexus3k-docfeedback@cisco.com.

Arguments

protocol: An integer ranging from <0-255> representing the protocol number, or a string representing the protocol name.

source: A string representing the source IP network in CIDR notation. For example, '1:1::1:1/32'.

destination: A string representing the destination IP network in CIDR notation. For example, '1:1::1:1/32'.

Optional Arguments

sequence: An integer ranging from <1-4294967295> where this rule is placed.

dscp: An integer ranging from <0-63> or a string representing the type of DSCP. Use this to match packets with a particular DSCP value.

fragments: A Boolean. Set to `TRUE` to check non-initial fragments.

no: A Boolean. Set to `TRUE` to stop forwarding particular packet types.

Data and other attributes defined here:

```
__abstractmethods__ = frozenset([])
```

Methods inherited from `ACL`:

delete()

Delete the `ACL` associated with this object.

Arguments

None

Returns

`TRUE` on success.

delete_entry(*sequence*)

Delete a particular entry in this `ACL` by specifying the sequence number.

Arguments

sequence: An integer ranging from <1-4294967295>.

Returns

`TRUE` on success.

set_per_entry_statistic(***kwargs*)

Set the per-entry statistics in this `ACL`. To remove this configuration, set the optional *no* argument to `TRUE`.

Arguments

None

Optional Arguments

Send document comments to nexus3k-docfeedback@cisco.com.

no: A Boolean. Set to TRUE to remove the per-entry statistics.

Returns

TRUE on success.

set_remark(*remark*, ******kwargs*)

Set a remark. To remove a remark, set the optional *no* argument to TRUE.

Arguments

remark: A string containing the remark.

Optional Arguments

no: A Boolean. Set to TRUE to remove a particular remark.

sequence: An integer sequence number where the remark is placed.

Returns

TRUE on success.

show()

Show the currently configured entries in this ACL.

Arguments

None

Returns

Outputs the ACL entries.

Data descriptors inherited from ACL:

__dict__

Dictionary for instance variables (if defined).

__weakref__

List of weak references to the object (if defined).

Data and other attributes inherited from ACL:

__metaclass__ = <class 'abc.ABCMeta'>

Metaclass for defining Abstract Base Classes (ABCs).

Use this metaclass to create an ABC. An ABC can be subclassed directly and then acts as a mix-in class. You can also register unrelated concrete classes (even built-in classes) and unrelated ABCs as 'virtual subclasses' -- these and their descendants are considered subclasses of the registering ABC by the built-in issubclass() function, but the registering ABC won't show up in their Method Resolution Order (MRO) nor are method implementations defined by the registering ABC callable (not even via super()).

ace_pat = <_sre.SRE_Pattern object>

Send document comments to nexus3k-docfeedback@cisco.com.

class Interface(*intf*)

Methods defined here:

__del__(*cls*)

System shared object. Don't delete.

apply_config()

config(*refresh*=FALSE)

set_access_vlan(*vlan*=1)

set_description(*d*=None)

set_mode(*mode*='access')

set_state(*s*='up')

set_trunk_allowed_vlans(*vlans*=[])

set_trunk_native_vlan(*vlan*=1)

show(*key*=None)

show_new_config()

Class methods defined here:

`interfaces(cls, refresh=FALSE)` from `__builtin__.type`

Returns the list of interfaces on the switch

`normalize(cls, intf)` from `__builtin__.type`

`parsed_if(cls, intf)` from `__builtin__.type`

Static methods defined here:

Send document comments to nexus3k-docfeedback@cisco.com.

__new__(*cls*, *intf*)

Data descriptors defined here:

`__dict__`

Dictionary for instance variables (if defined).

`__weakref__`

List of weak references to the object (if defined).

class Key(*key*='', *start*='', end='')

Methods defined here:

get_end_key()

get_key_mode(*key*)

get_keys()

get_patterns()

get_start_key()

is_match(*line*='', *key*='')

Data descriptors defined here:

`__dict__`

Dictionary for instance variables (if defined).

`__weakref__`

List of weak references to the object (if defined).

class MacAddressTable()

Methods defined here:

add_static_mac(**args)

Send document comments to nexus3k-docfeedback@cisco.com.

Add static MAC address.

Arguments

mac: MAC address.

vlan: VLAN ID.

interface: Interface Ethernetx/y or port-channelx.

drop: 0 or 1.

auto-learn: 0 or 1:

remove_static_mac(**_args_)

Remove static MAC address.

Arguments

mac: MAC address.

vlan: VLAN ID.

Data descriptors defined here:

`__dict__`

Dictionary for instance variables (if defined).

`__weakref__`

List of weak references to the object (if defined).

class PasswordProtoTransfer(*source*, *dest*='', *host*=(None, None), *vrf*='management', *login_timeout*=10, *user*='', *password*='')

Method resolution order:

`PasswordProtoTransfer`

`Transfer`

`__builtin__.object`

Methods defined here:

baseprocessresponse(*i*)

geturi()

inputvalidation()

processresponse(*i*)

Send document comments to nexus3k-docfeedback@cisco.com.

protoprocessresponse(*i***)**

setusercredentials(*user***, *password*)**

Methods inherited from `Transfer`:

find_local_filename(*filename*= ''**)**

getdest()

gethost()

getsource()

getstatus()

getswitchname()

gettimeout()

getvrf()

local_file_exist(*filename*=''**)**

postvalidation()

protosetup()

run()

setdest(*dest***)**

Send document comments to nexus3k-docfeedback@cisco.com.

sethost(*host*)

setsource(*source*)

settimeout(*login_timeout*)

setup(*source*, *dest*, *host*, *vrf*, *login_timeout*)

setvrf(*vrf*)

transferstatus()

Static methods inherited from `Transfer`:

gettransferobj(*protocol*='', *host*='', *source*='', *dest*='bootflash:', *vrf*='management', *login_timeout*=10, *user*='', *password*='')

Data descriptors inherited from `Transfer`:

`__dict__`

Dictionary for instance variables (if defined).

`__weakref__`

List of weak references to the object (if defined).

class RouteMap(*name*, *sequence*, *type*='permit')

Use this class to create and configure route map entries.

Methods defined here:

add_description(*description*=None, ***kwargs*)

Add a description to the route map.

Arguments

description: A string up to 90 characters long.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove any existing description.

Returns

Send document comments to nexus3k-docfeedback@cisco.com.

TRUE on success.

create(**kwargs)

Create the route map entry associated with this RouteMap object.

Arguments

None

Returns

TRUE on success.

delete()

Delete the route map entry associated with this RouteMap object.

Arguments

None

Returns

TRUE on success.

match_as_number(as_number, **kwargs)

Match BGP peer AS number.

Arguments

as_number: A string in the following format: <AA4>.

Optional Arguments

no: A Boolean. Set to TRUE to remove a BGP peer from the match list.

Returns

TRUE on success.

match_as_path(lists, **kwargs)

Match AS path access list(s).

Arguments

lists: A string that is a space-separated list of AS path list names.

Optional Arguments

no: A Boolean. Set to TRUE to remove an AS path list from the match list.

Returns

TRUE on success.

match_as_path_list(lists, **kwargs)

Match AS path access list(s).

Send document comments to nexus3k-docfeedback@cisco.com.

Arguments

lists: A string that is a space-separated list of AS path list names.

Optional Arguments

no: A Boolean. Set to TRUE to remove an AS path list from the match list.

Returns

TRUE on success.

match_community(*lists*, ***kwargs*)

Match BGP community list(s).

Arguments

lists: A string that is a space-separated list of community list names.

Optional Arguments

exact_match: A Boolean. Set to TRUE to do exact matching of communities.

no: A Boolean. Set to TRUE to remove a community list from the match list.

Returns

TRUE on success.

match_extcommunity(*lists*, ***kwargs*)

Match BGP Extended community list(s).

Arguments

lists: A string that is a space-separated list of extended community list names.

Optional Arguments

exact_match: A Boolean. Set to TRUE to do exact matching of communities.

no: A Boolean. Set to TRUE to remove a community list from the match list.

Returns

TRUE on success.

match_interface(*interface*, ***kwargs*)

Match first-hop interface of route.

Arguments

interface: A string representing an interface on the switch. Can be Ethernet, loopback, mgmt, port-channel or a VLAN in either expanded or short forms. For example, ‘e1/1.2’, ‘Ethernet1/1.2’.

Optional Arguments

no: A Boolean. Set to TRUE to remove an interface from the match list.

Returns

TRUE on success.

Send document comments to nexus3k-docfeedback@cisco.com.

match_ip_access_list(*lists*, ***kwargs*)

Match IP access list(s).

Arguments

lists: A string that is a space-separated list of IP access list names.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove an IP access list from the match list.

Returns

`TRUE` on success.

match_ip_prefix_list(*lists*, ***kwargs*)

Match IP prefix list(s).

Arguments

lists: A string that is a space-separated list of IP prefix list names.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove an IP prefix list from the match list.

Returns

`TRUE` on success.

match_ipv6_access_list(*list*, ***kwargs*)

Match IPv6 access list(s).

Arguments

lists: A string that is a space-separated list of IPv6 access list names.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove an IPv6 access list from the match list.

Returns

`TRUE` on success.

match_ipv6_prefix_list(*lists*, ***kwargs*)

Match IPv6 prefix list(s).

Arguments

lists: A string that is a space-separated list of IPv6 prefix list names.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove an IPv6 prefix list from the match list.

Returns

`TRUE` on success.

Send document comments to nexus3k-docfeedback@cisco.com.

match_mac_list(*lists*, ******kwargs*)

Match mac-list(s).

Arguments

lists: A string that is a space-separated list of mac-list names.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove a mac-list from the match list.

Returns

`TRUE` on success.

match_packet_length(*min*=None, *max*=None, ******kwargs*)

Match a range of packet lengths.

Arguments

min: An integer ranging from 0 to 2147483647.

max: An integer ranging from 0 to 2147483647.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove packet length matching.

Returns

`TRUE` on success.

match_route_type(******kwargs*)

Match route-type of route.

Optional Arguments

external: A Boolean. Set to `TRUE` to match external routes (BGP, EIGRP and OSPF type 1/2).

internal: A Boolean. Set to `TRUE` to match internal routes (including OSPF intra/inter area)

level_1: A Boolean. Set to `TRUE` to match IS-IS level-1 routes.

level_2: A Boolean. Set to `TRUE` to match IS_IS level-2 routes.

local: A Boolean. Set to `TRUE` to match locally generated routes.

nssa_external: A Boolean. Set to `TRUE` to match Not-So-Stubby Area (NSSA)-external routes (OSPF type 1/2).

type_1: A Boolean. Set to `TRUE` to match OSPF external type 1 routes.

type_2: A Boolean. Set to `TRUE` to match OSPF external type 2 routes.

no: A Boolean. Set to `TRUE` to stop matching the route-type of any other parameter that was sent in.

Returns

`TRUE` on success.

match_source_protocol(*lists*, ******kwargs*)

Match source protocol.

Send document comments to nexus3k-docfeedback@cisco.com.

Arguments

lists: A string that is a space-separated list of protocol instance names.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove a source protocol from the match list.

Returns

`TRUE` on success.

match_tag(*lists*, *******kwargs*)

Match tag of route.

Arguments

lists: A string that is a space-separated list of tags where each tag is <0-4294967295>.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove a tag from the match list.

Returns

`TRUE` on success.

match_vlan(*lists*, *******kwargs*)

Match VLAN ID(s).

Arguments

lists: A string that is a comma-separated list of VLANs and/or VLAN ranges where each VLAN can range from <1-3967, 4048-4093>. For example, 1-5, 10 or 2-5,7-19.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove a list of VLANs and/or VLAN ranges from the match list.

Returns

`TRUE` on success.

set_as_path_prepend(*list*, *******kwargs*)

Prepend string for a BGP `AS_PATH` attribute.

Arguments

list: A string that is a space-separated list of ASes where each AS is in the following format: <1-4294967295>|<1-65535>[.<0-65535>].

Optional Arguments

no: A Boolean. Set to `TRUE` to remove ASes from the prepend string.

Returns

`TRUE` on success.

Send document comments to nexus3k-docfeedback@cisco.com.

set_as_path_tag(**kwargs)

Set the tag as an AS_PATH attribute.

Arguments

None

Optional Arguments

no: A Boolean. Set to TRUE to remove this configuration.

Returns

TRUE on success.

set_comm_list_delete(list, **kwargs)

Set BGP community list (for deletion).

Arguments

list: A string up to 63 characters representing a community list name.

Optional Arguments

no: A Boolean. Set to TRUE to remove this configuration.

Returns

TRUE on success.

set_extcomm_list_delete(list, **kwargs)

Set BGP External community list (for deletion).

Arguments

list: A string up to 63 characters representing a external community list name.

Optional Arguments

no: A Boolean. Set to TRUE to remove this configuration.

Returns

TRUE on success.

set_forwarding_address(**kwargs)

Set the forwarding address.

Arguments

None

Optional Arguments

no: A Boolean. Set to TRUE to remove this configuration.

Returns

TRUE on success.

Send document comments to nexus3k-docfeedback@cisco.com.

set_vrf(*vrf*, *******kwargs*)

Set the VRF for next-hop resolution.

Arguments

vrf: A string representing an existing VRF on the switch.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove a VRF from next-hop resolution.

Returns

`TRUE` on success.

set_weight(*weight*, *******kwargs*)

Set BGP weight for routing table.

Arguments

weight: An integer ranging from <-65535>.

Optional Arguments

no: A Boolean. Set to `TRUE` to remove this configuration.

Returns

`TRUE` on success.

Data descriptors defined here:

`__dict__`

Dictionary for instance variables (if defined).

`__weakref__`

List of weak references to the object (if defined).

class Routes()

Methods defined here:

add_route(*srcIp=''*, *prefix=''*, *mask=''*, *intf=''*, *nexthop=''*, *nhMask=''*, *nhPrefix=''*, *tag=''*, *routePref=''*, *vrf='*default*'*)

delete_route(*srcIp=''*, *prefix=''*, *mask=''*, *intf=''*, *nexthop=''*, *nhMask=''*, *nhPrefix=''*, *tag=''*, *routePref=''*, *vrf='*default*'*)

show_arp_table()

Send document comments to nexus3k-docfeedback@cisco.com.

show_hw_routes()

show_vsh_routes()

verify_arp_table()

verify_route(*route*='')

verify_routes()

Data descriptors defined here:

`__dict__`

Dictionary for instance variables (if defined).

`__weakref__`

List of weak references to the object (if defined).

Data and other attributes defined here:

```
arpKey = None
egressKey = None
fwmIntVlanKey = None
hopEntries = {'Attached': '100003', 'Drop': '100000', 'Receive': '1000...
l3defipKey = None
l3intKey = None
l3tableKey = None
validHops = {'100000': ('Drop', 'Null0'), '100002': ('Receive', 'sup-e...
vshHeadKey = None
```

class SCPTransfer(*source*, *dest*='', *host*=(None, None), *vrf*='management', *login_timeout*=10, *user*='', *password*='')

Method resolution order:

`SCPTransfer`

`PasswordProtoTransfer`

`Transfer`

`__builtin__.object`

Methods defined here:

Send document comments to nexus3k-docfeedback@cisco.com.

protoprocessresponse(*i*)

protosetup()

Methods inherited from `PasswordProtoTransfer`:

baseprocessresponse(*i*)

geturi()

inputvalidation()

processresponse(*i*)

setusercredentials(*user*, *password*)

Methods inherited from `Transfer`:

find_local_filename(*filename*='')

getdest()

gethost()

getsource()

getstatus()

getswitchname()

gettimeout()

Send document comments to nexus3k-docfeedback@cisco.com.

getvrf()

local_file_exist(*filename=''***)**

postvalidation()

run()

setdest(*dest***)**

sethost(*host***)**

setsource(*source***)**

settimeout(*login_timeout***)**

setup(*source***, *dest*, *host*, *vrf*, *login_timeout*)**

setvrf(*vrf***)**

transferstatus()

Static methods inherited from `Transfer`:

gettransferobj(*protocol=''***, *host=''*, *source=''*, *dest=*'bootflash:', *vrf=*'management', *login_timeout=*10, *user=''*, *password=''*)**

Data descriptors inherited from `Transfer`:

`__dict__`

Dictionary for instance variables (if defined).

`__weakref__`

List of weak references to the object (if defined).

Send document comments to nexus3k-docfeedback@cisco.com.

class SFTPTransfer(*source*, dest='', *host*=(None, None), *vrf*='management', *login_timeout*=10, *user*='', *password*='')

Method resolution order:

`SFTPTransfer`

`PasswordProtoTransfer`

`Transfer`

`__builtin__.object`

Methods defined here:

protoprocessresponse(*i*)

protosetup()

Methods inherited from `PasswordProtoTransfer`:

baseprocessresponse(*i*)

geturi()

inputvalidation()

processresponse(*i*)

setusercredentials(*user*, *password*)

Methods inherited from `Transfer`:

find_local_filename(*filename*='')

getdest()

gethost()

Send document comments to nexus3k-docfeedback@cisco.com.

getsource()

getstatus()

getswitchname()

gettimeout()

getvrf()

local_file_exist(*filename=''***)**

postvalidation()

run()

setdest(*dest***)**

sethost(*host***)**

setsource(*source***)**

settimeout(*login_timeout***)**

setup(*source***, *dest*, *host*, *vrf*, *login_timeout*)**

setvrf(*vrf***)**

transferstatus()

Static methods inherited from `Transfer`:

Send document comments to nexus3k-docfeedback@cisco.com.

gettransferobj(*protocol*='', *host*='', *source*='', *dest*='bootflash:', *vrf*='management', *login_timeout*=10, *user*='', *password*='')

Data descriptors inherited from `Transfer`:

`__dict__`

Dictionary for instance variables (if defined).

`__weakref__`

List of weak references to the object (if defined).

class SSH()

Method resolution order:

`SSH`

`Feature`

`__builtin__.object`

Methods defined here:

gen_key(**_args_)

Generate Secure Shell (SSH) keys.

Arguments

key_type:

dsa: Generate Digital Signature Algorithm (DSA) keys.

rsa: Generate Ron Rivest, Adi Shamir and Leonard Adleman (RSA) keys.

Optional Arguments

bits: <768-2048>. Number of bits (only for RSA keys).

force: Force the generation of keys even if previous ones are present.

no: If set, remove SSH key.

get_keys(**_args_)

Generate SSH keys.

Optional Arguments

key_type:

dsa: DSA keys.

rsa: RSA keys.

Methods inherited from `Feature`:

Send document comments to nexus3k-docfeedback@cisco.com.

disable()

Disable feature.

enable(**kwargs)

Start feature.

Arguments

no=TRUE: Stops the TACACS.

Returns

TRUE on success.

is_enabled()

Returns

TRUE if the feature is enabled.

name()

Feature name—as used in the **config terminal** command.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of Feature.

Static methods inherited from Feature:

__new__(*typ*, *args, **kwargs)

Create single instance of object per derived class

class ShowInterface(*ifname*)

Method resolution order:

ShowInterface

CLI

__builtin__.object

Methods defined here:

Send document comments to nexus3k-docfeedback@cisco.com.

parse_specific()

Methods inherited from CLI:

get_command()

get_output()

get_raw_output()

get_status()

get_timestamp()

get_xml_dom_from_cli_output(*text*)

key_map(*key*)

key_value_colon_parser(*line*)

key_value_xml_parser(*element*)

rerun()

Data descriptors inherited from CLI:

__dict__

Dictionary for instance variables (if defined).

__weakref__

List of weak references to the object (if defined).

class System(__builtin__.object)

Class to provide miscellaneous system configuration.

Methods defined here:

Send document comments to nexus3k-docfeedback@cisco.com.

get_hostname()

Get the hostname of the switch.

get_mgmt0_ip()

Get IP address of mgmt0.

set_hostname(*name*=None)

Set the hostname of the switch.

set_mgmt0_ip(*address*=None, *mask*=None)

Set the IPv4 or IPv6 address and netmask on mgmt0.

Address of the form:

<ip> that is, 10.1.1.1

<ip>/<no. bits in mask> that is, 10.1.1.1/24

<ip>/<mask> that is, 10.1.1.1/255.255.255.0

set_password(*username*='admin', *password*=None, *expire*=None, *role*=None, *delete*=None)

Set user password and associated options.

Arguments

username

password: "[0 |5]WORD".

0—Indicates that the password that follows should be in cleartext.

5—Indicates that the password that follows should be encrypted.

WORD—Password for the user (cleartext) (max Size 64).

expire: WORD Expiry in YYYY-MM-DD format (max Size 10).

role: Where `role` is one of:

network-admin—System configured role.

network-operator—System configured role.

priv-0—Privilege role.

priv-1—Privilege role.

priv-10—Privilege role.

priv-11—Privilege role.

priv-12—Privilege role.

priv-13—Privilege role.

priv-14—Privilege role.

Send document comments to nexus3k-docfeedback@cisco.com.

priv-15—Privilege role.

priv-2—Privilege role.

priv-3—Privilege role.

priv-4—Privilege role.

priv-5—Privilege role.

priv-6—Privilege role.

priv-7—Privilege role.

priv-8—Privilege role.

priv-9—Privilege role.

vdc-admin— System configured role.

vdc-operator—System configured role.

Data descriptors defined here:

`__dict__`

Dictionary for instance variables (if defined).

`__weakref__`

List of weak references to the object (if defined).

class TFTPTransfer(*source*, *dest*='', *host*=(None, None), *vrf*='management', *login_timeout*=10)

Method resolution order:

`TFTPTransfer`

`Transfer`

`__builtin__.object`

Methods defined here:

protoprocessresponse(*i*)

protosetup()

Methods inherited from `Transfer`:

baseprocessresponse(*i*)

find_local_filename(*filename*='')

Send document comments to nexus3k-docfeedback@cisco.com.

getdest()

gethost()

getsource()

getstatus()

getswitchname()

gettimeout()

geturi()

getvrf()

inputvalidation()

local_file_exist(*filename=''***)**

postvalidation()

processresponse(*i***)**

run()

setdest(*dest***)**

sethost(*host***)**

setsource(*source***)**

Send document comments to nexus3k-docfeedback@cisco.com.

settimeout(*login_timeout*)

setup(*source*, *dest*, *host*, *vrf*, *login_timeout*)

setvrf(*vrf*)

transferstatus()

Static methods inherited from `Transfer`:

gettransferobj(*protocol*='', *host*='', *source*='', *dest*='bootflash:', *vrf*='management', *login_timeout*=10, **user**='', *password*='')

Data descriptors inherited from `Transfer`:

`__dict__`

Dictionary for instance variables (if defined).

`__weakref__`

List of weak references to the object (if defined).

class Tacacs()

Method resolution order:

`Tacacs`

`Feature`

`__builtin__.object`

Methods defined here:

add_group(*name*, *server*, ***args*)

Specify one or more remote authentication, authorization and accounting (AAA) servers to authenticate users using server groups. All members of a group must belong to the Terminal Access Controller Access-Control System Plus (TACACS+) protocol. The servers are tried in the same order in which you configure them.

You can configure these server groups at any time but they only take effect when you apply them to an AAA service.

Optional Arguments

deadtime=<1-1440>: Configures the monitoring dead time. The default is 0 minutes. The range is from 1 through 1440.

Send document comments to nexus3k-docfeedback@cisco.com.

If the dead-time interval for a TACACS+ server group is greater than zero (0), that value takes precedence over the global dead-time value.

source_interface=‘m0’ Configures a source interface to access the TACACS+ servers in the server group. You can use Ethernet interfaces, loopback interfaces, or the management interface (mgmt 0). The default is the global source interface.

vrf=<vrf-name>: Specifies the VRF to use to contact the servers in the server group.

no=TRUE: Deletes the group.

add_server(*server*, *******args*)

Adds TACACS+ servers. To access a remote TACACS+ server, you must configure the IP address or the hostname for the TACACS+ server on the Cisco NX-OS device. You can configure up to 64 TACACS+ servers.

Arguments

server: TACACS+ server’s Domain Name System (DNS) name or its IP address.

port=<1-65535>: TACACS+ server port.

key=CiscoSecret: Global TACACS+ server shared secret.

timeout=#: TACACS+ server timeout period in seconds.

no=TRUE: Deletes the server.

Returns

TRUE on success.

Example

```
t = Tacacs ()
t.add_servers (‘20.1.2.5’, key=CiscoSecret (‘Z75ftWs’, 7), timeout=60)
t.add_servers (‘FF01::101’)
```

commit()

(Optional) Applies the TACACS+ configuration changes in the temporary database to the running configuration and distributes TACACS+ configuration to other NX-OS devices in the network that you have enabled CFS configuration distribution for the TACACS+ feature.

deadtime(*mins*, *******args*)

Monitor the availability of TACACS+ servers. These parameters include the username and password to use for the server and an idle timer. The idle timer specifies the interval in which a TACACS+ server receives no requests before the Cisco NX-OS device sends out a test packet. You can configure this option to test servers periodically, or you can run a one-time-only test.

Arguments

mins: Dead time.

no=TRUE—Remove it.

Send document comments to nexus3k-docfeedback@cisco.com.

directed_request(**args)

Configure the switch to allow the user to specify which TACACS+ server to send the authentication request by enabling the directed-request option. By default, a Cisco NX-OS device forwards an authentication request based on the default AAA method. If you enable this option, the user can log in as `username@vrfname:hostname`, where `vrfname` is the VRF to use and `hostname` is the name of a configured TACACS+ server.

Arguments

no=`TRUE`: Stops it.

distribute(**args)

Start distribution of the TACACS+ configuration changes in the `CFS` region.

Arguments

no=`TRUE`: Stops distribution.

Returns

`TRUE` on success.

server(s)

Return information for the servers.

servers()

Return list of configured servers.

set_key(key, **args)

Specify a TACACS+ key for all TACACS+ servers. You can specify the key-value in the `cisco.CiscoSecret object`, if the key is in cleartext (0) format or is encrypted (7). The Cisco NX-OS software encrypts a cleartext key before saving it to the running configuration. The default format is cleartext. The maximum length is 63 characters. By default, no secret key is configured.

Arguments

key: Shared key.

no=`TRUE`: Deletes the key.

Example

```
t = Tacacs ()
t.set_key (CiscoSecret ('Secret'))
```

src_interface(ifname, **args)

Send document comments to nexus3k-docfeedback@cisco.com.

Configure a global source interface for TACACS+ server groups to use when accessing TACACS+ servers. To configure a different source interface for a specific TACACS+ server group, use the `add_group()` function. By default, the Cisco NX-OS software uses any available interface.

Arguments

ifname—Source interface.

no=`TRUE`: Deletes the key.

test_server(*host*, *******args*)

Monitor the availability of TACACS+ servers. These parameters include the username and password to use for the server and an idle timer. The idle timer specifies the interval in which a TACACS+ server receives no requests before the Cisco NX-OS device sends out a test packet. You can configure this option to test servers periodically, or you can run a one-time-only test.

timeout(*secs*, *******args*)

Set a global timeout interval that the Cisco NX-OS device waits for responses from all TACACS+ servers before declaring a timeout failure. The timeout interval determines how long the Cisco NX-OS device waits for responses from TACACS+ servers before declaring a timeout failure.

Arguments

secs: Timeout interval.

no=`TRUE`: Remove it.

Methods inherited from `Feature`:

disable()

Disable feature.

enable(*******kwargs*)

Start feature.

Arguments

no=`TRUE`: Stops the TACACS.

Returns

`TRUE` on success.

is_enabled()

Return `TRUE` if the feature is enabled.

name()

Feature name—as used in the **config terminal** command.

Send document comments to nexus3k-docfeedback@cisco.com.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of `Feature`.

Static methods inherited from `Feature`:

__new__(*typ*, **args*, ***kwargs*)

Create single instance of object per derived class.

class Transfer(*source*, *dest*='', *host*=(None, None), *vrf*='management', *login_timeout*=10)

Methods defined here:

baseprocessresponse(*i*)

find_local_filename(*filename*='')

getdest()

gethost()

getsource()

getstatus()

getswitchname()

gettimeout()

geturi()

Send document comments to nexus3k-docfeedback@cisco.com.

getvrf()

inputvalidation()

local_file_exist(*filename=''***)**

postvalidation()

processresponse(*i***)**

protoprocessresponse(*i***)**

protosetup()

run()

setdest(*dest***)**

sethost(*host***)**

setsource(*source***)**

settimeout(*login_timeout***)**

setup(*source***, *dest*, *host*, *vrf*, *login_timeout*)**

setvrf(*vrf***)**

transferstatus()

Static methods defined here:

Send document comments to nexus3k-docfeedback@cisco.com.

gettransferobj(*protocol*='', *host*='', *source*='', *dest*='bootflash:', *vrf*='management', *login_timeout*=10, *user*='', *password*='')

Data descriptors defined here:

`__dict__`

Dictionary for instance variables (if defined).

`__weakref__`

List of weak references to the object (if defined).

class VRF(*vrf*)

Use this object to create/delete a VRF on the switch, add/remove interfaces to a VRF or simply just to check if a VRF exists.

Methods defined here:

add_interface(*if_name*, ***args*)

Set the specified interface's VRF membership to this VRF.

Arguments

if_name: A string specifying the interface name.

Optional Arguments

no: A Boolean. Set to TRUE to remove this VRF from the specified interface's VRF membership.

Returns

`TRUE` on success.

Example

```
v = VRF('floor1')
v.create()
v.add_interface('Ethernet 1/1')
```

create(***args*)

Create the VRF.

Arguments

None

Optional Arguments

no: A Boolean. Set to `TRUE` to delete the VRF.

Returns

`TRUE` on success.

Example

Send document comments to nexus3k-docfeedback@cisco.com.

```
v = VRF('floor')
v.create()
```

delete()

Delete the VRF.

Arguments

None

Returns

TRUE on success.

Example

```
v = VRF''floor')
v.delete()
```

exists()

Check if this VRF exists.

Arguments

None

Returns

TRUE if this VRF exists.

FALSE if it doesn't exist.

Example

```
v = VRF('blahblah')
if not v.exists():
v.create()
```

get_name()

Get the name of this VRF.

Arguments

None

Returns

The name of this VRF as a string.

Example

```
v = VRF(2)
v.get_name()
```

remove_interface(*if_name*)

Removes this VRF from the specified interface's VRF membership.

Arguments

if_name: A string specifying the interface name.

Send document comments to nexus3k-docfeedback@cisco.com.

Returns

TRUE on success.

Example

```
v = VRF''floor1')
v.create()
v.add_interface('Ethernet 1/1')
v.delete_interface('Ethernet 1/1')
----------------------------------------------------------------------
```

Static methods defined here:

get_vrf_id_by_name(*target_vrf_name*)

Return the VRF ID associated with the specified VRF name.

get_vrf_name_by_id(*target_vrf_id*)

Return the VRF name associated with the specified VRF ID.

Data descriptors defined here:

`__dict__`

Dictionary for instance variables (if defined).

`__weakref__`

List of weak references to the object (if defined).

class Vlan()

Methods defined here:

create_vlan(*id*, ****args*)

Create the VLAN.

Arguments

id: VLAN ID.

Optional Arguments

name: VLAN description.

state: VLAN state.

mode: VLAN mode.

type: VLAN type.

Send document comments to nexus3k-docfeedback@cisco.com.

delete_vlan(*id*)

Delete the VLAN.

Arguments

id: VLAN ID.

show_vlan()

Return VLANs configured on the switch.

Data descriptors defined here:

`__dict__`

Dictionary for instance variables (if defined).

`__weakref__`

List of weak references to the object (if defined).

class bfd(*Feature*)

`Bfd`

Method resolution order:

`bfd`

`Feature`

`__builtin__.object`

Methods inherited from `Feature`:

disable()

Disable feature.

enable(******kwargs*)

Start feature.

Arguments

no=`TRUE`: Stops the TACACS.

Returns

`TRUE` on success.

is_enabled()

Send document comments to nexus3k-docfeedback@cisco.com.

Return `TRUE` if the feature is enabled.

name()

Feature name—as used in the **config terminal** command.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of `Feature`.

Static methods inherited from `Feature`:

__new__(*typ*, **args*, ***kwargs*)

Create single instance of object per derived class.

class dhcp(*Feature*)

Enable/Disable Dynamic Host Configuration Protocol (DHCP) snooping.

Method resolution order:

`dhcp`

`Feature`

`__builtin__.object`

Methods inherited from `Feature`:

disable()

Disable feature.

enable(***kwargs*)

Start feature.

Arguments

no=`TRUE`: Stops the TACACS.

Returns

`TRUE` on success.

is_enabled()

Send document comments to nexus3k-docfeedback@cisco.com.

Return `TRUE` if the feature is enabled.

name()

Feature name— as used in the **config terminal** command.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of `Feature`.

Static methods inherited from `Feature`:

__new__(*typ*, **args*, ***kwargs*)

Create single instance of object per derived class.

class eigrp(*Feature*)

Enable/Disable Enhanced Interior Gateway Routing Protocol (EIGRP).

Method resolution order:

`eigrp`

`Feature`

`__builtin__.object`

Methods inherited from `Feature`:

disable()

Disable feature.

enable(**kwargs)

Start feature.

Arguments

no=`TRUE`: Stops the TACACS.

Returns

`TRUE` on success.

is_enabled()

Send document comments to nexus3k-docfeedback@cisco.com.

Return TRUE if the feature is enabled.

name()

Feature name—as used in the **config terminal** command.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of Feature.

Static methods inherited from Feature:

__new__(*typ*, **args*, ***kwargs*)

Create single instance of object per derived class.

class hsrp(*Feature*)

Enable/Disable Hot Standby Router Protocol (HSRP).

Method resolution order:

hsrp

Feature

__builtin__.object

Methods inherited from Feature:

disable()

Disable feature.

enable(***kwargs*)

Start feature.

Arguments

no=TRUE: Stops the TACACS.

Returns

TRUE on success.

is_enabled()

Send document comments to nexus3k-docfeedback@cisco.com.

Return `TRUE` if the feature is enabled.

name()

Feature name—as used in the **config terminal** command.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of `Feature`.

Static methods inherited from `Feature`:

__new__(*typ*, **args*, ***kwargs*)

Create single instance of object per derived class.

class interface-vlan(*Feature*)

Enable/Disable interface VLAN.

Method resolution order:

`interface-vlan`

`Feature`

`__builtin__.object`

Methods inherited from `Feature`:

disable()

Disable feature.

enable(**kwargs)

Start feature.

Arguments

no=`TRUE`: Stops the TACACS.

Returns

`TRUE` on success.

is_enabled()

Send document comments to nexus3k-docfeedback@cisco.com.

Return TRUE if the feature is enabled.

name()

Feature name—as used in the **config terminal** command.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of Feature.

Static methods inherited from Feature:

__new__(*typ*, **args*, ***kwargs*)

Create single instance of object per derived class.

class lacp(*Feature*)

Enable/Disable Link Aggregation Control Protocol (LACP).

Method resolution order:

lacp

Feature

__builtin__.object

Methods inherited from Feature:

disable()

Disable feature.

enable(***kwargs*)

Start feature.

Arguments

no=TRUE: Stops the TACACS.

Returns

TRUE on success.

is_enabled()

Send document comments to nexus3k-docfeedback@cisco.com.

Return `TRUE` if the feature is enabled.

name()

Feature name—as used in the **config terminal** command.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of `Feature`.

Static methods inherited from `Feature`:

__new__(*typ*, **args*, ***kwargs*)

Create single instance of object per derived class.

class msdp(*Feature*)

Enable/Disable Multicast Source Discovery Protocol (MSDP).

Method resolution order:

`msdp`

`Feature`

`__builtin__.object`

Methods inherited from `Feature`:

disable()

Disable feature.

enable(******kwargs*)

Start feature.

Arguments

no=`TRUE`: Stops the TACACS.

Returns

`TRUE` on success.

is_enabled()

Send document comments to nexus3k-docfeedback@cisco.com.

Return `TRUE` if the feature is enabled.

name()

Feature name—as used in the **config terminal** command.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of `Feature`.

Static methods inherited from `Feature`:

__new__(*typ*, **args*, ***kwargs*)

Create single instance of object per derived class.

class ospf(*Feature*)

Enable/Disable Open Shortest Path First Protocol (OSPF).

Method resolution order:

`ospf`

`Feature`

`__builtin__.object`

Methods inherited from `Feature`:

disable()

Disable feature.

enable(******kwargs*)

Start feature.

Arguments

no=`TRUE`: Stops the TACACS.

Returns

`TRUE` on success.

is_enabled()

Send document comments to nexus3k-docfeedback@cisco.com.

Return `TRUE` if the feature is enabled.

name()

Feature name—as used in the **config terminal** command.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of `Feature`.

Static methods inherited from `Feature`:

__new__(*typ*, ***args*, ****kwargs*)

Create single instance of object per derived class.

class ospfv3(*Feature*)

Enable/Disable Open Shortest Path First Version 3 Protocol (OSPFv3).

Method resolution order:

`ospfv3`

`Feature`

`__builtin__.object`

Methods inherited from `Feature`:

disable()

Disable feature.

enable(****kwargs*)

Start feature.

Arguments

no=`TRUE`: Stops the TACACS.

Returns

`TRUE` on success.

is_enabled()

Send document comments to nexus3k-docfeedback@cisco.com.

Returns `TRUE` if the feature is enabled.

name()

Feature name—as used in the **config terminal** command.

show_name()

Feature name - as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of `Feature`.

Static methods inherited from `Feature`:

__new__(*typ*, **args*, ***kwargs*)

Create single instance of object per derived class

class pim(*Feature*)

Enable/Disable Protocol Independent Multicast (PIM).

Method resolution order:

`pim`

`Feature`

`__builtin__.object`

Methods inherited from `Feature`:

disable()

Disable feature.

enable(***kwargs*)

Start feature.

Arguments

no=`TRUE`: Stops the TACACS.

Returns

`TRUE` on success.

is_enabled()

Send document comments to nexus3k-docfeedback@cisco.com.

Returns `TRUE` if the feature is enabled.

name()

Feature name—as used in the **config terminal** command.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of `Feature`.

Static methods inherited from `Feature`:

__new__(*typ*, **args*, ***kwargs*)

Create single instance of object per derived class.

class private-vlan(*Feature*)

Enable/Disable private VLAN.

Method resolution order:

`private-vlan`

`Feature`

`__builtin__.object`

Methods inherited from `Feature`:

disable()

Disable feature.

enable(**kwargs)

Start feature.

Arguments

no=`TRUE`: Stops the TACACS.

Returns

`TRUE` on success.

is_enabled()

Send document comments to nexus3k-docfeedback@cisco.com.

Returns `TRUE` if the feature is enabled.

name()

Feature name—as used in the **config terminal** command.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of `Feature`.

Static methods inherited from `Feature`:

__new__(*typ*, **args*, ***kwargs*)

Create single instance of object per derived class.

class ptp(*Feature*)

Enable/Disable PTP.

Method resolution order:

`ptp`

`Feature`

`__builtin__.object`

Methods inherited from `Feature`:

disable()

Disable feature.

enable(***kwargs*)

Start feature.

Arguments

no=`TRUE`: Stops the TACACS.

Returns

`TRUE` on success.

is_enabled()

Send document comments to nexus3k-docfeedback@cisco.com.

Return `TRUE` if the feature is enabled.

name()

Feature name—as used in the **config terminal** command.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of `Feature`.

Static methods inherited from `Feature`:

__new__(*typ*, ***args*, ****kwargs*)

Create single instance of object per derived class.

class rip(*Feature*)

Enable/Disable Routing Information Protocol (RIP).

Method resolution order:

`rip`

`Feature`

`__builtin__.object`

Methods inherited from `Feature`:

disable()

Disable feature.

enable(****kwargs*)

Start feature.

Arguments

no=`TRUE`: Stops the TACACS.

Returns

`TRUE` on success.

is_enabled()

Send document comments to nexus3k-docfeedback@cisco.com.

Return `TRUE` if the feature is enabled.

name()

Feature name—as used in the **config terminal** command.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of `Feature`.

Static methods inherited from `Feature`:

__new__(*typ*, **args*, ***kwargs*)

Create single instance of object per derived class.

class scheduler(*Feature*)

Enable/Disable scheduler

Method resolution order:

`scheduler`

`Feature`

`__builtin__.object`

Methods inherited from `Feature`:

disable()

Disable feature.

enable(***kwargs*)

Start feature.

Arguments

no=`TRUE`: Stops the TACACS.

Returns

`TRUE` on success.

is_enabled()

Send document comments to nexus3k-docfeedback@cisco.com.

Returns `TRUE` if the feature is enabled.

name()

Feature name—as used in the **config terminal** command.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of `Feature`.

Static methods inherited from `Feature`:

__new__(*typ*, **args*, ***kwargs*)

Create single instance of object per derived class.

class telnet(*Feature*)

Enable/Disable Telnet.

Method resolution order:

`telnet`

`Feature`

`__builtin__.object`

Methods inherited from `Feature`:

disable()

Disable feature.

enable(***kwargs*)

Start feature.

Arguments

no=`TRUE`: Stops the TACACS.

Returns

`TRUE` on success.

is_enabled()

Send document comments to nexus3k-docfeedback@cisco.com.

Return TRUE if the feature is enabled.

name()

Feature name—as used in the **config terminal** command.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of Feature.

Static methods inherited from Feature:

__new__(*typ*, **args*, ***kwargs*)

Create single instance of object per derived class

class udld(*Feature*)

Enable/Disable Unidirectional Link Detection (UDLD).

Method resolution order:

udld

Feature

__builtin__.object

Methods inherited from Feature:

disable()

Disable feature.

enable(***kwargs*)

Start feature.

Arguments

no=TRUE: Stops the TACACS.

Returns

TRUE on success.

is_enabled()

Send document comments to nexus3k-docfeedback@cisco.com.

Return TRUE if the feature is enabled.

name()

Feature name—as used in the **config terminal** command.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of Feature.

Static methods inherited from Feature:

__new__(*typ*, **args*, ***kwargs*)

Create single instance of object per derived class.

class vpc(*Feature*)

Enable/Disable (Virtual Port Channel) (VPC).

Method resolution order:

vpc

Feature

__builtin__.object

Methods inherited from Feature:

disable()

Disable feature.

enable(***kwargs*)

Start feature.

Arguments

no=TRUE: Stops the TACACS.

Returns

TRUE on success.

is_enabled()

Send document comments to nexus3k-docfeedback@cisco.com.

Return `TRUE` if the feature is enabled.

name()

Feature name—as used in the **config terminal** command.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of `Feature`.

Static methods inherited from `Feature`:

__new__(*typ*, **args*, ***kwargs*)

Create single instance of object per derived class.

class vrrp(*Feature*)

Enable/Disable Virtual Router Redundancy Protocol (VRRP).

Method resolution order:

`vrrp`

`Feature`

`__builtin__.object`

Methods inherited from `Feature`:

disable()

Disable feature.

enable(******kwargs*)

Start feature.

Arguments

no=`TRUE`: Stops the TACACS.

Returns

`TRUE` on success.

is_enabled()

Send document comments to nexus3k-docfeedback@cisco.com.

Return `TRUE` if the feature is enabled.

name()

Feature name—as used in the **config terminal** command.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of `Feature`.

Static methods inherited from `Feature`:

__new__(*typ*, **args*, ***kwargs*)

Create single instance of object per derived class.

class vtp(*Feature*)

Enable/Disable VLAN Trunking Protocol (VTP).

Method resolution order:

`vtp`

`Feature`

`__builtin__.object`

Methods inherited from `Feature`:

disable()

Disable feature.

enable(**kwargs)

Start feature.

Arguments

no=`TRUE`: Stops the TACACS.

Returns

`TRUE` on success.

is_enabled()

Send document comments to nexus3k-docfeedback@cisco.com.

Return TRUE if the feature is enabled.

name()

Feature name—as used in the **config terminal** command.

show_name()

Feature name—as seen in the **show feature** command.

state(*instance*=0)

Return state (or states if multiple instances) of Feature.

Static methods inherited from Feature:

__new__(*typ*, *args, **kwargs)

Create single instance of object per derived class.

FUNCTIONS

cfg_if(*port*, *desc*=None, *vlan*=None, *state*=None, *mode*=None, *allowedVlan*=None, *channelGroup*=None)

This function is deprecated. Use the Interface class.

cli(*str*='', *do_print*=FALSE)

get_valid_port(*port*)

Validate and return correct port here.

show_queues(*type*=None)

show_run(*intf*=None)

transfer(*protocol*='', *host*='', *source*='', *dest*='bootflash', *vrf*='management', *login_timeout*=10, *user*='', *password*='')