Python Client for eAPI Documentation

Release 0.8.2

Arista EOS+ CS

Feb 09, 2018

Contents

The Python Client for eAPI (pyeapi) is a native Python library wrapper around Arista EOS eAPI. It provides a set of Python language bindings for configuring Arista EOS nodes.

The Python library can be used to communicate with EOS either locally (on-box) or remotely (off-box). It uses a standard INI-style configuration file to specify one or more nodes and connection profiles.

The pyeapi library also provides an API layer for building native Python objects to interact with the destination nodes. The API layer is a convenient implementation for working with the EOS configuration and is extensible for developing custom solutions.

This library is freely provided to the open source community for building robust applications using Arista EOS. Support is provided as best effort through Github issues.

CHAPTER 1

Installation

The installation of pyeapi is straightforward and simple. As mentioned in the *Introduction*, pyeapi can be run on-box or off-box. The instructions below will provide some tips to help you for either platform.

Contents

1.1 Pip with Network Access

If your platform has internet access you can use the Python Package manager to install pyeapi

```
admin:~ admin$ sudo pip install pyeapi
```

Note: You will likely notice Pip install netaddr, a dependency of pyeapi.

1.1.1 Pip - Upgrade Pyeapi

```
admin:~ admin$ sudo pip install --upgrade pyeapi
```

1.2 Pip without Network Access

If you want to install pyeapi on a switch with no internet access:

Step 1: Download Pypi Package

- Download the latest version of **pyeapi** on your local machine.
- You will also need a dependency package netaddr.

Step 2: SCP both files to the Arista switch and install

```
admin:~ admin$ scp path/to/pyeapi-<VERSION>.tar.gz ansible@veos01:/mnt/flash/
admin:~ admin$ scp path/to/netaddr-<VERSION>.tar.gz ansible@veos01:/mnt/flash/
```

Then SSH into your node and install it. Be sure to replace `<VERSION>` with the actual filename:

```
[admin@veos ~]$ sudo pip install /mnt/flash/netaddr-<VERSION>.tar.gz
[admin@veos ~]$ sudo pip install /mnt/flash/pyeapi-<VERSION>.tar.gz
```

These packages must be re-installed on reboot. Therefore, add the install commands to `/mnt/flash/rc.eos` to trigger the install on reboot:

```
[admin@veos ~]$ vi /mnt/flash/rc.eos
```

Add the lines (the #! may already exist in your rc.eos)

```
#!/bin/bash
sudo pip install /mnt/flash/netaddr-<VERSION>.tar.gz
sudo pip install /mnt/flash/pyeapi-<VERSION>.tar.gz
```

1.3 Development - Run pyeapi from Source

Tip: We recommend running pyeapi in a virtual environment. For more information, read this.

These instructions will help you install and run pyeapi from source. This is useful if you plan on contributing or if you'd always like to see the latest code in the develop branch.

Important: These steps require Pip and Git

Step 1: Clone the pyeapi Github repo

```
# Go to a directory where you'd like to keep the source
admin:~ admin$ cd ~/projects
admin:~ admin$ git clone https://github.com/arista-eosplus/pyeapi.git
admin:~ admin$ cd pyeapi
```

Step 2: Check out the desired version or branch

```
# Go to a directory where you'd like to keep the source
admin:~ admin$ cd ~/projects/pyeapi

# To see a list of available versions or branches
```

```
admin:~ admin$ git tag
admin:~ admin$ git branch

# Checkout the desired version of code
admin:~ admin$ git checkout v0.3.3
```

Step 3: Install pyeapi using Pip with -e switch

```
# Go to a directory where you'd like to keep the source
admin:~ admin$ cd ~/projects/pyeapi

# Install
admin:~ admin$ sudo pip install -e ~/projects/pyeapi
```

Step 4: Install pyeapi requirements

```
# Go to a directory where you'd like to keep the source
admin:~ admin$ pip install -r dev-requirements.txt
```

Tip: If you start using pyeapi and get import errors, make sure your PYTHONPATH is set to include the path to pyeapi.

1.3.1 Development - Upgrade Pyeapi

```
admin:~ admin$ cd ~/projects/pyeapi
admin:~ admin$ git pull
```

Note: If you followed the directions above and used `pip install -e`, pip will automatically use the updated code.

CHAPTER 2

Quickstart

In order to use pyeapi, the EOS command API must be enabled using configuration mode. This library supports eAPI calls over both HTTP/S and UNIX Domain Sockets. Once the command API is enabled on the destination node, create a configuration file with the node properties. There are some nuances about the configuration file that are important to understand; take a minute and read about the *Pyeapi Configuration File*.

2.1 Enable EOS Command API

Refer to your official Arista EOS Configuration Guide to learn how to enable EOS Command API. Depending upon your software version, the options available include:

- HTTP
- HTTPS
- HTTP Local
- Unix Socket

2.2 Install Pyeapi

Follow the instructions on the *Installation* guide to prepare your node for pyeapi.

2.3 Create an eapi.conf file

Follow the instructions on the *Pyeapi Configuration File* guide to create a pyeapi configuration file. You can skip this step if you are running the pyeapi script on-box and Unix Sockets are enabled for EOS Command API.

2.4 Connect to a Node

The Python client for eAPI was designed to be easy to use and implement for writing tools and applications that interface with the Arista EOS management plane.

Once EOS is configured properly and the config file created, getting started with a connection to EOS is simple. Below demonstrates a basic connection using pyeapi. For more examples, please see the examples folder on Github.

This first example shows how to instantiate the Node object. The Node object provides some helpful methods and attributes to work with the switch.

```
# start by importing the library
import pyeapi

# create a node object by specifying the node to work with
node = pyeapi.connect_to('veos01')

# send one or more commands to the node
node.enable('show hostname')
[{'command': 'show hostname',
  'encoding': 'json',
  'result': {u'hostname': u'veos01',
             u'fqdn': u'veos01.arista.com'}}]

# Request a specific revision of a command that has been updated
node.enable({'cmd': 'show cvx', 'revision': 2})
[{'command': {'cmd': 'show cvx', 'revision': 2},
  'encoding': 'json',
  'result': {u'clusterMode': False,
             u'controllerUUID': u'',
             u'enabled': False,
             u'heartbeatInterval': 20.0,
             u'heartbeatTimeout': 60.0}}]

# use the config method to send configuration commands
node.config('hostname veos01')
[{}]

# multiple commands can be sent by using a list
# (works for both enable or config)
node.config(['interface Ethernet1', 'description foo'])
[{}, {}]

# return the running or startup configuration from the
# node (output omitted for brevity)
node.running_config

node.startup_config
```

The pyeapi library provides both a client for send and receiving commands over eAPI as well as an API for working directly with EOS resources. The API is designed to be easy and straightforward to use yet also extensible. Below is an example of working with the `vlans` API

```
# create a connection to the node
import pyeapi
node = pyeapi.connect_to('veos01')

# get the instance of the API (in this case vlans)
```

```
vlans = node.api('vlans')

# return all vlans from the node
vlans.getall()
{'1': {'state': 'active', 'name': 'default', 'vlan_id': 1, 'trunk_groups': []},
'10': {'state': 'active', 'name': 'VLAN0010', 'vlan_id': 10, 'trunk_groups':
[]}}

# return a specific vlan from the node
vlans.get(1)
{'state': 'active', 'name': 'default', 'vlan_id': 1, 'trunk_groups': []}

# add a new vlan to the node
vlans.create(100)
True

# set the new vlan name
vlans.set_name(100, 'foo')
True
```

CHAPTER 3

Pyeapi Configuration File

The pyeapi configuration file is a convenient place to store node connection information. By keeping connection information central, your pyeapi scripts can effortlessly reference nodes without any manual import of credentials or location information. Therefore, the pyeapi configuration file becomes a reflection of your switch inventory and the way in which the EOS Command API is enabled on the node. The following explains how to craft your pyeapi configuration file to address specific implementation styles.

Contents

3.1 eapi.conf Parameters

The following configuration options are available for defining node entries:

host The IP address or FQDN of the remote device. If the host parameter is omitted then the connection name is used

username The eAPI username to use for authentication (only required for http or https connections)

password The eAPI password to use for authentication (only required for http or https connections)

enablepwd The enable mode password if required by the destination node

transport Configures the type of transport connection to use. Valid values are:

- socket (default, available in EOS 4.14.5 or later)
- http_local (available in EOS 4.14.5 or later)
- http
- https

port Configures the port to use for the eAPI connection. A default port is used if this parameter is absent, based on the transport setting using the following values:

- transport: http, default port: 80
- transport: https, deafult port: 443
- transport: http_local, default port: 8080
- transport: socket, default port: n/a

3.2 When is an eapi.conf file needed?

It's important to understand the nuances of the pyeapi configuration file so you can simplify your implementation. Here's a quick summary of when the eapi.conf file is needed:

Transport	eapi.conf Required	Script run from	Authentication Required
http	Yes	On/Off-switch	Yes
https	Yes	On/Off-switch	Yes
http_local	Yes	On-switch only	No
socket	No	On-switch only	No

3.3 Where should the file be placed?

Search Order	Search Location
1	Environment Variable EAPI_CONF=/path/to/eapi.conf
2	$HOME/.eapi.conf
3	/mnt/flash/eapi.conf

Note: There is a slight difference in #2 `.eapi.conf` versus #3 `eapi.conf`

3.4 eapi.conf for On-box Implementations

This method would be used to run a pyeapi script on-box. In this mode, eAPI can be configured to require or not require authentication depending upon how you enabled EOS Command API.

Notice from the table above, that if EOS Command API Unix Sockets are enabled, or HTTP Local, you get the benefit of not needing to pass in credentials since the connection can only be made from localhost and it assumes the user has already authenticated to get on the switch.

3.4.1 Using Unix Sockets

This is the preferred method. The default transport for pyeapi is `socket` and the default host is `localhost`. Therefore, if running a pyeapi script on-box and have Unix Sockets enabled, you do not need an eapi.conf, nor do you need to pass any credentials (quite handy!).

Note: Unix Sockets are supported on EOS 4.14.5+

3.4.2 Using HTTP Local

As the table above indicates, a pyeapi configuration file is required in `/mnt/flash/eapi.conf`. It would contain something like:

```
[connection:localhost]
transport: http_local
```

3.4.3 Using HTTP or HTTPS

As the table above indicates, a pyeapi configuration file is required in `/mnt/flash/eapi.conf`. It would contain something like:

```
[connection:localhost]
transport: http[s]
username: admin
password: admin
```

3.5 eapi.conf for Off-box Implementations

This method would be used to run a pyeapi script from another server. In this mode, eAPI will require authentication. The only real option is whether you connect over HTTP or HTTPS.

Note: The `socket` and `http_local` transport options are not applicable.

Notice from the table above, that if EOS Command API Unix Sockets are enabled, or HTTP Local, you get the benefit of not needing to pass in credentials since the connection can only be made from localhost and it assumes the user has already authenticated to get on the switch.

3.5.1 Using HTTP or HTTPS

As the table above indicates, a pyeapi configuration file is required in `$HOME/.eapi.conf`. It would contain something like:

```
[connection:veos01]
transport: http
username: paul
password: nottelling
```

```
[connection:veos03]
transport: https
username: bob
password: mysecret

[connection:veos04]
host: 192.168.2.10
transport: https
username: admin
password: admin
```

3.6 The DEFAULT Section

The [DEFAULT] section can be used to gather globally applicable settings. Say that all of your nodes use the same transport or username, you can do something like:

```
[connection:veos01]

[connection:veos03]
transport: https
username: bob
password: mysecret

[connection:veos04]
host: 192.168.2.10

[DEFAULT]
transport: https
username: admin
password: admin
```

CHAPTER 4

Modules

4.1 Client

4.1.1 Client

Python Client for eAPI

This module provides the client for eAPI. It provides the primary functions for building applications that work with Arista EOS eAPI-enabled nodes. The first function is to provide a client for sending and receiving eAPI request and response objects on a per node basis. The second function provides a library for building API enabled data models for configuring EOS nodes.

This library allows for creating connections to EOS eAPI enabled nodes using the connect or connect_to function. Both functions will return an instance of a Node object that can be used to send and receive eAPI commands. The Node object can autoload API modules for a structured object oriented approach to configuring the EOS node with native Python objects.

Example

```
>>> import pyeapi
>>> conn = pyeapi.connect(host='10.1.1.1', transport='http')
>>> conn.execute(['show verson'])
{u'jsonrpc': u'2.0', u'result': [{u'memTotal': 2028008, u'version':
u'4.14.5F', u'internalVersion': u'4.14.5F-2209869.4145F', u'serialNumber':
u'', u'systemMacAddress': u'00:0c:29:f5:d2:7d', u'bootupTimestamp':
1421765066.11, u'memFree': 213212, u'modelName': u'vEOS', u'architecture':
u'i386', u'internalBuildId': u'f590eed4-1e66-43c6-8943-cee0390fbafe',
u'hardwareRevision': u''}], u'id': u'4312565648'}
```

```
>>> node = pyeapi.connect_to('veos01')
>>> node.enable('show version')
{u'jsonrpc': u'2.0', u'result': [{u'memTotal': 2028008, u'version':
```

```
u'4.14.5F', u'internalVersion': u'4.14.5F-2209869.4145F', u'serialNumber':
u'', u'systemMacAddress': u'00:0c:29:f5:d2:7d', u'bootupTimestamp':
1421765066.11, u'memFree': 213212, u'modelName': u'vEOS', u'architecture':
u'i386', u'internalBuildId': u'f590eed4-1e66-43c6-8943-cee0390fbafe',
u'hardwareRevision': u''}], u'id': u'4312565648'}
```

Additionally the node object can automatically load API modules to work with the resources in the configuration. The API autoloader supports automatic loading of modules in pyeapi.api as well as provides the ability to build custom API modules to be loaded from a different namespace.

Example

```
>>> import pyeapi
>>> node = pyeapi.connect_to('veos01')
>>> node.api('vlans').get(1)
{'state': 'active', 'name': 'default', 'vlan_id': 1, 'trunk_groups': []}
```

The API autoloader loads API modules by their filename.

The following objects are provide in this module for creating clients to interface with eAPI.

Node – Creates an instance of a node object that represents a single EOS device. Each EOS device to be managed should have a Node instance

Config – A subclass of ConfigParser.SafeConfigParser that handles the configuration file. The configuration file is an INI style file that contains the settings for nodes used by the connect_to function.

class pyeapi.client.**Config**(*filename=None*)

Bases: backports.configparser.ConfigParser

Conifguration instance for managing the eapi.conf file.

This class provides an instance for handling the configuration file. It should normally need to be instantiated. A single config object is instantiated by the module for working with the config.

filename

str – The full path to the loaded filename

Parameters **filename** (*str*) – The full path to the filename to be loaded when the object is instantiated.

add_connection(*name*, ***kwargs*)

Adds a connection to the configuration

This method will add a connection to the configuration. The connection added is only available for the lifetime of the object and is not persisted.

Note: If a call is made to load() or reload(), any connections added with this method must be re-added to the config instance

---

Parameters

- **name** (*str*) – The name of the connection to add to the config. The name provided will automatically be prepended with the string connection:
- ****kwargs** (*dict*) – configuration

autoload()
Loads the eapi.conf file

This method will use the module variable CONFIG_SEARCH_PATH to attempt to locate a valid eapi.conf file if a filename is not already configured. This method will load the first eapi.conf file it finds and then return.

The CONFIG_SEARCH_PATH can be overridden using an environment variable by setting EAPI_CONF.

connections
Returns all of the loaded connections names as a list

generate_tags()
Generates the tags with collection with hosts

get_connection(*name*)
Returns the properties for a connection name

This method will return the settings for the configuration specified by name. Note that the name argument should only be the name.

For instance, give the following eapi.conf file

```
[connection:veos01]
transport: http
```

The name to use to retrieve the configuration would be veos01

```
>>> pyeapi.client.config.get_connection('veos01')
```

Parameters **name** (*str*) – The name of the connection to return

Returns

A Python dictionary object of key/value pairs that represent the node configuration. If the name provided in the argument is not found, then None is returned.

load(*filename*)
Loads the file specified by filename

This method works in conjunction with the autoload method to load the file specified by filename.

Parameters **filename** (*str*) – The full path to the file to be loaded

read(*filename*)
Reads the file specified by filename

This method will load the eapi.conf file specified by filename into the instance object. It will also add the default connection localhost if it was not defined in the eapi.conf file

Parameters **filename** (*str*) – The full path to the file to load

reload()
Reloades the configuration

This method will reload the configuration instance using the last known filename. Note this method will initially clear the configuration and reload all entries.

class pyeapi.client.**Node**(*connection*, ***kwargs*)
Bases: object

Represents a single device for sending and receiving eAPI messages

The Node object provides an instance for communicating with Arista EOS devices. The Node object provides easy to use methods for sending both enable and config commands to the device using a specific transport. This object forms the base for communicating with devices.

connection
EapiConnection – The connection property represents the underlying transport used by the Node object to communicate with the device using eAPI.

running_config
str – The running-config from the device. This property is lazily loaded and refreshed over the life cycle of the instance.

startup_config
str – The startup-config from the device. This property is lazily loaded and refreshed over the life cycle of the instance.

autorefresh
bool – If True, the running-config and startup-config are refreshed on config events. If False, then the config properties must be manually refreshed.

settings
dict – Provides access to the settings used to create the Node instance.

Parameters

- **connection** (EapiConnection) – An instance of EapiConnection used as the transport for sending and receiving eAPI requests and responses.
- ****kwargs** – An arbitrary list of keyword arguments

api(*name*, *namespace='pyeapi.api'*)
Loads the specified api module

This method is the API autoload mechanism that will load the API module specified by the name argument. The API module will be loaded and look first for an initialize() function and secondly for an instance() function. In both cases, the node object is passed to the module.

Parameters

- **name** (*str*) – The name of the module to load. The name should be the name of the python file to import
- **namespace** (*str*) – The namespace to use to load the module. The default value is 'pyeapi.api'

Returns The API module loaded with the node instance.

config(*commands*, ***kwargs*)
Configures the node with the specified commands

This method is used to send configuration commands to the node. It will take either a string or a list and prepend the necessary commands to put the session into config mode.

Parameters

- **commands** (*str, list*) – The commands to send to the node in config mode. If the commands argument is a string it will be cast to a list. The list of commands will also be prepended with the necessary commands to put the session in config mode.
- ****kwargs** – Additional keyword arguments for expanded eAPI functionality. Only supported eAPI params are used in building the request

Returns

The config method will return a list of dictionaries with the output from each command. The function will strip the response from any commands it prepends.

connection

enable(*commands*, *encoding='json'*, *strict=False*, *send_enable=True*, ***kwargs*)
Sends the array of commands to the node in enable mode

This method will send the commands to the node and evaluate the results. If a command fails due to an encoding error, then the command set will be re-issued individual with text encoding.

Parameters

- **commands** (*list*) – The list of commands to send to the node
- **encoding** (*str*) – The requested encoding of the command output. Valid values for encoding are JSON or text
- **strict** (*bool*) – If False, this method will attempt to run a command with text encoding if JSON encoding fails
- **send_enable** (*bool*) – If True the enable command will be prepended to the command list automatically.
- ****kwargs** – Additional keyword arguments for expanded eAPI functionality. Only supported eAPI params are used in building the request

Returns

A dict object that includes the response for each command along with the encoding

Raises

- TypeError – This method does not support sending configure commands and will raise a TypeError if configuration commands are found in the list of commands provided

  This method will also raise a TypeError if the specified encoding is not one of 'json' or 'text'
- CommandError – This method will raise a CommandError if any one of the commands fails.

enable_authentication(*password*)
Configures the enable mode authentication password

EOS supports an additional password authentication mechanism for sessions that want to switch to executive (or enable) mode. This method will configure the password, if required, for entering executive mode

Parameters **password** (*str*) – The password string in clear text used to authenticate to exec mode

get_config(*config='running-config'*, *params=None*, *as_string=False*)
Retreives the config from the node

This method will retrieve the config from the node as either a string or a list object. The config to retrieve can be specified as either the startup-config or the running-config.

Parameters

- **config** (*str*) – Specifies to return either the nodes startup-config or running-config. The default value is the running-config
- **params** (*str*) – A string of keywords to append to the command for retrieving the config.

- **as_string** (*boo*) – Flag that determines the response. If True, then the configuration is returned as a raw string. If False, then the configuration is returned as a list. The default value is False

Returns This method will return either a string or a list depending on the states of the as_string keyword argument.

Raises `TypeError` – If the specified config is not one of either 'running-config' or 'startup-config'

model

refresh()
Refreshes the instance config properties

This method will refresh the public running_config and startup_config properites. Since the properties are lazily loaded, this method will clear the current internal instance variables. One the next call the instance variables will be repopulated with the current config

run_commands(*commands*, *encoding='json'*, *send_enable=True*, ***kwargs*)
Sends the commands over the transport to the device

This method sends the commands to the device using the nodes transport. This is a lower layer function that shouldn't normally need to be used, preferring instead to use config() or enable().

Parameters

- **commands** (*list*) – The ordered list of commands to send to the device using the transport
- **encoding** (*str*) – The encoding method to use for the request and excpected response.
- **send_enable** (*bool*) – If True the enable command will be prepended to the command list automatically.
- ****kwargs** – Additional keyword arguments for expanded eAPI functionality. Only supported eAPI params are used in building the request

Returns

This method will return the raw response from the connection which is a Python dictionary object.

running_config

section(*regex*, *config='running_config'*)
Returns a section of the config

Parameters

- **regex** (*str*) – A valid regular expression used to select sections of configuration to return
- **config** (*str*) – The configuration to return. Valid values for config are "running_config" or "startup_config". The default value is "running_config"

Returns The configuration section as a string object.

startup_config

version

version_number

pyeapi.client.**config_for**(*name*)
Function to get settings for named config

This function will return the settings for a specific connection as specified by name. Its a convenience function that calls get_connection on the global config instance

Parameters **name** (*str*) – The name of the connection to return. The connection name is specified as the string right of the : in the INI file

Returns

A Python dictionary object of key/value pairs that represent the nodes configuration settings from the config instance

pyeapi.client.**connect**(*transport=None*, *host='localhost'*, *username='admin'*, *password=''*, *port=None*, *timeout=60*, *return_node=False*, ***kwargs*)
Creates a connection using the supplied settings

This function will create a connection to an Arista EOS node using the arguments. All arguments are optional with default values.

Parameters

- **transport** (*str*) – Specifies the type of connection transport to use. Valid values for the connection are socket, http_local, http, and https. The default value is specified in DEFAULT_TRANSPORT
- **host** (*str*) – The IP addres or DNS host name of the connection device. The default value is 'localhost'
- **username** (*str*) – The username to pass to the device to authenticate the eAPI connection. The default value is 'admin'
- **password** (*str*) – The password to pass to the device to authenticate the eAPI connection. The default value is ''
- **port** (*int*) – The TCP port of the endpoint for the eAPI connection. If this keyword is not specified, the default value is automatically determined by the transport type. (http=80, https=443)
- **return_node** (*bool*) – Returns a Node object if True, otherwise returns an EapiConnection object.

Returns An instance of an EapiConnection object for the specified transport.

pyeapi.client.**connect_to**(*name*)
Creates a node instance based on an entry from the config

This function will retrieve the settings for the specified connection from the config and return a Node instance. The configuration must be loaded prior to calling this function.

Parameters **name** (*str*) – The name of the connection to load from the config. The name argument should be the connection name (everything right of the colon from the INI file)

Returns

This function will return an instance of Node with the settings from the config instance.

Raises AttributeError – raised if the specified configuration name is not found in the loaded configuration

pyeapi.client.**hosts_for_tag**(*tag*)
Returns the hosts assocated with the specified tag

This function will return the hosts assoicated with the tag specified in the argument. It will return an array of connecition names.

Parameters **tag** (*str*) – The name of the tag to retrieve the list of hosts for

Returns

A Python list object that includes the list of hosts assoicated with the specified tag.

None: If the specified tag does not exist, then None is returned.

Return type list

pyeapi.client.**load_config**(*filename*)
Function method that loads a conf file

This function will load the file specified by filename into the config instance. Its a convenience function that calls load on the config instance

Parameters **filename** (*str*) – The full path to the filename to load

pyeapi.client.**make_connection**(*transport*, ***kwargs*)
Creates a connection instance based on the transport

This function creates the EapiConnection object based on the desired transport. It looks up the transport class in the TRANSPORTS global dictionary.

Parameters

- **transport** (*string*) – The transport to use to create the instance.
- ****kwargs** – Arbitrary keyword arguments.

Returns An instance of a connection object based on the transport

Raises TypeError – A TypeError is raised if the transport keyword is not found in the list (keys) of available transports.

4.1.2 Eapilib

Provides wrapper for eAPI calls

This module provides a connection to eAPI by wrapping eAPI calls in an instance of Connection. The connection module provides an easy implementation for sending and receiving calls over eAPI using a HTTP/S transport.

exception pyeapi.eapilib.**CommandError**(*code*, *message*, ***kwargs*)
Bases: *pyeapi.eapilib.EapiError*

Base exception raised for command errors

The CommandError instance provides a custom exception that can be used if the eAPI command(s) fail. It provides some additional information that can be used to understand what caused the exception.

Parameters

- **error_code** (*int*) – The error code returned from the eAPI call.
- **error_text** (*string*) – The error text message that coincides with the error_code
- **commands** (*array*) – The list of commands that were sent to the node that generated the error
- **message** (*string*) – The exception error message which is a concatenation of the error_code and error_text

get_trace()

trace

exception pyeapi.eapilib.**ConnectionError**(*connection_type*, *message*, *commands=None*)
Bases: *pyeapi.eapilib.EapiError*

Base exception raised for connection errors

Connection errors are raised when a connection object is unable to connect to the node. Typically these errors can result from using the wrong transport type or not providing valid credentials.

Parameters

- **commands** (*array*) – The list of commands there were sent to the node that when the exception was raised
- **connection_type** (*string*) – The string identifier for the connection object that generate the error
- **message** (*string*) – The exception error message
- **response** (*string*) – The message generate from the response packet

class pyeapi.eapilib.**EapiConnection**
Bases: object

Creates a connection to eAPI for sending and receiving eAPI requests

The EapiConnection object provides an implementation for sending and receiving eAPI requests and responses. This class should not need to be instantiated directly.

authentication(*username*, *password*)
Configures the user authentication for eAPI

This method configures the username and password combination to use for authenticating to eAPI.

Parameters

- **username** (*str*) – The username to use to authenticate the eAPI connection with
- **password** (*str*) – The password in clear text to use to authenticate the eAPI connection with

execute(*commands*, *encoding='json'*, ***kwargs*)
Executes the list of commands on the destination node

This method takes a list of commands and sends them to the destination node, returning the results. The execute method handles putting the destination node in enable mode and will pass the enable password, if required.

Parameters

- **commands** (*list*) – A list of commands to execute on the remote node
- **encoding** (*string*) – The encoding to send along with the request message to the destination node. Valid values include 'json' or 'text'. This argument will influence the response object encoding
- ****kwargs** – Arbitrary keyword arguments

Returns A decoded response message as a native Python dictionary object that has been deserialized from JSON.

Raises *CommandError* – A CommandError is raised that includes the error code, error message along with the list of commands that were sent to the node. The exception instance is also stored in the error property and is availble until the next request is sent

request(*commands*, *encoding=None*, *reqid=None*, ***kwargs*)

Generates an eAPI request object

This method will take a list of EOS commands and generate a valid eAPI request object form them. The eAPI request object is then JSON encoding and returned to the caller.

eAPI Request Object

```
{
    "jsonrpc": "2.0",
    "method": "runCmds",
    "params": {
        "version": 1,
        "cmds": [
            <commands>
        ],
        "format": [json, text],
    }
    "id": <reqid>
}
```

Parameters

- **commands** (*list*) – A list of commands to include in the eAPI request object
- **encoding** (*string*) – The encoding method passed as the *format* parameter in the eAPI request
- **reqid** (*string*) – A custom value to assign to the request ID field. This value is automatically generated if not passed
- ****kwargs** – Additional keyword arguments for expanded eAPI functionality. Only supported eAPI params are used in building the request

Returns A JSON encoding request structure that can be send over eAPI

send(*data*)

Sends the eAPI request to the destination node

This method is responsible for sending an eAPI request to the destination node and returning a response based on the eAPI response object. eAPI responds to request messages with either a success message or failure message.

eAPI Response - success

```
{
    "jsonrpc": "2.0",
    "result": [
        {},
        {}
        {
            "warnings": [
                <message>
            ]
        },
    ],
    "id": <reqid>
}
```

eAPI Response - failure

```
{
    "jsonrpc": "2.0",
    "error": {
        "code": <int>,
        "message": <string>
        "data": [
            {},
            {},
            {
                "errors": [
                    <message>
                ]
            }
        ]
    }
    "id": <reqid>
}
```

Parameters **data** (*string*) – The data to be included in the body of the eAPI request object

Returns

A decoded response. The response object is deserialized from JSON and returned as a standard Python dictionary object

Raises *CommandError if an eAPI failure response object is returned from* – the node. The CommandError exception includes the error code and error message from the eAPI response.

exception pyeapi.eapilib.**EapiError**(*message*, *commands=None*)
Bases: exceptions.Exception

Base exception class for all exceptions generated by eapilib

This is the base exception class for all exceptions generated by eapilib. It is provided as a catch all for exceptions and should not be directly raised by an methods or functions

Parameters

- **commands** (*array*) – The list of commands there were sent to the node that when the exception was raised
- **message** (*string*) – The exception error message

class pyeapi.eapilib.**HttpConnection**(*path*, **args*, ***kwargs*)
Bases: httplib.HTTPConnection

class pyeapi.eapilib.**HttpEapiConnection**(*host*, *port=None*, *path=None*, *username=None*, *password=None*, *timeout=60*, ***kwargs*)
Bases: *pyeapi.eapilib.EapiConnection*

class pyeapi.eapilib.**HttpLocalEapiConnection**(*port=None*, *path=None*, *timeout=60*, ***kwargs*)
Bases: *pyeapi.eapilib.EapiConnection*

class pyeapi.eapilib.**HttpsConnection**(*path*, **args*, ***kwargs*)
Bases: httplib.HTTPSConnection

class pyeapi.eapilib.**HttpsEapiConnection**(*host*, *port=None*, *path=None*, *username=None*, *password=None*, *context=None*, *timeout=60*, ***kwargs*)
Bases: *pyeapi.eapilib.EapiConnection*

disable_certificate_verification()

class pyeapi.eapilib.**SocketConnection**(*path*, *timeout=60*)
Bases: httplib.HTTPConnection

connect()

class pyeapi.eapilib.**SocketEapiConnection**(*path=None*, *timeout=60*, ***kwargs*)
Bases: *pyeapi.eapilib.EapiConnection*

pyeapi.eapilib.**https_connection_factory**(*path*, *host*, *port*, *context=None*, *timeout=60*)

4.1.3 Utils

class pyeapi.utils.**ProxyCall**(*proxy*, *method*)
Bases: object

pyeapi.utils.**collapse_range**(*arg*, *value_delimiter=', '*, *range_delimiter='-'*)
Collapses a list of values into a range set

Parameters

- **arg** – The list of values to collapse
- **value_delimiter** – The delimiter that separates values
- **range_delimiter** – The delimiter that separates a value range

Returns An array of collapsed string values

Return type list

pyeapi.utils.**debug**(*text*)
Log a message to syslog and stderr

Parameters **text** (*str*) – The string object to print

pyeapi.utils.**expand_range**(*arg*, *value_delimiter=', '*, *range_delimiter='-'*)
Expands a delimited string of ranged integers into a list of strings

Parameters

- **arg** – The string range to expand
- **value_delimiter** – The delimiter that separates values
- **range_delimiter** – The delimiter that signifies a range of values

Returns An array of expanded string values

Return type list

pyeapi.utils.**import_module**(*name*)
Imports a module into the current runtime environment

This function emulates the Python import system that allows for importing full path modules. It will break down the module and import each part (or skip if it is already loaded in cache).

Parameters **name** (*str*) – The name of the module to import. This should be the full path of the module

Returns The module that was imported

pyeapi.utils.**islocalconnection**()
Checks if running locally on EOS device or remotely

This function will return a boolean indicating if the current execution environment is running locally on an EOS device (True) or running remotely and communicating over HTTP/S (False)

Returns

A boolean value that indicates whether or not the current thread is local or remote

pyeapi.utils.**load_module**(*name*)
Attempts to load a module into the current environment

This function will load a module specified by name. The module name is first checked to see if it is already loaded and will return the module if it is. If the module hasn't been previously loaded it will attempt to import it

Parameters **name** (*str*) – Specifies the full name of the module. For instance pyeapi.api.vlans

Returns The module that has been imported or retrieved from the sys modules

pyeapi.utils.**lookahead**(*it*)

pyeapi.utils.**make_iterable**(*value*)
Converts the supplied value to a list object

This function will inspect the supplied value and return an iterable in the form of a list.

Parameters **value** (*object*) – An valid Python object

Returns An iterable object of type list

4.2 Api

4.2.1 Abstract

Provides an abstract implementation for building API modules

This module provides a set of classes that are used to build API modules that work with Node objects. Using this module will allow the API modules to be automatically loaded using the Node.api method.

The classes in this module should not be instantiated directly but rather provide parent class for API implementations. All API modules will ultimately derive from BaseEntity which provides some common functions to make building API modules easier.

class pyeapi.api.abstract.**BaseEntity**(*node*)
Bases: object

Base class for all resources to derive from

This BaseEntity class should not be directly instatiated. It is designed to be implemented by all resource classes to provide common methods.

node
Node – The node instance this resource will perform operations against for configuration

config
Config – Returns an instance of Config with the nodes current running configuration

error
CommandError – Holds the latest CommandError exception instance if raised

Parameters **node** (Node) – An instance of Node

command_builder(*string*, *value=None*, *default=None*, *disable=None*)
Builds a command with keywords

Notes

Negating a command string by overriding 'value' with None or an assigned value that evalutates to false has been deprecated. Please use 'disable' to negate a command.

Parameters are evaluated in the order 'default', 'disable', 'value'

Parameters

- **string** (*str*) – The command string
- **value** (*str*) – The configuration setting to subsititue into the command string. If value is a boolean and True, just the command string is used
- **default** (*bool*) – Specifies the command should use the default keyword argument. Default preempts disable and value.
- **disable** (*bool*) – Specifies the command should use the no keyword argument. Disable preempts value.

Returns A command string that can be used to configure the node

config

configure(*commands*)
Sends the commands list to the node in config mode

This method performs configuration the node using the array of commands specified. This method wraps the configuration commands in a try/except block and stores any exceptions in the error property.

Note: If the return from this method is False, use the error property to investigate the exception

---

Parameters **commands** (*list*) – A list of commands to be sent to the node in config mode

Returns

True if the commands are executed without exception otherwise False is returned

configure_interface(*name*, *commands*)
Configures the specified interface with the commands

Parameters

- **name** (*str*) – The interface name to configure
- **commands** – The commands to configure in the interface

Returns True if the commands completed successfully

error

get_block(*parent*, *config='running_config'*)
Scans the config and returns a block of code

Parameters

- **parent** (*str*) – The parent string to search the config for and return the block

- **config** (*str*) – A text config string to be searched. Default is to search the running-config of the Node.

Returns A string object that represents the block from the config. If the parent string is not found, then this method will return None.

class pyeapi.api.abstract.**Entity**(*node*)
Bases: *pyeapi.api.abstract.BaseEntity*, _abcoll.Callable

Abstract class for building Entity resources

The Entity class provides an abstract implementation that allows for building an API configuration resource. The Entity class should not be directly instantiated. It is used in instances where a single config entity is appropriate in the configuration.

Examples of Entity candidates include global spanning tree

get()

class pyeapi.api.abstract.**EntityCollection**(*node*)
Bases: *pyeapi.api.abstract.BaseEntity*, _abcoll.Mapping

Abstract class for building EntityCollection resources

The EntityCollection class provides an abstract implementat that allows for building API configuration resources with multiple resources. The EntityCollection class should not be directly instantiated.

Examples of an EntityCollection candidate include VLANs and interfaces

get(*name*, *default=None*)

getall()

4.2.2 Acl

Module for working with EOS access control list resources

This module provides an implementation for configuring and managing access access control lists on Arista EOS nodes. Access control lists can be specified as either 'standard' or 'extended' ACLs. This module provides the following class implementations:

- **Acls – The top-level class used to manage both standard and extended** access control lists in EOS
- StandardAcls – Class that manages the set of standard ACLs
- ExtendedAcls – Class that manages the set of extended ACLs

class pyeapi.api.acl.**Acls**(*node*, **args*, ***kwargs*)
Bases: *pyeapi.api.abstract.EntityCollection*

create(*name*, *type='standard'*)

create_instance(*name*, *acl_type*)

get(*name*)

get_instance(*name*)

getall()
Returns all ACLs in a dict object.

Returns

A Python dictionary object containing all ACL configuration indexed by ACL name:

```
{
    "<ACL1 name>": {...},
    "<ACL2 name>": {...}
}
```

marshall(*name*, **args*, ***kwargs*)

class pyeapi.api.acl.**ExtendedAcls**(*node*)
Bases: *pyeapi.api.abstract.EntityCollection*

add_entry(*name*, *action*, *protocol*, *srcaddr*, *srcprefixlen*, *dstaddr*, *dstprefixlen*, *log=False*, *seqno=None*)

create(*name*)

default(*name*)

delete(*name*)

entry_re = <_sre.SRE_Pattern object at 0x37e8440>

get(*name*)

remove_entry(*name*, *seqno*)

update_entry(*name*, *seqno*, *action*, *protocol*, *srcaddr*, *srcprefixlen*, *dstaddr*, *dstprefixlen*, *log=False*)

class pyeapi.api.acl.**StandardAcls**(*node*)
Bases: *pyeapi.api.abstract.EntityCollection*

add_entry(*name*, *action*, *addr*, *prefixlen*, *log=False*, *seqno=None*)

create(*name*)

default(*name*)

delete(*name*)

entry_re = <_sre.SRE_Pattern object at 0x3794be0>

get(*name*)

remove_entry(*name*, *seqno*)

update_entry(*name*, *seqno*, *action*, *addr*, *prefixlen*, *log=False*)

pyeapi.api.acl.**instance**(*node*)

pyeapi.api.acl.**mask_to_prefixlen**(*mask*)
Converts a subnet mask from dotted decimal to bit length

Parameters **mask** (*str*) – The dotted decimal subnet mask to convert

Returns The subnet mask as a valid prefix length

Return type str

pyeapi.api.acl.**prefixlen_to_mask**(*prefixlen*)
Converts a prefix length to a dotted decimal subnet mask

Parameters **prefixlen** (*str*) – The prefix length value to convert

Returns The subt mask as a dotted decimal string

Return type str

4.2.3 Bgp

API module for Bgp

class pyeapi.api.bgp.**Bgp**(**args*, ***kwargs*)
Bases: *pyeapi.api.abstract.Entity*

The Bgp class implements global BGP router configuration

add_network(*prefix*, *length*, *route_map=None*)

configure_bgp(*cmd*)

create(*bgp_as*)

default()

delete()

get()
Returns the bgp routing configuration as a dict object

neighbors

remove_network(*prefix*, *masklen*, *route_map=None*)

set_maximum_paths(*max_path=None*, *max_ecmp_path=None*, *default=False*, *disable=False*)

set_router_id(*value=None*, *default=False*, *disable=False*)

set_shutdown(*default=False*, *disable=True*)

class pyeapi.api.bgp.**BgpNeighbors**(*node*)
Bases: *pyeapi.api.abstract.EntityCollection*

command_builder(*name*, *cmd*, *value*, *default*, *disable*)

configure(*cmd*)

create(*name*)

delete(*name*)

get(*name*)

getall()

ispeergroup(*name*)

set_description(*name*, *value=None*, *default=False*, *disable=False*)

set_next_hop_self(*name*, *value=None*, *default=False*, *disable=False*)

set_peer_group(*name*, *value=None*, *default=False*, *disable=False*)

set_remote_as(*name*, *value=None*, *default=False*, *disable=False*)

set_route_map_in(*name*, *value=None*, *default=False*, *disable=False*)

set_route_map_out(*name*, *value=None*, *default=False*, *disable=False*)

set_send_community(*name*, *value=None*, *default=False*, *disable=False*)

set_shutdown(*name*, *default=False*, *disable=True*)

class pyeapi.api.bgp.**Network**(*prefix*, *length*, *route_map*)
Bases: tuple

length
Alias for field number 1

prefix
Alias for field number 0

route_map
Alias for field number 2

pyeapi.api.bgp.**instance**(*api*)

4.2.4 Interfaces

Module for working with interfaces in EOS

This module provides an API for pragmatically working with EOS interface configurations. Interfaces include any data or management plane interface available in EOS.

param name The name of the interface the configuration should be applied to. The interface name is the full interface identifier.

type name string

param shutdown True if the interface is administratively disabled, and False if the interface is administratively enable. This value does not validate the interfaces operational state.

type shutdown boolean

param description The interface description string. This value is an arbitrary operator defined value.

type description string

param sflow True if sFlow is enabled on the interface otherwise False

type sflow boolean

param flowcontrol_send The flowcontrol send configuration value for the interface. Valid values are on or off

type flowcontrol_send string

param flowcontrol_receive The flowcontrol receive configuration value for the interface. Valid values are on or off

type flowcontrol_receive string

class pyeapi.api.interfaces.**BaseInterface**(*node*)
Bases: *pyeapi.api.abstract.EntityCollection*

create(*name*)
Creates a new interface on the node

> **Note:** This method will attempt to create the interface regardless if the interface exists or not. If the interface already exists then this method will still return True

Parameters **name** (*string*) – The full name of the interface.

Returns True if the interface could be created otherwise False (see Note)

default(*name*)
Defaults an interface in the running configuration

Parameters **name** (*string*) – The full name of the interface

Returns True if the command operation succeeds otherwise False

delete(*name*)
Deletes the interface from the node

Note: This method will attempt to delete the interface from the nodes operational config. If the interface does not exist then this method will not perform any changes but still return True

---

Parameters **name** (*string*) – The full name of the interface

Returns True if the interface could be deleted otherwise False (see Node)

get(*name*)
Returns a generic interface as a set of key/value pairs

This class is should normally serve as a base class for building more specific interface resources. The attributes of this resource are common to all interfaces regardless of type in EOS.

The generic interface resource returns the following:

- name (str): The name of the interface
- type (str): Always returns 'generic'
- shutdown (bool): True if the interface is shutdown
- description (str): The interface description value

Parameters **name** (*str*) – The interface identifier to retrieve from the running-configuration

Returns

A Python dictionary object of key/value pairs that represents the interface configuration. If the specified interface does not exist, then None is returned

set_description(*name*, *value=None*, *default=False*, *disable=False*)
Configures the interface description

EosVersion: 4.13.7M

Parameters

- **name** (*string*) – The interface identifier. It must be a full interface name (ie Ethernet, not Et)
- **value** (*string*) – The value to set the description to.
- **default** (*boolean*) – Specifies to default the interface description
- **disable** (*boolean*) – Specifies to negate the interface description

Returns True if the operation succeeds otherwise False

set_encapsulation(*name*, *vid*, *default=False*, *disable=False*)
Configures the subinterface encapsulation value

Parameters

- **name** (*string*) – The interface identifier. It must be a full interface name (ie Ethernet, not Et)

- **vid** (*int*) – The vlan id number
- **default** (*boolean*) – Specifies to default the subinterface encapsulation
- **disable** (*boolean*) – Specifies to disable the subinterface encapsulation

Returns True if the operation succeeds otherwise False is returned

set_shutdown(*name*, *default=False*, *disable=True*)
Configures the interface shutdown state

Default configuration for set_shutdown is disable=True, meaning 'no shutdown'. Setting both default and disable to False will effectively enable shutdown on the interface.

Parameters

- **name** (*string*) – The interface identifier. It must be a full interface name (ie Ethernet, not Et)
- **default** (*boolean*) – Specifies to default the interface shutdown
- **disable** (*boolean*) – Specifies to disable interface shutdown, i.e. disable=True => no shutdown

Returns True if the operation succeeds otherwise False is returned

class pyeapi.api.interfaces.**EthernetInterface**(*node*)
Bases: *pyeapi.api.interfaces.BaseInterface*

create(*name*)
Create an Ethernet subinterface

Parameters **name** (*string*) – The subinterface name. Ex: Ethernet1.1

Raises

- NotImplementedError – creating physical Ethernet interfaces is not
- supported. Only subinterfaces can be created.

delete(*name*)
Delete an Ethernet subinterfaces

Parameters **name** (*string*) – The subinterface name. Ex: Ethernet1.1

Raises

- NotImplementedError – creating physical Ethernet interfaces is not
- supported. Only subinterfaces can be created.

get(*name*)
Returns an interface as a set of key/value pairs

Parameters **name** (*string*) – the interface identifier to retrieve the from the configuration

Returns

A Python dictionary object of key/value pairs that represent the current configuration for the specified node. If the specified interface name does not exist, then None is returned:

```
{
    "name": <string>,
    "type": "ethernet",
    "sflow": [true, false],
    "flowcontrol_send": [on, off],
```

```
    "flowcontrol_receive": [on, off]
}
```

set_flowcontrol(*name*, *direction*, *value=None*, *default=False*, *disable=False*)
Configures the interface flowcontrol value

Parameters

- **name** (*string*) – The interface identifier. It must be a full interface name (ie Ethernet, not Et)
- **direction** (*string*) – one of either 'send' or 'receive'
- **value** (*boolean*) – True if the interface should enable flow control packet handling, otherwise False
- **default** (*boolean*) – Specifies to default the interface flow control send or receive value
- **disable** (*boolean*) – Specifies to disable the interface flow control send or receive value

Returns True if the operation succeeds otherwise False is returned

set_flowcontrol_receive(*name*, *value=None*, *default=False*, *disable=False*)
Configures the interface flowcontrol receive value

Parameters

- **name** (*string*) – The interface identifier. It must be a full interface name (ie Ethernet, not Et)
- **value** (*boolean*) – True if the interface should enable receiving flow control packets, otherwise False
- **default** (*boolean*) – Specifies to default the interface flow control receive value
- **disable** (*boolean*) – Specifies to disable the interface flow control receive value

Returns True if the operation succeeds otherwise False is returned

set_flowcontrol_send(*name*, *value=None*, *default=False*, *disable=False*)
Configures the interface flowcontrol send value

Parameters

- **name** (*string*) – The interface identifier. It must be a full interface name (ie Ethernet, not Et)
- **value** (*boolean*) – True if the interface should enable sending flow control packets, otherwise False
- **default** (*boolean*) – Specifies to default the interface flow control send value
- **disable** (*boolean*) – Specifies to disable the interface flow control send value

Returns True if the operation succeeds otherwise False is returned

set_sflow(*name*, *value=None*, *default=False*, *disable=False*)
Configures the sFlow state on the interface

Parameters

- **name** (*string*) – The interface identifier. It must be a full interface name (ie Ethernet, not Et)

- **value** (*boolean*) – True if sFlow should be enabled otherwise False
- **default** (*boolean*) – Specifies the default value for sFlow
- **disable** (*boolean*) – Specifies to disable sFlow

Returns True if the operation succeeds otherwise False is returned

set_vrf(*name*, *vrf*, *default=False*, *disable=False*)

Applies a VRF to the interface

Note: VRF being applied to interface must already exist in switch config. Ethernet port must be in routed mode. This functionality can also be handled in the VRF api.

Parameters

- **name** (*str*) – The interface identifier. It must be a full interface name (ie Ethernet, not Et)
- **vrf** (*str*) – The vrf name to be applied to the interface
- **default** (*bool*) – Specifies the default value for VRF
- **disable** (*bool*) – Specifies to disable VRF

Returns True if the operation succeeds otherwise False is returned

class pyeapi.api.interfaces.**Interfaces**(*node*, **args*, ***kwargs*)

Bases: *pyeapi.api.abstract.EntityCollection*

get(*name*)

get_instance(*interface*)

getall()

Returns all interfaces in a dict object.

Returns

A Python dictionary object containing all interface configuration indexed by interface name:

```
{
    "Ethernet1": {...},
    "Ethernet2": {...}
}
```

marshall(*name*, **args*, ***kwargs*)

class pyeapi.api.interfaces.**PortchannelInterface**(*node*)

Bases: *pyeapi.api.interfaces.BaseInterface*

get(*name*)

Returns a Port-Channel interface as a set of key/value pairs

Parameters **name** (*str*) – The interface identifier to retrieve from the running-configuration

Returns

A Python dictionary object of key/value pairs that represents the interface configuration. If the specified interface does not exist, then None is returned:

```
{
    "name": <string>,
    "type": "portchannel",
    "members": <arrary of interface names>,
```

```
    "minimum_links: <integer>,
    "lacp_mode": [on, active, passive]
}
```

get_lacp_mode(*name*)
Returns the LACP mode for the specified Port-Channel interface

Parameters **name** (*str*) – The Port-Channel interface name to return the LACP mode for from the configuration

Returns

The configured LACP mode for the interface. Valid mode values are 'on', 'passive', 'active'

get_members(*name*)
Returns the member interfaces for the specified Port-Channel

Parameters **name** (*str*) – The Port-channel interface name to return the member interfaces for

Returns

A list of physical interface names that belong to the specified interface

set_lacp_fallback(*name*, *mode=None*)
Configures the Port-Channel lacp_fallback

Parameters

- **name** (*str*) – The Port-Channel interface name
- **mode** (*str*) – The Port-Channel LACP fallback setting Valid values are 'disabled', 'static', 'individual':
  - static - Fallback to static LAG mode
  - individual - Fallback to individual ports
  - disabled - Disable LACP fallback

Returns True if the operation succeeds otherwise False is returned

set_lacp_mode(*name*, *mode*)
Configures the LACP mode of the member interfaces

Parameters

- **name** (*str*) – The Port-Channel interface name to configure the LACP mode
- **mode** (*str*) – The LACP mode to configure the member interfaces to. Valid values are 'on, 'passive', 'active'

Returns True if the operation succeeds otherwise False

set_lacp_timeout(*name*, *value=None*)

Configures the Port-Channel LACP fallback timeout The fallback timeout configures the period an interface in fallback mode remains in LACP mode without receiving a PDU.

Parameters

- **name** (*str*) – The Port-Channel interface name
- **value** (*int*) – port-channel lacp fallback timeout in seconds

Returns True if the operation succeeds otherwise False is returned

set_members(*name*, *members*, *mode=None*)
Configures the array of member interfaces for the Port-Channel

Parameters

- **name** (*str*) – The Port-Channel interface name to configure the member interfaces
- **members** (*list*) – The list of Ethernet interfaces that should be member interfaces
- **mode** (*str*) – The LACP mode to configure the member interfaces to. Valid values are 'on, 'passive', 'active'. When there are existing channel-group members and their lacp mode differs from this attribute, all of those members will be removed and then re-added using the specified lacp mode. If this attribute is omitted, the existing lacp mode will be used for new member additions.

Returns True if the operation succeeds otherwise False

set_minimum_links(*name*, *value=None*, *default=False*, *disable=False*)
Configures the Port-Channel min-links value

Parameters

- **name** (*str*) – The Port-Channel interface name
- **value** (*str*) – The value to configure the min-links
- **default** (*bool*) – Specifies to default the port channel min-links value
- **disable** (*bool*) – Specifies to disable the port channel min-links value

Returns True if the operation succeeds otherwise False is returned

class pyeapi.api.interfaces.**VxlanInterface**(*node*)
Bases: *pyeapi.api.interfaces.BaseInterface*

DEFAULT_MCAST_GRP = ''

DEFAULT_SRC_INTF = ''

add_vtep(*name*, *vtep*, *vlan=None*)
Adds a new VTEP endpoint to the global or local flood list

EosVersion: 4.13.7M

Parameters

- **name** (*str*) – The name of the interface to configure
- **vtep** (*str*) – The IP address of the remote VTEP endpoint to add
- **vlan** (*str*) – The VLAN ID associated with this VTEP. If the VLAN
- **is used, then the VTEP is configured as a local flood** (*keyword*) –
- **endpoing** –

Returns True if the command completes successfully

get(*name*)
Returns a Vxlan interface as a set of key/value pairs

The Vxlan interface resource returns the following:

- name (str): The name of the interface
- type (str): Always returns 'vxlan'

- source_interface (str): The vxlan source-interface value
- multicast_group (str): The vxlan multicast-group value
- udp_port (int): The vxlan udp-port value
- vlans (dict): The vlan to vni mappings
- flood_list (list): The list of global VTEP flood list
- **multicast_decap (bool): If the mutlicast decap** feature is configured

Parameters **name** (*str*) – The interface identifier to retrieve from the running-configuration

Returns

A Python dictionary object of key/value pairs that represents the interface configuration. If the specified interface does not exist, then None is returned

remove_vlan(*name*, *vid*)
Removes a vlan to vni mapping for the interface

EosVersion: 4.13.7M

Parameters **vlan** (*str, int*) – The vlan id to map to the vni

Returns True if the command completes successfully

remove_vtep(*name*, *vtep*, *vlan=None*)
Removes a VTEP endpoint from the global or local flood list

EosVersion: 4.13.7M

Parameters

- **name** (*str*) – The name of the interface to configure
- **vtep** (*str*) – The IP address of the remote VTEP endpoint to add
- **vlan** (*str*) – The VLAN ID associated with this VTEP. If the VLAN
- **is used, then the VTEP is configured as a local flood** (*keyword*) –
- **endpoing** –

Returns True if the command completes successfully

set_multicast_decap(*name*, *default=False*, *disable=False*)
Configures the Vxlan multicast-group decap feature

EosVersion: 4.15.0M

Parameters **name** – The interface identifier to configure, defaults to Vxlan1

Returns True if the operation succeeds otherwise False

set_multicast_group(*name*, *value=None*, *default=False*, *disable=False*)
Configures the Vxlan multicast-group value

EosVersion: 4.13.7M

Parameters **name** – The interface identifier to configure, defaults to Vxlan1

Returns True if the operation succeeds otherwise False

set_source_interface(*name*, *value=None*, *default=False*, *disable=False*)
Configures the Vxlan source-interface value

EosVersion: 4.13.7M

Parameters **name** – The interface identifier to configure, defaults to Vxlan1

Returns True if the operation succeeds otherwise False

set_udp_port(*name*, *value=None*, *default=False*, *disable=False*)
Configures vxlan udp-port value

EosVersion: 4.13.7M

Parameters

- **name** (*str*) – The name of the interface to configure
- **value** (*str*) – The value to set udp-port to
- **default** (*bool*) – Configure using the default keyword
- **disable** (*bool*) – Negate the udp-port value

Returns True if the operation succeeds otherwise False

update_vlan(*name*, *vid*, *vni*)
Adds a new vlan to vni mapping for the interface

EosVersion: 4.13.7M

Parameters

- **vlan** (*str, int*) – The vlan id to map to the vni
- **vni** (*str, int*) – The vni value to use

Returns True if the command completes successfully

pyeapi.api.interfaces.**instance**(*api*)

pyeapi.api.interfaces.**isvalidinterface**(*value*)

4.2.5 Ipinterfaces

Module for working the logical IP interfaces in EOS

This module provides an API for configuring logical IP interfaces using EOS and eAPI.

param name The interface name the configuration is in reference to. The interface name is the full interface identifier

type name string

param address The interface IP address in the form of address/len.

type address string

param mtu The interface MTU value. The MTU value accepts integers in the range of 68 to 65535 bytes

type mtu integer

class pyeapi.api.ipinterfaces.**Ipinterfaces**(*node*)

Bases: *pyeapi.api.abstract.EntityCollection*

create(*name*)

Creates a new IP interface instance

This method will create a new logical IP interface for the specified physical interface. If a logical IP interface already exists then this operation will have no effect.

Note: Configuring a logical IP interface on a physical interface will remove any existing logical switchports have have been created

---

Parameters **name** (*string*) – The interface identifier to create the logical layer 3 IP interface for. The name must be the full interface name and not an abbreviated interface name (eg Ethernet1, not Et1).

Returns

True if the create operation succeeds otherwise False. If the specified interface is already created the this method will have no effect but will still return True

delete(*name*)

Deletes an IP interface instance from the running configuration

This method will delete the logical IP interface for the specified physical interface. If the interface does not have a logical IP interface defined, then this method will have no effect.

Parameters **name** (*string*) – The interface identifier to create the logical layer 3 IP interface for. The name must be the full interface name and not an abbreviated interface name (eg Ethernet1, not Et1).

Returns True if the delete operation succeeds otherwise False.

get(*name*)

Returns the specific IP interface properties

The Ipinterface resource returns the following:

- name (str): The name of the interface
- **address (str): The IP address of the interface in the form** of A.B.C.D/E
- mtu (int): The configured value for IP MTU.

Parameters **name** (*string*) – The interface identifier to retrieve the configuration for

Returns

A Python dictionary object of key/value pairs that represents the current configuration of the node. If the specified interface does not exist then None is returned.

getall()

Returns all of the IP interfaces found in the running-config

Returns

A Python dictionary object of key/value pairs keyed by interface name that represents all of the IP interfaces on the current node:

```
{
    'Ethernet1': {...},
    'Ethernet2': {...}
}
```

set_address(*name*, *value=None*, *default=False*, *disable=False*)
Configures the interface IP address

Parameters

- **name** (*string*) – The interface identifier to apply the interface config to
- **value** (*string*) – The IP address and mask to set the interface to. The value should be in the format of A.B.C.D/E
- **default** (*bool*) – Configures the address parameter to its default value using the EOS CLI default command
- **disable** (*bool*) – Negates the address parameter value using the EOS CLI no command

Returns True if the operation succeeds otherwise False.

set_mtu(*name*, *value=None*, *default=False*, *disable=False*)
Configures the interface IP MTU

Parameters

- **name** (*string*) – The interface identifier to apply the interface config to
- **value** (*integer*) – The MTU value to set the interface to. Accepted values include 68 to 65535
- **default** (*bool*) – Configures the mtu parameter to its default value using the EOS CLI default command
- **disable** (*bool*) – CLI no command

Returns True if the operation succeeds otherwise False.

Raises `ValueError` – If the value for MTU is not an integer value or outside of the allowable range

pyeapi.api.ipinterfaces.**instance**(*node*)
Returns an instance of Ipinterfaces

This method will create and return an instance of the Ipinterfaces object passing the value of node to the instance. This function is required for the resource to be autoloaded by the Node object

Parameters **node** (Node) – The node argument provides an instance of Node to the Ipinterfaces instance

4.2.6 Mlag

Module for working with EOS MLAG resources

The Mlag resource provides configuration management of global and interface MLAG settings on an EOS node.

param config The global MLAG configuration values

type config dict

param interfaces The configured MLAG interfaces

type interfaces dict

Config Parameters:

domain_id (str): The domain_id parameter is parsed from the nodes mlag configuration. The domain id is an alphanumeric string that names the MLAG domain

local_interface (str): The local VLAN interface used as the control plane endpoint between MLAG peers. Valid values include any VLAN SVI

peer_address (str): The IP address of the MLAG peer used to send MLAG control traffic. The peer address must be reachable from the local interface. Valid values include any IPv4 unicast address

peer_link (str): The physical link that connects the node to its MLAG peer. Valid values for the peer link include layer 2 Ethernet or Port-Channel interfaces

shutdown (bool): The administrative state of the global MLAG process.

Interface Parameters:

mlag_id (str): The interface mlag parameter parsed from the nodes interface configuration. Valid values for the mlag id are in the range of 1 to 2000

class pyeapi.api.mlag.**Mlag**(*node*)
Bases: *pyeapi.api.abstract.Entity*

The Mlag class provides management of the MLAG configuration

The Mlag class is derived from Entity and provides an API for working with the nodes MLAG configuraiton.

get()
Returns the Mlag configuration as a resource dict

Returns A dict ojbect containing the Mlag resource attributes.

Return type dict

set_domain_id(*value=None*, *default=False*, *disable=False*)
Configures the mlag domain-id value

Parameters

- **value** (*str*) – The value to configure the domain-id
- **default** (*bool*) – Configures the domain-id using the default keyword
- **disable** (*bool*) – Negates the domain-id using the no keyword

Returns Returns True if the commands complete successfully

Return type bool

set_local_interface(*value=None*, *default=False*, *disable=False*)
Configures the mlag local-interface value

Parameters

- **value** (*str*) – The value to configure the local-interface
- **default** (*bool*) – Configures the local-interface using the default keyword
- **disable** (*bool*) – Negates the local-interface using the no keyword

Returns Returns True if the commands complete successfully

Return type bool

set_mlag_id(*name*, *value=None*, *default=False*, *disable=False*)
Configures the interface mlag value for the specified interface

Parameters

- **name** (*str*) – The interface to configure. Valid values for the name arg include Port-Channel*
- **value** (*str*) – The mlag identifier to cofigure on the interface
- **default** (*bool*) – Configures the interface mlag value using the default keyword
- **disable** (*bool*) – Negates the interface mlag value using the no keyword

Returns Returns True if the commands complete successfully

Return type bool

set_peer_address(*value=None*, *default=False*, *disable=False*)
Configures the mlag peer-address value

Parameters

- **value** (*str*) – The value to configure the peer-address
- **default** (*bool*) – Configures the peer-address using the default keyword
- **disable** (*bool*) – Negates the peer-address using the no keyword

Returns Returns True if the commands complete successfully

Return type bool

set_peer_link(*value=None*, *default=False*, *disable=False*)
Configures the mlag peer-link value

Parameters

- **value** (*str*) – The value to configure the peer-link
- **default** (*bool*) – Configures the peer-link using the default keyword
- **disable** (*bool*) – Negates the peer-link using the no keyword

Returns Returns True if the commands complete successfully

Return type bool

set_shutdown(*default=False*, *disable=True*)
Configures the mlag shutdown value

Default setting for set_shutdown is disable=True, meaning 'no shutdown'. Setting both default and disable to False will effectively enable shutdown.

Parameters

- **default** (*bool*) – Configures the shutdown using the default keyword
- **disable** (*bool*) – Negates shutdown using the no keyword

Returns Returns True if the commands complete successfully

Return type bool

pyeapi.api.mlag.**instance**(*node*)
Returns an instance of Mlag

Parameters **node** (Node) – The node argument passes an instance of Node to the resource

Returns An instance of Mlag

Return type object

4.2.7 Ntp

Module for managing the NTP configuration in EOS

This module provides an API for configuring NTP resources using EOS and eAPI.

param name The interface port that specifies the NTP source.

type name string

class pyeapi.api.ntp.**Ntp**(**args*, ***kwargs*)
Bases: *pyeapi.api.abstract.Entity*

The Ntp class implements global NTP router configuration

add_server(*name*, *prefer=False*)
Add or update an NTP server entry to the node config

Parameters

- **name** (*string*) – The IP address or FQDN of the NTP server.
- **prefer** (*bool*) – Sets the NTP server entry as preferred if True.

Returns True if the operation succeeds, otherwise False.

create(*name*)
Instantiate the NTP by setting the source interface.

Parameters **name** (*string*) – The interface port that specifies the NTP source.

Returns True if the operation succeeds, otherwise False.

default()
Default the NTP source entry from the node.

Returns True if the operation succeeds, otherwise False.

delete()
Delete the NTP source entry from the node.

Returns True if the operation succeeds, otherwise False.

get()
Returns the current NTP configuration

The Ntp resource returns the following:

- **source_interface (str): The interface port that specifies** NTP server
- **servers (list): A list of the NTP servers that have been** assigned to the node. Each entry in the list is a key/value pair of the name of the server as the key and None or 'prefer' as the value if the server is preferred.

Returns

A Python dictionary object of key/value pairs that represents the current NTP configuration of the node:

```
{
    "source_interface": 'Loopback0',
    'servers': [
        { '1.1.1.1': None },
        { '1.1.1.2': 'prefer' },
        { '1.1.1.3': 'prefer' },
```

```
        { '1.1.1.4': None },
    ]
}
```

remove_all_servers()
Remove all NTP server entries from the node config

Returns True if the operation succeeds, otherwise False.

remove_server(*name*)
Remove an NTP server entry from the node config

Parameters **name** (*string*) – The IP address or FQDN of the NTP server.

Returns True if the operation succeeds, otherwise False.

set_source_interface(*name*)
Assign the NTP source on the node

Parameters **name** (*string*) – The interface port that specifies the NTP source.

Returns True if the operation succeeds, otherwise False.

pyeapi.api.ntp.**instance**(*node*)
Returns an instance of Ntp

Parameters **node** (Node) – The node argument passes an instance of Node to the resource

Returns An instance of Ntp

Return type object

4.2.8 Ospf

Module for working with OSPF configuration in EOS

This module provides an API for creating/modifying/deleting OSPF configurations

class pyeapi.api.ospf.**Ospf**(**args*, ***kwargs*)
Bases: *pyeapi.api.abstract.Entity*

The Ospf class implements global Ospf router configuration

add_network(*network*, *netmask*, *area=0*)
Adds a network to be advertised by OSPF

Parameters

- **network** (*str*) – The network to be advertised in dotted decimal notation
- **netmask** (*str*) – The netmask to configure
- **area** (*str*) – The area the network belongs to. By default this value is 0

Returns True if the command completes successfully

Return type bool

Exception:

ValueError: This will get raised if network or netmask are not passed to the method

add_redistribution(*protocol*, *route_map_name=None*)
Adds a protocol redistribution to OSPF

Parameters

- **protocol** (*str*) – protocol to redistribute
- **route_map_name** (*str*) – route-map to be used to filter the protocols

Returns True if the command completes successfully

Return type bool

Exception:

ValueError: This will be raised if the protocol pass is not one of the following: [rip, bgp, static, connected]

configure_ospf(*cmd*)
Allows for a list of OSPF subcommands to be configured"

Parameters **cmd** – (list or str): Subcommand to be entered

Returns True if all the commands completed successfully

Return type bool

create(*ospf_process_id*, *vrf=None*)
Creates a OSPF process in the specified VRF or the default VRF.

Parameters

- **ospf_process_id** (*str*) – The OSPF process Id value
- **vrf** (*str*) – The VRF to apply this OSPF process to

Returns True if the command completed successfully

Return type bool

Exception:

ValueError: If the ospf_process_id passed in less than 0 or greater than 65536

delete()
Removes the entire ospf process from the running configuration

Parameters **None** –

Returns True if the command completed succssfully

Return type bool

get(*vrf=None*)
Returns the OSPF routing configuration

Parameters **vrf** (*str*) – VRF name to return OSPF routing config for

Returns

keys: router_id (int): OSPF router-id vrf (str): VRF of the OSPF process networks (dict): All networks that

are advertised in OSPF

ospf_process_id (int): OSPF proc id redistribution (dict): All protocols that

are configured to be redistributed in OSPF

shutdown (bool): Gives the current shutdown off the process

Return type dict

remove_network(*network*, *netmask*, *area=0*)
Removes a network advertisment by OSPF

Parameters

- **network** (*str*) – The network to be removed in dotted decimal notation
- **netmask** (*str*) – The netmask to configure
- **area** (*str*) – The area the network belongs to. By default this value is 0

Returns True if the command completes successfully

Return type bool

Exception:

ValueError: This will get raised if network or netmask are not passed to the method

remove_redistribution(*protocol*)
Removes a protocol redistribution to OSPF

Parameters

- **protocol** (*str*) – protocol to redistribute
- **route_map_name** (*str*) – route-map to be used to filter the protocols

Returns True if the command completes successfully

Return type bool

Exception:

ValueError: This will be raised if the protocol pass is not one of the following: [rip, bgp, static, connected]

set_no_shutdown()
Removes the shutdown property from the OSPF process

Parameters **None** –

Returns True if the commands are completed successfully

Return type bool

set_router_id(*value=None*, *default=False*, *disable=False*)
Controls the router id property for the OSPF Proccess

Parameters

- **value** (*str*) – The router-id value
- **default** (*bool*) – Controls the use of the default keyword
- **disable** (*bool*) – Controls the use of the no keyword

Returns True if the commands are completed successfully

Return type bool

set_shutdown()
Shutdowns the OSPF process

Parameters **None** –

Returns True if the commands are completed successfully

Return type bool

pyeapi.api.ospf.**instance**(*api*)
Returns an instance of Ospf

4.2.9 Routemaps

Module for working with EOS routemap resources

The Routemap resource provides configuration management of global route-map resources on an EOS node. It provides the following class implementations:

- Routemaps - Configures routemaps in EOS

Notes

The set and match attributes produce a list of strings with The corresponding configuration. These strings will omit the preceeding set or match words, respectively.

class pyeapi.api.routemaps.**Routemaps**(*node*)
Bases: *pyeapi.api.abstract.EntityCollection*

The Routemaps class provides management of the routemaps configuration

The Routemaps class is derived from Entity and provides an API for working with the nodes routemaps configuraiton.

create(*name*, *action*, *seqno*)
Creates a new routemap on the node

Note: This method will attempt to create the routemap regardless if the routemap exists or not. If the routemap already exists then this method will still return True.

Parameters

- **name** (*string*) – The full name of the routemap.
- **action** (*string*) – The action to take for this routemap clause.
- **seqno** (*integer*) – The sequence number for the routemap clause.

Returns True if the routemap could be created otherwise False (see Note)

default(*name*, *action*, *seqno*)
Defaults the routemap on the node

Note: This method will attempt to default the routemap from the nodes operational config. Since routemaps do not exist by default, the default action is essentially a negation and the result will be the

removal of the routemap clause. If the routemap does not exist then this method will not perform any changes but still return True

Parameters

- **name** (*string*) – The full name of the routemap.
- **action** (*string*) – The action to take for this routemap clause.
- **seqno** (*integer*) – The sequence number for the routemap clause.

Returns True if the routemap could be deleted otherwise False (see Node)

delete (*name*, *action*, *seqno*)

Deletes the routemap from the node

Note: This method will attempt to delete the routemap from the nodes operational config. If the routemap does not exist then this method will not perform any changes but still return True

Parameters

- **name** (*string*) – The full name of the routemap.
- **action** (*string*) – The action to take for this routemap clause.
- **seqno** (*integer*) – The sequence number for the routemap clause.

Returns True if the routemap could be deleted otherwise False (see Node)

get (*name*)

Provides a method to retrieve all routemap configuration related to the name attribute.

Parameters **name** (*string*) – The name of the routemap.

Returns

None if the specified routemap does not exists. If the routermap exists a dictionary will be provided as follows:

```
{
    'deny': {
            30: {
                    'continue': 200,
                    'description': None,
                    'match': ['as 2000',
                              'source-protocol ospf',
                              'interface Ethernet2'],
                    'set': []
                }
            },
    'permit': {
            10: {
                    'continue': 100,
                    'description': None,
                    'match': ['interface Ethernet1'],
                    'set': ['tag 50']},
            20: {
                    'continue': 200,
                    'description': None,
```

```
                    'match': ['as 2000',
                              'source-protocol ospf',
                              'interface Ethernet2'],
                    'set': []
                }
            }
}
```

getall()

set_continue(*name*, *action*, *seqno*, *value=None*, *default=False*, *disable=False*)
Configures the routemap continue value

Parameters

- **name** (*string*) – The full name of the routemap.
- **action** (*string*) – The action to take for this routemap clause.
- **seqno** (*integer*) – The sequence number for the routemap clause.
- **value** (*integer*) – The value to configure for the routemap continue
- **default** (*bool*) – Specifies to default the routemap continue value
- **disable** (*bool*) – Specifies to negate the routemap continue value

Returns True if the operation succeeds otherwise False is returned

set_description(*name*, *action*, *seqno*, *value=None*, *default=False*, *disable=False*)
Configures the routemap description

Parameters

- **name** (*string*) – The full name of the routemap.
- **action** (*string*) – The action to take for this routemap clause.
- **seqno** (*integer*) – The sequence number for the routemap clause.
- **value** (*string*) – The value to configure for the routemap description
- **default** (*bool*) – Specifies to default the routemap description value
- **disable** (*bool*) – Specifies to negate the routemap description

Returns True if the operation succeeds otherwise False is returned

set_match_statements(*name*, *action*, *seqno*, *statements*)
Configures the match statements within the routemap clause. The final configuration of match statements will reflect the list of statements passed into the statements attribute. This implies match statements found in the routemap that are not specified in the statements attribute will be removed.

Parameters

- **name** (*string*) – The full name of the routemap.
- **action** (*string*) – The action to take for this routemap clause.
- **seqno** (*integer*) – The sequence number for the routemap clause.
- **statements** (*list*) – A list of the match-related statements. Note that the statements should omit the leading match.

Returns True if the operation succeeds otherwise False

set_set_statements(*name*, *action*, *seqno*, *statements*)
Configures the set statements within the routemap clause. The final configuration of set statements will reflect the list of statements passed into the statements attribute. This implies set statements found in the routemap that are not specified in the statements attribute will be removed.

Parameters

- **name** (*string*) – The full name of the routemap.
- **action** (*string*) – The action to take for this routemap clause.
- **seqno** (*integer*) – The sequence number for the routemap clause.
- **statements** (*list*) – A list of the set-related statements. Note that the statements should omit the leading set.

Returns True if the operation succeeds otherwise False

pyeapi.api.routemaps.**instance**(*node*)
Returns an instance of Routemaps

Parameters **node** (Node) – The node argument passes an instance of Node to the resource

Returns An instance of Routemaps

Return type object

4.2.10 Spanningtree

4.2.11 Staticroute

Module for working with EOS static routes

The staticroute resource provides configuration management of static route resources on an EOS node. It provides the following class implementations:

- StaticRoute - Configure static routes in EOS

StaticRoute Attributes:

ip_dest (string): The ip address of the destination in the form of A.B.C.D/E

next_hop (string): The next hop interface or ip address next_hop_ip (string): The next hop address on destination interface distance (int): Administrative distance for this route tag (int): Route tag route_name (string): Route name

Notes

The 'default' prefix function of the 'ip route' command, 'default ip route . . . ', currently equivalent to the 'no ip route . . . ' command.

class pyeapi.api.staticroute.**StaticRoute**(*node*)
Bases: *pyeapi.api.abstract.EntityCollection*

The StaticRoute class provides a configuration instance for working with static routes

create(*ip_dest*, *next_hop*, ***kwargs*)
Create a static route

Parameters

- **ip_dest** (*string*) – The ip address of the destination in the form of A.B.C.D/E

- **next_hop** (*string*) – The next hop interface or ip address
- ****kwargs['next_hop_ip']** (*string*) – The next hop address on destination interface
- ****kwargs['distance']** (*string*) – Administrative distance for this route
- ****kwargs['tag']** (*string*) – Route tag
- ****kwargs['route_name']** (*string*) – Route name

Returns True if the operation succeeds, otherwise False.

default(*ip_dest*, *next_hop*, ***kwargs*)
Set a static route to default (i.e. delete the matching route)

Parameters

- **ip_dest** (*string*) – The ip address of the destination in the form of A.B.C.D/E
- **next_hop** (*string*) – The next hop interface or ip address
- ****kwargs['next_hop_ip']** (*string*) – The next hop address on destination interface
- ****kwargs['distance']** (*string*) – Administrative distance for this route
- ****kwargs['tag']** (*string*) – Route tag
- ****kwargs['route_name']** (*string*) – Route name

Returns True if the operation succeeds, otherwise False.

delete(*ip_dest*, *next_hop*, ***kwargs*)
Delete a static route

Parameters

- **ip_dest** (*string*) – The ip address of the destination in the form of A.B.C.D/E
- **next_hop** (*string*) – The next hop interface or ip address
- ****kwargs['next_hop_ip']** (*string*) – The next hop address on destination interface
- ****kwargs['distance']** (*string*) – Administrative distance for this route
- ****kwargs['tag']** (*string*) – Route tag
- ****kwargs['route_name']** (*string*) – Route name

Returns True if the operation succeeds, otherwise False.

get(*name*)
Retrieves the ip route information for the destination ip address specified.

Parameters **name** (*string*) – The ip address of the destination in the form of A.B.C.D/E

Returns

An dict object of static route entries in the form:

```
{ ip_dest:
    { next_hop:
        { next_hop_ip:
            { distance:
                { 'tag': tag,
                  'route_name': route_name
```

```
                    }
                }
            }
        }
}
```

If the ip address specified does not have any associated static routes, then None is returned.

Return type dict

Notes

The keys ip_dest, next_hop, next_hop_ip, and distance in the returned dictionary are the values of those components of the ip route specification. If a route does not contain a next_hop_ip, then that key value will be set as 'None'.

getall()
Return all ip routes configured on the switch as a resource dict

Returns

An dict object of static route entries in the form:

```
{ ip_dest:
    { next_hop:
        { next_hop_ip:
            { distance:
                { 'tag': tag,
                  'route_name': route_name
                }
            }
        }
    }
}
```

If the ip address specified does not have any associated static routes, then None is returned.

Return type dict

Notes

The keys ip_dest, next_hop, next_hop_ip, and distance in the returned dictionary are the values of those components of the ip route specification. If a route does not contain a next_hop_ip, then that key value will be set as 'None'.

set_route_name(*ip_dest*, *next_hop*, ***kwargs*)
Set the route_name value for the specified route

Parameters

- **ip_dest** (*string*) – The ip address of the destination in the form of A.B.C.D/E
- **next_hop** (*string*) – The next hop interface or ip address
- ****kwargs['next_hop_ip']** (*string*) – The next hop address on destination interface
- ****kwargs['distance']** (*string*) – Administrative distance for this route

- ****kwargs['tag']** (*string*) – Route tag
- ****kwargs['route_name']** (*string*) – Route name

Returns True if the operation succeeds, otherwise False.

Notes

Any existing tag value must be included in call to set_route_name, otherwise the tag will be reset by the call to EOS.

set_tag(*ip_dest*, *next_hop*, ***kwargs*)
Set the tag value for the specified route

Parameters

- **ip_dest** (*string*) – The ip address of the destination in the form of A.B.C.D/E
- **next_hop** (*string*) – The next hop interface or ip address
- ****kwargs['next_hop_ip']** (*string*) – The next hop address on destination interface
- ****kwargs['distance']** (*string*) – Administrative distance for this route
- ****kwargs['tag']** (*string*) – Route tag
- ****kwargs['route_name']** (*string*) – Route name

Returns True if the operation succeeds, otherwise False.

Notes

Any existing route_name value must be included in call to set_tag, otherwise the tag will be reset by the call to EOS.

pyeapi.api.staticroute.**instance**(*node*)
Returns an instance of StaticRoute

This method will create and return an instance of the StaticRoute object passing the value of API to the object. The instance method is required for the resource to be autoloaded by the Node object

Parameters **node** (Node) – The node argument passes an instance of Node to the resource

4.2.12 Stp

Module for working with spanning-tree in EOS

This module provides an API for working with spanning-tree configuration in EOS. This includes both global spanning-tree configuration as well as interface config.

Global Parameters:

mode (string): The spanning-tree operational mode. Accepted values for this version are 'mstp' or 'none'. This configuration parameter is not defaultable

interfaces (StpInterfaces): The collection of STP enabled interfaces.

Interface Parameters:

name (string): The name of the interface the STP configuration is in reference to. The interface name is the full interface identifier

portfast (string): The portfast configuration value for the interface. Accepted values are 'edge', 'network', or 'disabled'

bpduguard (boolean): True if the BPDU Guard feature is enabled on the interface or False if it is disabled

class pyeapi.api.stp.**Stp**(**args*, ***kwargs*)
Bases: *pyeapi.api.abstract.Entity*

The Stp class implements global configuration for spanning-tree

The spanning-tree protocol provides both global and interface configuration options. This class is the top-level class that provides access to all spanning-tree configuration options supported.

Example

The below example demonstrates how to use the STP class to work with both global configuration and interface configuration.

```
>>> import pyeapi.resources.stp
>>> stp = pyeapi.resources.stp.instance(node)
>>> stp.set_mode('mstp')
True
>>> stp.interfaces.set_bpduguard('Ethernet1', True)
True
```

interfaces
StpInterfaces – An instance for configuration spanning-tree configuration interfaces

instances
StpInstances – An instance object for working with spanning-tree global instances

get()
Returns the spanning-tree configuration as a dict object

The dictionary object represents the entire spanning-tree configuration derived from the nodes running config. This includes both globally configuration attributes as well as interfaces and instances. See the StpInterfaces and StpInstances classes for the key/value pair definitions.

Note: See the individual classes for detailed message structures

Returns

A Python dictionary object of key/value pairs the represent the entire supported spanning-tree configuration:

```
{
    "mode": [mstp, none],
    "interfaces": {...},
    "instances": {...}
}
```

instances

interfaces

set_mode(*value=None*, *default=False*, *disable=False*)
Configures the global spanning-tree mode

Note: This configuration parameter is not defaultable

Parameters

- **value** (*string*) – The value to configure the global spanning-tree mode of operation. Valid values include 'mstp', 'none'
- **default** (*bool*) – Set the global spanning-tree mode to default.
- **disable** (*bool*) – Negate the global spanning-tree mode.

Returns True if the configuration operation succeeds otherwise False

Raises ValueError if the value is not in the accepted range

class pyeapi.api.stp.**StpInstances**(*node*)
Bases: *pyeapi.api.abstract.EntityCollection*

Provides a configuration resource for spanning-tree instances

This class provides an API for working with spanning-tree instances from the global configuration. Spanning tree instances work with MST configuration

getall()

class pyeapi.api.stp.**StpInterfaces**(*node*)
Bases: *pyeapi.api.abstract.EntityCollection*

Provides a configuration resource for spanning-tree interfaces

This class provides an API for working with spanning-tree interface configurations. It provides access to managing specific interface spanning-tree configuration options. Note that spanning-tree interfaces cannot be created or deleted.

configure_interface(*name*, *cmds*)

get(*name*)
Returns the specified interfaces STP configuration resource

The STP interface resource contains the following

- name (str): The interface name
- portfast (bool): The spanning-tree portfast admin state
- bpduguard (bool): The spanning-tree bpduguard admin state
- **portfast_type (str): The spanning-tree portfast <type> value.** Valid values include "edge", "network", "normal"

Parameters **name** (*string*) – The interface identifier to retrieve the config for. Note: Spanning-tree interfaces are only supported on Ethernet and Port-Channel interfaces

Returns

A resource dict object that represents the interface configuration.

None: If the specified interace is not a STP port

Return type dict

getall()
Returns the collection of STP interfaces

This method will return all of the configured spanning-tree interfaces from the current nodes configuration.

Returns

A Python dictionary object that represents all configured spanning-tree interfaces indexed by interface name.

set_bpduguard(*name*, *value=False*, *default=False*, *disable=False*)
Configures the bpduguard value for the specified interface

Parameters

- **name** (*string*) – The interface identifier to configure. The name must be the full interface name (eg Ethernet1, not Et1)
- **value** (*bool*) – True if bpduguard is enabled otherwise False
- **default** (*bool*) – Configures the bpduguard parameter to its default value using the EOS CLI default config command
- **disable** (*bool*) – Negates the bpduguard parameter using the EOS CLI no config command

Returns True if the command succeeds, otherwise False

Raises

- `ValueError` – Rasied if an invalid interface name is specified
- `TypeError` – Raised if the value keyword argument does not evaluate to a valid boolean

set_portfast(*name*, *value=None*, *default=False*, *disable=False*)
Configures the portfast value for the specified interface

Parameters

- **name** (*string*) – The interface identifier to configure. The name must be the full interface name (eg Ethernet1, not Et1)
- **value** (*bool*) – True if portfast is enabled otherwise False
- **default** (*bool*) – Configures the portfast parameter to its default value using the EOS CLI default config command
- **disable** (*bool*) – Negates the portfast parameter using the EOS CLI no config command

Returns True if the command succeeds, otherwise False

Raises

- `ValueError` – Rasied if an invalid interface name is specified
- `TypeError` – Raised if the value keyword argument does not evaluate to a valid boolean

set_portfast_type(*name*, *value='normal'*)
Configures the portfast value for the specified interface

Parameters

- **name** (*string*) – The interface identifier to configure. The name must be the full interface name (eg Ethernet1, not Et1).

- **value** (*string*) – The value to configure the portfast setting to. Valid values include "edge", "network", "normal". The default value is "normal"

Returns True if the command succeeds, otherwise False

Raises ValueError – Raised if an invalid interface name or value is specified

pyeapi.api.stp.**instance**(*api*)

pyeapi.api.stp.**isvalidinterface**(*value*)

Checks value to see if it could be a spanning-tree interface

This function will check the value and return a boolean whether or not the interface could be a spanning-tree interface

---

Note:

This function only checks *if* the interface could be a spanning-tree interface but does not check if it *is* configured as a spanning-tree interface

---

Parameters **value** (*string*) – The interface name to validate

Returns True if it could be a spanning-tree interface, otherwise False

4.2.13 Switchports

Module for working with logical layer 2 switchports in EOS

This module provides an API for working with logical layer 2 interfaces (switchports) in EOS. Switchports are interfaces built on top of physical Ethernet and bundled Port-Channel interfaces.

class pyeapi.api.switchports.**Switchports**(*node*)

Bases: *pyeapi.api.abstract.EntityCollection*

The Switchports class provides a configuration resource for swichports

Logical layer 2 interfaces built on top of physical Ethernet and bundled Port-Channel interfaces can be configured and managed with an instance of Switchports. The Switchports class is a resource collection and supports get and getall methods. The Switchports class is derived from the BaseResource class

add_trunk_group(*intf*, *value*)

Adds the specified trunk group to the interface

Parameters

- **intf** (*str*) – The interface name to apply the trunk group to
- **value** (*str*) – The trunk group value to apply to the interface

Returns True if the operation as successfully applied otherwise false

create(*name*)

Creates a new logical layer 2 interface

This method will create a new switchport for the interface specified in the arguments (name). If the logical switchport already exists then this command will have no effect

Parameters **name** (*string*) – The interface identifier to create the logical layer 2 switchport for. The name must be the full interface name and not an abbreviated interface name (eg Ethernet1, not Et1)

Returns

True if the create operation succeeds otherwise False. If the interface specified in args is already a switchport then this method will have no effect but will still return True

default (*name*)

Defaults the configuration of the switchport interface

This method will default the configuration state of the logical layer 2 interface.

Parameters **name** (*string*) – The interface identifier to create the logical layer 2 switchport for. The name must be the full interface name and not an abbreviated interface name (eg Ethernet1, not Et1)

Returns

True if the create operation succeeds otherwise False. If the interface specified in args is already a switchport then this method will have no effect but will still return True

delete (*name*)

Deletes the logical layer 2 interface

This method will delete the logical switchport for the interface specified in the arguments. If the interface doe not have a logical layer 2 interface defined, then this method will have no effect.

Parameters **name** (*string*) – The interface identifier to create the logical layer 2 switchport for. The name must be the full interface name and not an abbreviated interface name (eg Ethernet1, not Et1)

Returns

True if the create operation succeeds otherwise False. If the interface specified in args is already a switchport then this method will have no effect but will still return True

get (*name*)

Returns a dictionary object that represents a switchport

The Switchport resource returns the following:

- name (str): The name of the interface
- mode (str): The switchport mode value
- access_vlan (str): The switchport access vlan value
- trunk_native_vlan (str): The switchport trunk native vlan vlaue
- trunk_allowed_vlans (str): The trunk allowed vlans value
- trunk_groups (list): The list of trunk groups configured

Parameters **name** (*string*) – The interface identifier to get. Note: Switchports are only supported on Ethernet and Port-Channel interfaces

Returns

A Python dictionary object of key/value pairs that represent the switchport configuration for the interface specified If the specified argument is not a switchport then None is returned

Return type dict

getall ()

Returns a dict object to all Switchports

This method will return all of the configured switchports as a dictionary object keyed by the interface identifier.

Returns

A Python dictionary object that represents all configured switchports in the current running configuration

remove_trunk_group(*intf*, *value*)
Removes a specified trunk group to the interface

Parameters

- **intf** (*str*) – The interface name to remove the trunk group from
- **value** (*str*) – The trunk group value

Returns True if the operation as successfully applied otherwise false

set_access_vlan(*name*, *value=None*, *default=False*, *disable=False*)
Configures the switchport access vlan

Parameters

- **name** (*string*) – The interface identifier to create the logical layer 2 switchport for. The name must be the full interface name and not an abbreviated interface name (eg Ethernet1, not Et1)
- **value** (*string*) – The value to set the access vlan to. The value must be a valid VLAN ID in the range of 1 to 4094.
- **default** (*bool*) – Configures the access vlan parameter to its default value using the EOS CLI
- **disable** (*bool*) – Negate the access vlan parameter using the EOS CLI

Returns True if the create operation succeeds otherwise False.

set_mode(*name*, *value=None*, *default=False*, *disable=False*)
Configures the switchport mode

Parameters

- **name** (*string*) – The interface identifier to create the logical layer 2 switchport for. The name must be the full interface name and not an abbreviated interface name (eg Ethernet1, not Et1)
- **value** (*string*) – The value to set the mode to. Accepted values for this argument are access or trunk
- **default** (*bool*) – Configures the mode parameter to its default value using the EOS CLI
- **disable** (*bool*) – Negate the mode parameter using the EOS CLI

Returns True if the create operation succeeds otherwise False.

set_trunk_allowed_vlans(*name*, *value=None*, *default=False*, *disable=False*)
Configures the switchport trunk allowed vlans value

Parameters

- **name** (*string*) – The interface identifier to create the logical layer 2 switchport for. The name must be the full interface name and not an abbreviated interface name (eg Ethernet1, not Et1)
- **value** (*string*) – The value to set the trunk allowed vlans to. The value must be a valid VLAN ID in the range of 1 to 4094.

- **default** (*bool*) – Configures the access vlan parameter to its default value using the EOS CLI
- **disable** (*bool*) – Negate the access vlan parameter using the EOS CLI

Returns True if the create operation succeeds otherwise False.

set_trunk_groups(*intf*, *value=None*, *default=False*, *disable=False*)
Configures the switchport trunk group value

Parameters

- **intf** (*str*) – The interface identifier to configure.
- **value** (*str*) – The set of values to configure the trunk group
- **default** (*bool*) – Configures the trunk group default value
- **disable** (*bool*) – Negates all trunk group settings

Returns True if the config operation succeeds otherwise False

set_trunk_native_vlan(*name*, *value=None*, *default=False*, *disable=False*)
Configures the switchport trunk native vlan value

Parameters

- **name** (*string*) – The interface identifier to create the logical layer 2 switchport for. The name must be the full interface name and not an abbreviated interface name (eg Ethernet1, not Et1)
- **value** (*string*) – The value to set the trunk nativevlan to. The value must be a valid VLAN ID in the range of 1 to 4094.
- **default** (*bool*) – Configures the access vlan parameter to its default value using the EOS CLI
- **disable** (*bool*) – Negate the access vlan parameter using the EOS CLI

Returns True if the create operation succeeds otherwise False.

pyeapi.api.switchports.**instance**(*node*)
Returns an instance of Switchports

This method will create and return an instance of the Switchports object passing the value of node to the instance. The module method is required for the resource to be autoloaded by the Node object

Parameters **node** (Node) – The node argument provides an instance of Node to the resource

4.2.14 System

Module for working with the global system in EOS

This module provides an API for working with the global system settings in EOS. It provides the following class implementations:

- System – Configures global system settings

System Attributes:

hostname (string): The hostname of the node as configured in the running-configuration.

class pyeapi.api.system.**System**(*node*)
Bases: *pyeapi.api.abstract.Entity*

The System class implements global config for the node

Global configuration settings include those thaat identify the node and provide node level configuration such as hostname

get()
Returns the system configuration abstraction

The System resource returns the following:

- hostname (str): The hostname value

Returns Represents the node's system configuration

Return type dict

set_banner(*banner_type*, *value=None*, *default=False*, *disable=False*)
Configures system banners

Parameters

- **banner_type** (*str*) – banner to be changed (likely login or motd)
- **value** (*str*) – value to set for the banner
- **default** (*bool*) – Controls the use of the default keyword
- **disable** (*bool*) – Controls the use of the no keyword‘

Returns True if the commands completed successfully otherwise False

Return type bool

set_hostname(*value=None*, *default=False*, *disable=False*)
Configures the global system hostname setting

EosVersion: 4.13.7M

Parameters

- **value** (*str*) – The hostname value
- **default** (*bool*) – Controls use of the default keyword
- **disable** (*bool*) – Controls the use of the no keyword

Returns True if the commands are completed successfully

Return type bool

set_iprouting(*value=None*, *default=False*, *disable=False*)
Configures the state of global ip routing

EosVersion: 4.13.7M

Parameters

- **value** (*bool*) – True if ip routing should be enabled or False if ip routing should be disabled
- **default** (*bool*) – Controls the use of the default keyword
- **disable** (*bool*) – Controls the use of the no keyword

Returns True if the commands completed successfully otherwise False

Return type bool

pyeapi.api.system.**instance**(*api*)
Returns an instance of System

4.2.15 Users

API Module for working with EOS local user resources

The Users resource provides configuration of local user resources for an EOS node.

param username The username parameter maps to the local username defined in the running-config.

type username string

param nopassword Configures the username to use no password at login. This parameter is mutually exclusive with secret

type nopassword boolean

param privilege Configures the user privilege level in EOS

type privilege integer

param role Configures the users role in EOS

type role string

param secret Configures the users secret (password) to use at login. This parameter is mutually exclusive with secret and is used in conjunction with format.

type secret string

param format Configures the format of the secret value. Accepted values for format are "cleartext", "md5" and "sha512"

type format string

class pyeapi.api.users.**Users**(*node*)
Bases: *pyeapi.api.abstract.EntityCollection*

The Users class provides a configuration resource for local users. The regex used here parses the running configuration to find username entries. There is extra logic in the regular expression to store the username as 'user' and then creates a backreference to find a following configuration line that might contain the users sshkey.

create(*name*, *nopassword=None*, *secret=None*, *encryption=None*)
Creates a new user on the local system.

Creating users requires either a secret (password) or the nopassword keyword to be specified.

Parameters

- **name** (*str*) – The name of the user to craete
- **nopassword** (*bool*) – Configures the user to be able to authenticate without a password challenage
- **secret** (*str*) – The secret (password) to assign to this user
- **encryption** (*str*) – Specifies how the secret is encoded. Valid values are "cleartext", "md5", "sha512". The default is "cleartext"

Returns True if the operation was successful otherwise False

Raises TypeError – if the required arguments are not satisfied

create_with_nopassword(*name*)
Creates a new user on the local node

Parameters **name** (*str*) – The name of the user to create

Returns True if the operation was successful otherwise False

create_with_secret(*name*, *secret*, *encryption*)
Creates a new user on the local node

Parameters

- **name** (*str*) – The name of the user to craete
- **secret** (*str*) – The secret (password) to assign to this user
- **encryption** (*str*) – Specifies how the secret is encoded. Valid values are "cleartext", "md5", "sha512". The default is "cleartext"

Returns True if the operation was successful otherwise False

default(*name*)
Configures the local username using the default keyword

Parameters **name** (*str*) – The name of the user to configure

Returns True if the operation was successful otherwise False

delete(*name*)
Deletes the local username from the config

Parameters **name** (*str*) – The name of the user to delete

Returns True if the operation was successful otherwise False

get(*name*)
Returns the local user configuration as a resource dict

Parameters **name** (*str*) – The username to return from the nodes global running- config.

Returns

A resource dict object

If the *name* does not exist, then None is returned

Return type dict

getall()
Returns all local users configuration as a resource dict

Returns A dict of usernames with a nested resource dict object

Return type dict

set_privilege(*name*, *value=None*)
Configures the user privilege value in EOS

Parameters

- **name** (*str*) – The name of the user to craete
- **value** (*int*) – The privilege value to assign to the user. Valid values are in the range of 0 to 15

Returns True if the operation was successful otherwise False

Raises `TypeError` – if the value is not in the valid range

set_role(*name*, *value=None*, *default=False*, *disable=False*)
Configures the user role vale in EOS

Parameters

- **name** (*str*) – The name of the user to create
- **value** (*str*) – The value to configure for the user role
- **default** (*bool*) – Configure the user role using the EOS CLI default command
- **disable** (*bool*) – Negate the user role using the EOS CLI no command

Returns True if the operation was successful otherwise False

set_sshkey(*name*, *value=None*, *default=False*, *disable=False*)
Configures the user sshkey

Parameters

- **name** (*str*) – The name of the user to add the sshkey to
- **value** (*str*) – The value to configure for the sshkey.
- **default** (*bool*) – Configure the sshkey using the EOS CLI default command
- **disable** (*bool*) – Negate the sshkey using the EOS CLI no command

Returns True if the operation was successful otherwise False

users_re = <_sre.SRE_Pattern object at 0x3595fd0>

pyeapi.api.users.**instance**(*node*)
Returns an instance of Users

This method will create and return an instance of the Users object passing the value of API to the object. The instance method is required for the resource to be autoloaded by the Node object

Parameters **node** (Node) – The node argument passes an instance of Node to the resource

pyeapi.api.users.**isprivilege**(*value*)
Checks value for valid privilege level

Parameters **value** (*str, int*) – Checks if value is a valid user privilege

Returns True if the value is valid, otherwise False

4.2.16 Varp

Module for managing the VARP configuration in EOS

This module provides an API for configuring VARP resources using EOS and eAPI.

param name The interface name the configuration is in reference to. The interface name is the full interface identifier

type name string

param address The interface IP address in the form of address/len.

type address string

param mtu The interface MTU value. The MTU value accepts integers in the range of 68 to 65535 bytes

type mtu integer

class pyeapi.api.varp.**Varp**(**args*, ***kwargs*)
Bases: *pyeapi.api.abstract.EntityCollection*

get()
Returns the current VARP configuration

The Varp resource returns the following:

- mac_address (str): The virtual-router mac address
- **interfaces (dict): A list of the interfaces that have a** virtual-router address configured.

Returns

A Python dictionary object of key/value pairs that represents the current configuration of the node. If the specified interface does not exist then None is returned:

```
{
    "mac_address": "aa:bb:cc:dd:ee:ff",
    "interfaces": {
        "Vlan100": {
            "addresses": [ "1.1.1.1", "2.2.2.2"]
        },
        "Vlan200": [...]
    }
}
```

interfaces

set_mac_address(*mac_address=None*, *default=False*, *disable=False*)
Sets the virtual-router mac address

This method will set the switch virtual-router mac address. If a virtual-router mac address already exists it will be overwritten.

Parameters

- **mac_address** (*string*) – The mac address that will be assigned as the virtual-router mac address. This should be in the format, aa:bb:cc:dd:ee:ff.
- **default** (*bool*) – Sets the virtual-router mac address to the system default (which is to remove the configuration line).
- **disable** (*bool*) – Negates the virtual-router mac address using the system no configuration command

Returns True if the set operation succeeds otherwise False.

class pyeapi.api.varp.**VarpInterfaces**(*node*)
Bases: *pyeapi.api.abstract.EntityCollection*

The VarpInterfaces class helps manage interfaces with virtual-router configuration.

get(*name*)

getall()

set_addresses(*name*, *addresses=None*, *default=False*, *disable=False*)

pyeapi.api.varp.**instance**(*node*)
Returns an instance of Ipinterfaces

This method will create and return an instance of the Varp object passing the value of node to the instance. This function is required for the resource to be autoloaded by the Node object

Parameters **node** (Node) – The node argument provides an instance of Node to the Varp instance

4.2.17 Vlans

Module for working with EOS VLAN resources

The Vlans resource provides configuration of VLAN resources for an EOS node.

param name The name parameter maps to the VLAN name in EOS. Valid values include any consecutive sequence of numbers, letters and underscore up to the maximum number of characters. This parameter is defaultable.

type name string

param state The state parameter sets the operational state of the VLAN on the node. It has two valid values: active or suspend. The state parameter is defaultable.

type state string

param trunk_groups The trunk_groups parameter provides a list of trunk groups configured for this VLAN. This parameter is defaultable.

type trunk_groups array

class pyeapi.api.vlans.**Vlans**(*node*)
Bases: *pyeapi.api.abstract.EntityCollection*

The Vlans class provides a configuration resource for VLANs

The Vlans class is derived from ResourceBase a standard set of methods for working with VLAN configurations on an EOS node.

add_trunk_group(*vid*, *name*)
Adds a new trunk group to the Vlan in the running-config

EosVersion: 4.13.7M

Parameters

- **vid** (*str*) – The VLAN ID to configure
- **name** (*str*) – The trunk group to add to the list

Returns True if the operation was successful otherwise False

configure_vlan(*vid*, *commands*)
Configures the specified Vlan using commands

Parameters

- **vid** (*str*) – The VLAN ID to configure
- **commands** – The list of commands to configure

Returns True if the commands completed successfully

create(*vid*)
Creates a new VLAN resource

Parameters **vid** (*str*) – The VLAN ID to create

Returns True if create was successful otherwise False

default(*vid*)
Defaults the VLAN configuration

```
default vlan <vlanid>
```

Parameters **vid** (*str*) – The VLAN ID to default

Returns True if the operation was successful otherwise False

delete(*vid*)
Deletes a VLAN from the running configuration

Parameters **vid** (*str*) – The VLAN ID to delete

Returns True if the operation was successful otherwise False

get(*value*)
Returns the VLAN configuration as a resource dict.

Parameters **vid** (*string*) – The vlan identifier to retrieve from the running configuration. Valid values are in the range of 1 to 4095

Returns

A Python dict object containing the VLAN attributes as key/value pairs.

getall()
Returns a dict object of all Vlans in the running-config

Returns A dict object of Vlan attributes

remove_trunk_group(*vid*, *name*)
Removes a trunk group from the list of configured trunk groups for the specified VLAN ID

EosVersion: 4.13.7M

Parameters

- **vid** (*str*) – The VLAN ID to configure
- **name** (*str*) – The trunk group to add to the list

Returns True if the operation was successful otherwise False

set_name(*vid*, *name=None*, *default=False*, *disable=False*)
Configures the VLAN name

EosVersion: 4.13.7M

Parameters

- **vid** (*str*) – The VLAN ID to Configures
- **name** (*str*) – The value to configure the vlan name
- **default** (*bool*) – Defaults the VLAN ID name
- **disable** (*bool*) – Negates the VLAN ID name

Returns True if the operation was successful otherwise False

set_state(*vid*, *value=None*, *default=False*, *disable=False*)
Configures the VLAN state

EosVersion: 4.13.7M

Parameters

- **vid** (*str*) – The VLAN ID to configure
- **value** (*str*) – The value to set the vlan state to
- **default** (*bool*) – Configures the vlan state to its default value
- **disable** (*bool*) – Negates the vlan state

Returns True if the operation was successful otherwise False

set_trunk_groups(*vid*, *value=None*, *default=False*, *disable=False*)
Configures the list of trunk groups support on a vlan

This method handles configuring the vlan trunk group value to default if the default flag is set to True. If the default flag is set to False, then this method will calculate the set of trunk group names to be added and to be removed.

EosVersion: 4.13.7M

Parameters

- **vid** (*str*) – The VLAN ID to configure
- **value** (*str*) – The list of trunk groups that should be configured for this vlan id.
- **default** (*bool*) – Configures the trunk group value to default if this value is true
- **disable** (*bool*) – Negates the trunk group value if set to true

Returns True if the operation was successful otherwise False

pyeapi.api.vlans.**instance**(*node*)
Returns an instance of Vlans

This method will create and return an instance of the Vlans object passing the value of API to the object. The instance method is required for the resource to be autoloaded by the Node object

Parameters **node** (Node) – The node argument passes an instance of Node to the resource

pyeapi.api.vlans.**isvlan**(*value*)
Checks if the argument is a valid VLAN

A valid VLAN is an integer value in the range of 1 to 4094. This function will test if the argument falls into the specified range and is considered a valid VLAN

Parameters **value** – The value to check if is a valid VLAN

Returns True if the supplied value is a valid VLAN otherwise False

4.2.18 Vrfs

Module for working with EOS VRF resources

The Vrfs resource provides configuration of VRF resources for an EOS node.

param name The name parameter maps to the VRF name in EOS. Valid values include any consecutive sequence of numbers, letters and underscore up to the maximum number of characters. This parameter is defaultable.

type name string

param description The vrf description set by the user

type description string

param ipv4_routing Tells whether IPv4 routing is enabled on the VRF

type ipv4_routing bool

param ipv6_routing Tells whether IPv6 unicast routing is enabled on the VRF

type ipv6_routing bool

class pyeapi.api.vrfs.**Vrfs**(*node*)

Bases: *pyeapi.api.abstract.EntityCollection*

The Vrfs class provides a configuration resource for VRFs

The Vrfs class is derived from ResourceBase a standard set of methods for working with VRF configurations on an EOS node.

configure_vrf(*vrf_name*, *commands*)

Configures the specified VRF using commands

Parameters

- **vrf_name** (*str*) – The VRF name to configure
- **commands** – The list of commands to configure

Returns True if the commands completed successfully

create(*vrf_name*, *rd=None*)

Creates a new VRF resource

Note: A valid RD has the following format admin_ID:local_assignment. The admin_ID can be an AS number or globally assigned IPv4 address. The local_assignment can be an integer between 0-65,535 if the admin_ID is an IPv4 address and can be between 0-4,294,967,295 if the admin_ID is an AS number. If the admin_ID is an AS number the local_assignment could also be in the form of an IPv4 address.

Parameters

- **vrf_name** (*str*) – The VRF name to create
- **rd** (*str*) – The value to configure the vrf rd

Returns True if create was successful otherwise False

default(*vrf_name*)

Defaults the VRF configuration for given name

Parameters **vrf_name** (*str*) – The VRF name to default

Returns True if the operation was successful otherwise False

delete(*vrf_name*)

Deletes a VRF from the running configuration

Parameters **vrf_name** (*str*) – The VRF name to delete

Returns True if the operation was successful otherwise False

get(*value*)

Returns the VRF configuration as a resource dict.

Parameters **value** (*string*) – The vrf name to retrieve from the running configuration.

Returns

A Python dict object containing the VRF attributes as key/value pairs.

getall()
Returns a dict object of all VRFs in the running-config

Returns A dict object of VRF attributes

set_description(*vrf_name*, *description=None*, *default=False*, *disable=False*)
Configures the VRF description

Parameters

- **vrf_name** (*str*) – The VRF name to configure
- **description** (*str*) – The string to set the vrf description to
- **default** (*bool*) – Configures the vrf description to its default value
- **disable** (*bool*) – Negates the vrf description

Returns True if the operation was successful otherwise False

set_interface(*vrf_name*, *interface*, *default=False*, *disable=False*)
Adds a VRF to an interface

Notes

Requires interface to be in routed mode. Must apply ip address after VRF has been applied. This feature can also be accessed through the interfaces api.

Parameters

- **vrf_name** (*str*) – The VRF name to configure
- **interface** (*str*) – The interface to add the VRF too
- **default** (*bool*) – Set interface VRF forwarding to default
- **disable** (*bool*) – Negate interface VRF forwarding

Returns True if the operation was successful otherwise False

set_ipv4_routing(*vrf_name*, *default=False*, *disable=False*)
Configures ipv4 routing for the vrf

Parameters

- **vrf_name** (*str*) – The VRF name to configure
- **default** (*bool*) – Configures ipv4 routing for the vrf value to default if this value is true
- **disable** (*bool*) – Negates the ipv4 routing for the vrf if set to true

Returns True if the operation was successful otherwise False

set_ipv6_routing(*vrf_name*, *default=False*, *disable=False*)
Configures ipv6 unicast routing for the vrf

Parameters

- **vrf_name** (*str*) – The VRF name to configure

- **default** (*bool*) – Configures ipv6 unicast routing for the vrf value to default if this value is true
- **disable** (*bool*) – Negates the ipv6 unicast routing for the vrf if set to true

Returns True if the operation was successful otherwise False

set_rd(*vrf_name*, *rd*)
Configures the VRF rd (route distinguisher)

Note: A valid RD has the following format admin_ID:local_assignment. The admin_ID can be an AS number or globally assigned IPv4 address. The local_assignment can be an integer between 0-65,535 if the admin_ID is an IPv4 address and can be between 0-4,294,967,295 if the admin_ID is an AS number. If the admin_ID is an AS number the local_assignment could also be in the form of an IPv4 address.

Parameters

- **vrf_name** (*str*) – The VRF name to set rd for
- **rd** (*str*) – The value to configure the vrf rd

Returns True if the operation was successful otherwise False

pyeapi.api.vrfs.**instance**(*node*)
Returns an instance of Vrfs

This method will create and return an instance of the Vrfs object passing the value of API to the object. The instance method is required for the resource to be autoloaded by the Node object

Parameters **node** (Node) – The node argument passes an instance of Node to the resource

4.2.19 Vrrp

Module for working with EOS VRRP resources

The Vrrp resource provides configuration management of interface specific vrrp resources on and EOS node. It provides the following class implementations:

- Vrrp - Configure vrrps in EOS

Vrrp Attributes:

- enable (boolean): The shutdown state of the vrrp
- primary_ip (string): The ip address of the vrrp
- **secondary_ip (dict): The secondary ip addresses configured for the vrrp** This is a dictionary in the format:

```
{ key: [ list of ip addresses ] }
```

where key is 'add', 'remove', or 'exists'. 'add' is used to add the list of secondary ip addresses to the vrrp. 'remove' will remove the list of secondary ip addresses from the vrrp. 'exists' is a report only key for retrieving the current secondary ip addresses on a vrrp.

- priority (int): The priority rank of the vrrp
- description (string): The description for the vrrp
- ip_version (int): The ip version value for the vrrp
- timers_advertise (int): The timers advertise setting for the vrrp

- **mac_addr_adv_interval (int): The mac-address advertisement-interval** setting for the vrrp
- preempt (boolean): The preempt state of the vrrp
- preempt_delay_min (int): The preempt delay minimum setting for the vrrp
- preempt_delay_reload (int): The preempt delay reload setting for the vrrp
- delay_reload (int): The delay reload setting for the vrrp
- track (list): The object tracking settings for the vrrp
- bfd_ip (string): The bfd ip set for the vrrp

Notes

The get method will return a dictionary of all the currently configured vrrps on a single interface, with the VRID of each vrrp as the keys in the dictionary:

```
{
    vrrp1: { data },
    vrrp2: { data },
}
```

The getall method will return a dictionary of all the currently configured vrrps on the node, with the interface name as the top-level keys, with the VRIDs for each vrrp on an interface as a sub-key of that interface:

```
{
    interface1: {
        vrrp1: { data },
        vrrp2: { data },
    },
    interface2: {
        vrrp1: { data },
        vrrp2: { data },
    }
}
```

The data for a configured vrrp is a dictionary with the following format:

```
{
    enable: <True|False>
    primary_ip: <string>
    priority: <int>
    description: <string|None>
    secondary_ip: {
        exists: [ <ip string1>, <ip string2> ]
        }
    ip_version: <int>
    timers_advertise: <int>
    mac_addr_adv_interval: <int>
    preempt: <True|False>
    preempt_delay_min: <int>
    preempt_delay_reload: <int>
    delay_reload: <int>
    track: [
        {
            name: <string>
            action: <shutdown|decrement>
```

```
            amount: <int>|default|no|None
        },
        {
            name: <string>
            action: <shutdown|decrement>
            amount: <int>|default|no|None
        },
    ]
    bfd_ip: <string>
}
```

The create and method accepts a kwargs dictionary which defines the properties to be applied to the new or existing vrrp configuration. The available keywords and values are as follows:

- enable: True to enable (no shutdown)|False to disable (shutdown)
- primary_ip: <ip_string>|no|default|None
- priority: <int>|no|default|None
- description: <string>|no|default|None
- **secondary_ip: <dict> may include the following**
  - add: <list of ip address strings>
  - remove: <list of ip address strings>
- ip_version: <int>|no|default|None
- timers_advertise: <int>|no|default|None
- mac_addr_adv_interval: <int>|no|default|None
- preempt: True to enable (preempt)|False to disable (no preempt)
- preempt_delay_min: <int>|no|default|None
- preempt_delay_reload: <int>|no|default|None
- delay_reload: <int>|no|default|None
- track: <list> of dicts in the following format:

```
{
    name: <string>
    action: <shutdown|decrement>
    amount: <int>|default|no|None
}
```

- bfd_ip: <ip string>|no|default|None

class pyeapi.api.vrrp.**Vrrp**(*node*)

Bases: *pyeapi.api.abstract.EntityCollection*

The Vrrp class provides management of the VRRP configuration

The Vrrp class is derived from EntityCollection and provides an API for working with the node's vrrp configurations.

create(*interface*, *vrid*, ***kwargs*)

Creates a vrrp instance from an interface

Note: This method will attempt to create a vrrp in the node's operational config. If the vrrp already exists on the interface, then this method will set the properties of the existing vrrp to those that have been passed in, if possible.

Parameters

- **interface** (*string*) – The interface to configure.
- **vrid** (*integer*) – The vrid number for the vrrp to be created.
- **kwargs** (*dict*) – A dictionary specifying the properties to be applied to the new vrrp instance. See library documentation for available keys and values.

Returns True if the vrrp could be created otherwise False (see Node)

default (*interface*, *vrid*)
Defaults a vrrp instance from an interface

Note: This method will attempt to default the vrrp on the node's operational config. Default results in the deletion of the specified vrrp . If the vrrp does not exist on the interface then this method will not perform any changes but still return True

Parameters

- **interface** (*string*) – The interface to configure.
- **vrid** (*integer*) – The vrid number for the vrrp to be defaulted.

Returns True if the vrrp could be defaulted otherwise False (see Node)

delete (*interface*, *vrid*)
Deletes a vrrp instance from an interface

Note: This method will attempt to delete the vrrp from the node's operational config. If the vrrp does not exist on the interface then this method will not perform any changes but still return True

Parameters

- **interface** (*string*) – The interface to configure.
- **vrid** (*integer*) – The vrid number for the vrrp to be deleted.

Returns True if the vrrp could be deleted otherwise False (see Node)

get (*name*)
Get the vrrp configurations for a single node interface

Parameters **name** (*string*) – The name of the interface for which vrrp configurations will be retrieved.

Returns A dictionary containing the vrrp configurations on the interface. Returns None if no vrrp configurations are defined or if the interface is not configured.

getall ()
Get the vrrp configurations for all interfaces on a node

Returns A dictionary containing the vrrp configurations on the node, keyed by interface.

set_bfd_ip(*name*, *vrid*, *value=None*, *disable=False*, *default=False*, *run=True*)
Set the bfd_ip property of the vrrp

Parameters

- **name** (*string*) – The interface to configure.
- **vrid** (*integer*) – The vrid number for the vrrp to be managed.
- **value** (*string*) – The bfd ip address to be set.
- **disable** (*boolean*) – Unset bfd ip if True.
- **default** (*boolean*) – Set bfd ip to default if True.
- **run** (*boolean*) – Set to True to execute the command, False to return a string with the formatted command.

Returns

If run is True, returns True if the command executed successfully, error if failure.

If run is False, returns the formatted command string which can be passed to the node

set_delay_reload(*name*, *vrid*, *value=None*, *disable=False*, *default=False*, *run=True*)
Set the preempt_delay_min property of the vrrp

Parameters

- **name** (*string*) – The interface to configure.
- **vrid** (*integer*) – The vrid number for the vrrp to be managed.
- **value** (*integer*) – Preempt delay reload value to set on the vrrp.
- **disable** (*boolean*) – Unset preempt delay reload if True.
- **default** (*boolean*) – Set preempt delay reload to default if True.
- **run** (*boolean*) – Set to True to execute the command, False to return a string with the formatted command.

Returns

If run is True, returns True if the command executed successfully, error if failure.

If run is False, returns the formatted command string which can be passed to the node

set_description(*name*, *vrid*, *value=None*, *disable=False*, *default=False*, *run=True*)
Set the description property of the vrrp

Parameters

- **name** (*string*) – The interface to configure.
- **vrid** (*integer*) – The vrid number for the vrrp to be managed.
- **value** (*string*) – Description to assign to the vrrp.
- **disable** (*boolean*) – Unset description if True.
- **default** (*boolean*) – Set description to default if True.
- **run** (*boolean*) – Set to True to execute the command, False to return a string with the formatted command.

Returns

If run is True, returns True if the command executed successfully, error if failure.

If run is False, returns the formatted command string which can be passed to the node

set_enable(*name*, *vrid*, *value=False*, *run=True*)
Set the enable property of the vrrp

Parameters

- **name** (*string*) – The interface to configure.
- **vrid** (*integer*) – The vrid number for the vrrp to be managed.
- **value** (*boolean*) – True to enable the vrrp, False to disable.
- **run** (*boolean*) – True to execute the command, False to return a string with the formatted command.

Returns

If run is True, returns True if the command executed successfully, error if failure

If run is False, returns the formatted command string which can be passed to the node

set_ip_version(*name*, *vrid*, *value=None*, *disable=False*, *default=False*, *run=True*)
Set the ip_version property of the vrrp

Parameters

- **name** (*string*) – The interface to configure.
- **vrid** (*integer*) – The vrid number for the vrrp to be managed.
- **value** (*integer*) – IP version to assign to the vrrp.
- **disable** (*boolean*) – Unset ip_version if True.
- **default** (*boolean*) – Set ip_version to default if True.
- **run** (*boolean*) – Set to True to execute the command, False to return a string with the formatted command.

Returns

If run is True, returns True if the command executed successfully, error if failure.

If run is False, returns the formatted command string which can be passed to the node

set_mac_addr_adv_interval(*name*, *vrid*, *value=None*, *disable=False*, *default=False*, *run=True*)
Set the mac_addr_adv_interval property of the vrrp

Parameters

- **name** (*string*) – The interface to configure.
- **vrid** (*integer*) – The vrid number for the vrrp to be managed.
- **value** (*integer*) – mac-address advertisement-interval value to assign to the vrrp.
- **disable** (*boolean*) – Unset mac-address advertisement-interval if True.
- **default** (*boolean*) – Set mac-address advertisement-interval to default if True.
- **run** (*boolean*) – Set to True to execute the command, False to return a string with the formatted command.

Returns

If run is True, returns True if the command executed successfully, error if failure.

If run is False, returns the formatted command string which can be passed to the node

set_preempt(*name*, *vrid*, *value=None*, *disable=False*, *default=False*, *run=True*)
Set the preempt property of the vrrp

Parameters

- **name** (*string*) – The interface to configure.
- **vrid** (*integer*) – The vrid number for the vrrp to be managed.
- **value** (*boolean*) – True to enable preempt, False to disable preempt on the vrrp.
- **disable** (*boolean*) – Unset preempt if True.
- **default** (*boolean*) – Set preempt to default if True.
- **run** (*boolean*) – Set to True to execute the command, False to return a string with the formatted command.

Returns

If run is True, returns True if the command executed successfully, error if failure.

If run is False, returns the formatted command string which can be passed to the node

set_preempt_delay_min(*name*, *vrid*, *value=None*, *disable=False*, *default=False*, *run=True*)
Set the preempt_delay_min property of the vrrp

Parameters

- **name** (*string*) – The interface to configure.
- **vrid** (*integer*) – The vrid number for the vrrp to be managed.
- **value** (*integer*) – Preempt delay minimum value to set on the vrrp.
- **disable** (*boolean*) – Unset preempt delay minimum if True.
- **default** (*boolean*) – Set preempt delay minimum to default if True.
- **run** (*boolean*) – Set to True to execute the command, False to return a string with the formatted command.

Returns

If run is True, returns True if the command executed successfully, error if failure.

If run is False, returns the formatted command string which can be passed to the node

set_preempt_delay_reload(*name*, *vrid*, *value=None*, *disable=False*, *default=False*, *run=True*)
Set the preempt_delay_min property of the vrrp

Parameters

- **name** (*string*) – The interface to configure.
- **vrid** (*integer*) – The vrid number for the vrrp to be managed.
- **value** (*integer*) – Preempt delay reload value to set on the vrrp.
- **disable** (*boolean*) – Unset preempt delay reload if True.
- **default** (*boolean*) – Set preempt delay reload to default if True.

- **run** (*boolean*) – Set to True to execute the command, False to return a string with the formatted command.

Returns

If run is True, returns True if the command executed successfully, error if failure.

If run is False, returns the formatted command string which can be passed to the node

set_primary_ip (*name*, *vrid*, *value=None*, *disable=False*, *default=False*, *run=True*)
Set the primary_ip property of the vrrp

Parameters

- **name** (*string*) – The interface to configure.
- **vrid** (*integer*) – The vrid number for the vrrp to be managed.
- **value** (*string*) – IP address to be set.
- **disable** (*boolean*) – Unset primary ip if True.
- **default** (*boolean*) – Set primary ip to default if True.
- **run** (*boolean*) – Set to True to execute the command, False to return a string with the formatted command.

Returns

If run is True, returns True if the command executed successfully, error if failure.

If run is False, returns the formatted command string which can be passed to the node

set_priority (*name*, *vrid*, *value=None*, *disable=False*, *default=False*, *run=True*)
Set the primary_ip property of the vrrp

Parameters

- **name** (*string*) – The interface to configure.
- **vrid** (*integer*) – The vrid number for the vrrp to be managed.
- **value** (*integer*) – Priority to assign to the vrrp.
- **disable** (*boolean*) – Unset priority if True.
- **default** (*boolean*) – Set priority to default if True.
- **run** (*boolean*) – Set to True to execute the command, False to return a string with the formatted command.

Returns

If run is True, returns True if the command executed successfully, error if failure.

If run is False, returns the formatted command string which can be passed to the node

set_secondary_ips (*name*, *vrid*, *secondary_ips*, *run=True*)
Configure the secondary_ip property of the vrrp

Notes

set_secondary_ips takes a list of secondary ip addresses which are to be set on the virtal router. An empty list will remove any existing secondary ip addresses from the vrrp. A list containing addresses will configure the virtual router with only the addresses specified in the list - any existing addresses not included in the list will be removed.

Parameters

- **name** (*string*) – The interface to configure.
- **vrid** (*integer*) – The vrid number for the vrrp to be managed.
- **secondary_ips** (*list*) – A list of secondary ip addresses to be assigned to the virtual router.
- **run** (*boolean*) – Set to True to execute the command, False to return a string with the formatted command.

Returns

If run is True, returns True if the command executed successfully, error if failure.

If run is False, returns the formatted command string which can be passed to the node

set_timers_advertise(*name*, *vrid*, *value=None*, *disable=False*, *default=False*, *run=True*)
Set the ip_version property of the vrrp

Parameters

- **name** (*string*) – The interface to configure.
- **vrid** (*integer*) – The vrid number for the vrrp to be managed.
- **value** (*integer*) – Timers advertise value to assign to the vrrp.
- **disable** (*boolean*) – Unset timers advertise if True.
- **default** (*boolean*) – Set timers advertise to default if True.
- **run** (*boolean*) – Set to True to execute the command, False to return a string with the formatted command.

Returns

If run is True, returns True if the command executed successfully, error if failure.

If run is False, returns the formatted command string which can be passed to the node

set_tracks(*name*, *vrid*, *tracks*, *run=True*)
Configure the track property of the vrrp

Notes

set_tracks takes a list of tracked objects which are to be set on the virtual router. An empty list will remove any existing tracked objects from the vrrp. A list containing track entries configures the virtual router to track only the objects specified in the list - any existing tracked objects on the vrrp not included in the list will be removed.

Parameters

- **name** (*string*) – The interface to configure.
- **vrid** (*integer*) – The vrid number for the vrrp to be managed.
- **tracks** (*list*) – A list of track definition dictionaries. Each dictionary is a definition of a tracked object in one of the two formats:

```
{'name': tracked_object_name,
 'action': 'shutdown'}
{'name': tracked_object_name,
```

```
'action': 'decrement',
'amount': amount_of_decrement}
```

- **run** (*boolean*) – Set to True to execute the command, False to return a string with the formatted command.

Returns

If run is True, returns True if the command executed successfully, error if failure.

If run is False, returns the formatted command string which can be passed to the node

vrconf_format(*vrconfig*)

Formats a vrrp configuration dictionary to match the information as presented from the get and getall methods. vrrp configuration dictionaries passed to the create method may contain data for setting properties which results in a default value on the node. In these instances, the data for setting or changing the property is replaced with the value that would be returned from the get and getall methods.

Intended for validating updated vrrp configurations.

pyeapi.api.vrrp.**instance**(*node*)

Returns an instance of Vrrp

Parameters **node** (Node) – The node argument passes an instance of Node to the resource

Returns An instance of Vrrp

Return type object

CHAPTER 5

Requirements

- Arista EOS 4.12 or later
- Arista eAPI enabled for at least one transport (see Official EOS Config Guide at arista.com for details)
- Python 2.7 or 3.4+ (Python 3 support is work in progress)
- Pyeapi requires the netaddr Python module

Note: netaddr gets installed automatically if you use pip to install pyeapi

CHAPTER 6

Configuration Examples Using pyeapi

6.1 Configuring Subinterfaces Using Python Client for eAPI

Subinterfaces can be configured on Ethernet and Port-Channel interfaces. To do this in pyeapi simply call create or delete with your subinterface name.

Subinterfaces require that the primary interface be in routed mode.

```
import pyeapi
node = pyeapi.connect_to('veos01')
node.api('ipinterfaces').create('Ethernet1')
```

At this point the below should be in your running configuration.

```
!
interface Ethernet1
   no switchport
!
```

Next step is to create the subinterface

```
node.api('interfaces').create('Ethernet1.1')
```

```
!
interface Ethernet1
   no switchport
!
interface Ethernet1.1
!
```

Subinterfaces also require a vlan to be applied to them.

```
node.api('interfaces').set_encapsulation('Ethernet1.1', 4)
```

```
!
interface Ethernet1
   no switchport
!
interface Ethernet1.1
   encapsulation dot1q vlan 4
!
```

Using delete in the same format as create will remove the subinterface.

For more detailed information about configuring subinterfaces in EOS, reference the user manual for the version of EOS running on your switch.

CHAPTER 7

Contribute

The Arista EOS+ team is happy to accept community contributions to the Pyeapi project. Please note that all contributions that modify the library behavior require corresponding test cases otherwise the pull request will be rejected.

7.1 Testing

The pyeapi library provides both unit tests and system tests. The unit tests can be run without an EOS node. To run the system tests, you will need to update the `dut.conf` file found in test/fixtures.

7.1.1 Unit Test

To run the unit tests, simply run `make unittest` from the root of the pyeapi source folder

7.1.2 System Test

To run the system tests, simply run `make systest` from the root of the pyeapi source folder.

Tip: To run all tests, use `make tests` from the root of the pyeapi source folder

7.2 Coverage

Contributions should maintain 100% code coverage. You can check this locally before submitting your Pull Request.

1. Run `make unittest`
2. Run `make coverage_report` to confirm code coverage.

CHAPTER 8

Release Notes

8.1 Release 0.8.2

2018-02-09

8.1.1 New Modules

8.1.2 Enhancements

- **Support eapi command revision syntax (158) [jerearista]** Support requests for specific revisions of EOS command output

```
>>> node.enable({'cmd': 'show cvx', 'revision': 2})
[{'command': {'cmd': 'show cvx', 'revision': 2},
  'encoding': 'json',
  'result': {u'clusterMode': False,
             u'controllerUUID': u'',
             u'enabled': False,
             u'heartbeatInterval': 20.0,
             u'heartbeatTimeout': 60.0}}]
```

- **Add clearer error message for bad user credentials. (152) [mharista]**
- **Reformat EapiConnection send methods exception handling. (148) [mharista]**

8.1.3 Fixed

- **Fix route map getall function to find route maps with a hyphen in the name. (154) [mharista]**

8.1.4 Known Caveats

8.2 v0.8.1

2017-07-20

8.2.1 New Modules

8.2.2 Enhancements

8.2.3 Fixed

- **hard coded /command-api was probably a safe default (141)** This issue/PR (142) reverts a change that introduced a bug breaking pyeapi's unix-socket connection. Regardless of using TCP/IP or unix-socket for communicating with the EAPI process the data being sent to the process is formatted in HTTP requests. The HTTP requests should always POST to the /command-api endpoint. The change being reverted by this prevents the unix-socket path from ever erroneously being used as the HTTP endpoint.
- **Execute does not work well with long running commands (138)** Added socket specific error messages in PR (144) to help distinguish errors caused by command timeouts.
- **eAPI does not handle commands sent as unicode strings (137)** Bug fixed in PR (139) unicode commands converted to strings when detected.

8.3 v0.8.0

2017-03-14

8.3.1 New Modules

- **Base API for VRF support. (133) [mharista]** Added new API module for creating VRFs. In addition to creating, configuring and removing VRFs the updates allow for applying a VRF to an interface and creating a VRF specific OSPF instance.

8.3.2 Enhancements

- **Add base extended ACL support. (135) [mharista]** Updated ACL api to include extended ACLs in addition to standard. To create an extended ACL provide the type as extended when creating the ACL (default is standard). Currently extended ACL statements can be added with action, protocol, and source/destination address. The API will determine the type of ACL by name after it has been created for future updates.
- **Add support for creating and deleting ethernet subinterfaces (132) [mharista]** Allow for creation and deletion of ethernet subinterfaces as part of the EthernetInterface class. Subinterfaces are also supported on PortChannel interfaces. An example using the API to create an ethernet subinterface is provided in the docs.
- **Add node attributes from show version command (131) [mharista]** Added information from show version as attributes to a node. Version, version number and model are added. Version number is simply the numeric portion of the version. For example 4.17.1 if the version is 4.17.1M. All three parameters are populated from the output of show version when any one of the parameters is accessed the first time.

- **Add support for eAPI expandAliases parameter (127) [mharista]** Allowed users to provide the expandAliases parameter to eAPI calls. This allows users to use aliased commands via the API. For example if an alias is configured as 'sv' for 'show version' then an API call with sv and the expandAliases parameter will return the output of show version.
- **Add support for autoComplete parameter. Issue 119 (123) [mharista]** Allows users to use short hand commands in eAPI calls. With this parameter included a user can send 'sh ver' via eAPI to get the output of show version.
- **expose portchannel attributes :lacp fallback, lacp fallback timeout (118) [lruslan]** Helps for configuring LACP fallback in EOS.

8.3.3 Fixed

- **API path is hardcoded in EapiConnection.send() (129)** Updated the previously hardcoded path to /command-api in the EAPI connection to use the transport objects path.
- **Cannot run 'no ip virtual-router mac-address' in eos 4.17.x (122)** Fixed command format for negating an ip virtual-router mac-address. Default and disable forms of the command changed and require the mac-address value in EOS 4.17. Update fixes this for newer versions and is backwards compatible.
- **Non-standard HTTP/s port would cause connection to fail (120)** Bug fixed in PR (121) where a port passed in via eapi.conf as a unicode value caused the http connection to fail.

8.4 v0.7.0

2016-09-08

8.4.1 New Modules

- **Add OSPF API (95) [brigoldberg]** Big thanks for the community support!

8.4.2 Enhancements

- **Enhance Node enable() method (100) [dathelen]** This enhancement adds a send_enable flag to the enable and run_commands Node methods. By default the enable command will be sent, however you can now run commands without prepending the enable.
- **Finish OSPF API (99) [dathelen]** Create system tests and add unit tests to increase code coverage.
- **Add Cross-Platform Support for pyeapi (94) [grybak-arista]** Use logging instead of syslog for better cross-platform compatibility. This enhancement provides support for Windows users.

8.4.3 Fixed

- **Allow dot and hyphen in mlag domain-id (91)** Include handling any character in domain-id string, including dot, hyphen, and space.

8.5 v0.6.1

2016-03-04

(This is a hotfix release)

8.5.1 New Modules

- None

8.5.2 Enhancements

- None

8.5.3 Fixed

- **SR56181 - pyeapi 0.5.0 and 0.6.0 aren't able to execute commands with enable password (88)** A regression was introduced in 0.5.0 which broke passing the enable password to the Node() object.

8.6 v0.6.0

2016-02-22

8.6.1 New Modules

- None

8.6.2 Enhancements

- **Added support for multiline commands without having to pass them as a dictionary (78) [dbarrosop]** (See example below)

```
>>> import pyeapi
>>> connection = pyeapi.client.connect(
...     transport='https',
...     host='192.168.56.201',
...     username='vagrant',
...     password='vagrant',
...     port=443,
...     timeout=60
... )
>>> device = pyeapi.client.Node(connection)
>>> device.run_commands('show hostname')
[{u'hostname': u'localhost', u'fqdn': u'localhost'}]
>>> device.run_commands('show banner login')
[{u'loginBanner': u''}]
>>> my_commands = [
...     'configure session whatever',
...     'hostname new-hostname',
```

```
...     'banner login MULTILINE:This is a new banner\nwith different lines!!!',
...     'end'
... ]
>>>
>>> device.run_commands(my_commands)
[{}, {}, {}, {}]
>>> print device.run_commands(['show session-config named whatever diffs'], encoding=
↪'text')[0]['output']
--- system:/running-config
+++ session:/whatever-session-config
@@ -3,6 +3,8 @@
! boot system flash:/vEOS-lab.swi
!
transceiver qsfp default-mode 4x10G
+!
+hostname new-hostname
!
spanning-tree mode mstp
!
@@ -22,6 +24,11 @@
!
no ip routing
!
+banner login
+This is a new banner
+with different lines!!!
+EOF
+!
management api http-commands
  no shutdown
!
>>> device.run_commands(['configure session whatever', 'commit'])
[{}, {}]
>>> device.run_commands('show hostname')
[{u'hostname': u'new-hostname', u'fqdn': u'new-hostname'}]
>>> device.run_commands('show banner login')
[{u'loginBanner': u'This is a new banner\nwith different lines!!!\n'}]
>>>
```

8.7 v0.5.0

2016-02-16

8.7.1 New APIs

- **NTP module (72) [grybak-arista]** Add NTP functionality.

8.7.2 Enhancements

- **Add banner support to System API (75) [dathelen]** Add API support for EOS banners and motd.
- **Issue #18 performance fixes (74) [cheynearista]** Rework underlying HTTP transport to improve receive performance.

- **Redmine issue 648 (73) [grybak-arista]** Fix some instances where an empty string as negation would not properly negate the option/command.
- **setup.py print statement for python 3 (71) [mzbenami]** Reformat print statement to work properly with Python3+
- **Implement add ACL with seq nos (70) [dathelen]** Add a sequence number when adding a new ACL entry.
- **Fix for redmine issues 234 and 268 (68) [grybak-arista]** Reworked some system tests for robustness get_block accepts a config string as well as the default 'running_config'
- **fix #7 and fix #37 (67) [grybak-arista]** Certain command errors will return more detailed information. The connect() method can optionally return a node object.
- **Add disable key to existing modules for negation of properties (65) [grybak-arista]** Modules now take disable=<True/False> to negate the command, rather than overloading value.
- Compatibility fix for current mock versions (64) [wtucker]
- **Add key error checking to set_tracks (63) [grybak-arista]**

8.7.3 Fixed

- **Failure when eapi.conf is not formatted correctly (38)** Adds more robust error handling when parsing eapi.conf files. Also, if an error is found it will syslog the error and continue parsing any other eapi.conf files.

8.7.4 Known Caveats

8.8 v0.4.0

2015-11-05

8.8.1 New APIs

- **Add VRRP (57) [grybak]** Add support for VRRP configuration.
- **Add Staticroute (45) [grybak]** The staticroute API enables you to set static IPv4 routes on your EOS device.
- **Add VARP (43) [phil-arista]** The Varp API includes the subclass VarpInterfaces. These two combine to provide methods to set virtual IP addresses on interfaces as well as set the global virtual-router mac-address.
- **Add Routemap (40) [phil-arista]**

8.8.2 Enhancements

- **Making configure RADIUS compatible (53) [GaryCarneiro]** Modifies the syntax of the `config` method to use `configure terminal` instead of just `configure`.
- **Close #46 (47) [phil-arista]** This enhancement allows you to set the LACP Mode while executing the set_members method. The call would look like `node.api('interfaces').set_members(1, [Ethernet1,Ethernet2], mode='active')`
- **Added support to specify timeout (41) [dbarrosop]** This enhancement provides a way to specify a connection timeout. The default is set to 60 seconds.

- **Add BGP maximum-paths support (36) [phil-arista]** This enhancement adds more attributes to `eos_bgp_config`. This provides the ability to configure `maximum-paths N ecmp M` in your `router bgp R` configuration.
- **Add sshkey support to users API (34) [phil-arista]** This enhancement augments the `users` API to now support SSH Keys.

8.8.3 Fixed

- **client.py 'def enable' returned dictionary key inconsistency (35)** The key that's supposed to be returned is `result` but instead the method formerly returned the key `response`. For now, both `response` and `result` will be returned with the same data, but `response` will be removed in a future release.
- **[API Users] Can't run set_role with no value (33)** The node.api('users').set_role('test') method didn't remove the role or default the role as you would expect. This bug fix resolves that.
- **[API Users] Can't run set_privilege with no value (32)** The set_privilege('user') method did not properly negate the privilege level when no argument was passed into the method.
- **[API interfaces] get_members regex wrongly includes PeerEthernet when lag is up (28)** The get_members() method wrongly included a duplicate member when the `show port-channel N all-ports` showed the PeerEthernetX. The regular expression has been updated to ignore these entries.
- **[API] users - can't create password with non-alpha/int characters (23)** The characters `(){}[]` cannot be part of a username. Documentation has been updated to reflect this.
- **Users with sha512 passwords don't get processed correctly using api('users').getall() (22)** Fixed regex to extract the encrypted passwords accurately.

8.8.4 Known Caveats

- **failure when eapi.conf is not formatted correctly (38)**

8.9 v0.3.3

2015-07-31

- added initial support for bgp api module
- add trunk group functionality to switchports
- add ip routing to system api

8.10 v0.3.2

2015-07-16

- fixes a problem with parsing the hostname value in the system module

8.11 v0.3.1

2015-06-14

- make pyeapi compatible under Python 3.4 with all unit tests passing ok
- added socket_error property to connection to capture socket errors
- adds function to create per vlan vtep flood lists

8.12 v0.3.0

2015-05-04

- fixes an issue with configuring stp portfast edge correctly
- fixes #13
- fixes #11
- added initial support for system api module
- added initial support for acl api module (standard)
- added initial api support for mlag configuration
- added tag feature to eapi.conf

8.13 v0.2.4

2015-04-30

- adds required docs/description.rst for setup.py

8.14 v0.2.3

2015-04-29

- fixes issue with importing syslog module on Windows

8.15 v0.2.2

2015-04-15

- fixes an issue with eAPI error messages that do not return a data key

8.16 v0.2.1

2015-03-28

- restores default certificate validation behavior for py2.7.9

8.17 v0.2.0

2015-03-19

- adds udp_port, vlans and flood_list attributes to vxlan interfaces
- renames spanningtree api module to stp for consistency
- depreciated spanningtree api module in favor of stp
- interfaces module now properly responds to hasattr calls
- fixes an issue with collecting the vxlan global flood list from the config
- fixes an issue with properly parsing portchannel configurations
- adds portfast_type attribute to stp interfaces resource

8.18 v0.1.1

2015-02-17

- adds introspection properties to CommandError for more details (#4)
- changed the default transport from HTTP to HTTPS to align with EOS
- updates the message returned if the connection profile name is not found
- fixes connection name not copied to host parameter if host not configured
- fixes an issue where an ipinterface wasnt properly recognized
- fixes an issue where a switchport interface was propertly recognized

8.19 v0.1.0

2015-01-23

- initial public release of pyeapi
- initial support for vlans
- initial support for interfaces
- initial support for spanningtree
- initial support for switchports
- initial support for ipinterfaces

CHAPTER 9

Support

9.1 Contact

Pyeapi is developed by Arista EOS+ CS and supported by the Arista EOS+ community. Support for the code is provided on a best effort basis by the Arista EOS+ CS team and the community. You can contact the team that develops these modules by sending an email to eosplus-dev@arista.com.

For customers that are looking for a premium level of support, please contact your local account team or email eosplus@arista.com for help.

9.2 Submitting Issues

The Arista EOS+ CS development team uses Github Issues to track discovered bugs and enhancement requests. The issues tracker can be found at https://github.com/arista-eosplus/pyeapi/issues.

For defect issues, please provide as much relevant data as possible as to what is causing the issue, if and how it is reproducible, the version of EOS, python, and any OS details.

For enhancement requests, please provide a brief description of the enhancement request and the version of EOS to be supported.

The issue tracker is monitored by Arista EOS+ CS and issues submitted are categorized and scheduled for inclusion in upcoming Pyeapi versions.

CHAPTER 10

License

CHAPTER 11

Indices and tables

- genindex
- modindex

Python Module Index

p

Index

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V