pyhpecw7 Documentation

Author

Aug 21, 2020

Contents

This project is a fork of HPENetworking/pyhpecw7. The most notable change is the compatibility with Python 3. The original project is unfortunately inactive. As soon as the situation changes, the changes included in this fork will be proposed upstream and, hopefully, this fork will be terminated.

You can install this version with:

```
$ pip install py3hpecw7
```

This library was built to simplify working with HP Comware 7 switches that have a NETCONF API. Rather than to have network developers worry about the underlying NETCONF API, this library provides a means to manage HP Com7 switches through pre-built Python objects that make it extremely easy to get started programming in an HP environment.

Before getting started, it's important to understand the high level workflow required when using this library, which again, is meant to be fairly straight forward and simple.

CHAPTER 1

Step 1 - Create HPCOM7 object for each HP switch being managed

```
>>> from pyhpecw7.comware import HPCOM7
>>> args = dict(host='hp1', username='hp', password='hp123')
>>> device = HPCOM7(**args)
```

CHAPTER 2

Step 2 - Open a Connection to the Device

```
>>> device.open()
<ncclient.manager.Manager object at 0x7fa5088f98d0>
```

CHAPTER 3

Step 3a - Get Config Data

Nearly all features supported such as `Vlan`, which is used in an example below, have a `get_config()` method.

The object is imported, it is instantiated, and the `get_config()` can be called. This method usually returns a Python dictionary with several key value pairs if the configuration resource exists and if it doesn't it's an empty dictionary.

Examples are below.

CHAPTER 4

Step 3b - Configure the Device

Maybe you don't want to get any data, but rather want to make a configuration change.

Making a configuration change is a two-step process (for most configuration features).

Build the configuration, which creates the appropriate configuration object and places it in a *staging* area, but does not push it.

Most methods of the feature class to *build* the configuration are called `build` or `remove`.

Execute, or push, the configuration object(s) to the device.

While in most cases, users may just want or need to push a single object, it is possible to build, or stage, multiple configuration objects and push them all at once in which case they are executed in the order as they were built. Each time a `build` or `remove` is executed, a list is being appended to, which is storing the staged objects.

The final execution (config push) happens by using the `execute` method of the `HPCOM7` object.

Examples are below.

CHAPTER 5

Getting Started

5.1 Initialize and Open Connection to the Device

```
>>> from pyhpecw7.comware import HPCOM7
>>> args = dict(host='hp1', username='hp', password='hp123')
>>> device = HPCOM7(**args)
>>> device.open()
<ncclient.manager.Manager object at 0x7fa5088f98d0>
```

5.2 Examples

5.2.1 Get VLAN Information

```
>>> from pyhpecw7.features.vlan import Vlan
>>> vlan = Vlan(device, '20')
>>> vlan.get_config()
{}
>>> vlan = Vlan(device, '10')
>>> vlan.get_config()
{'name': 'VLAN10_WEB', 'vlanid': '10', 'descr': 'VLAN 0010'}
```

VLAN 20 did not exist on the switch, but VLAN 10 did.

A VLAN object for any VLAN can also return all VLAN IDs that exist on the switch.

```
>>> vlan.get_vlan_list()
['1', '10']
```

5.2.2 Configure a VLAN

Let's create a new VLAN object.

```
>>> vlan = Vlan(device, '20')
```

Now add VLAN 20. To do this we'll use the `build` method.

```
>>> vlan.build(stage=True)
True
```

When `True` is returned, it means that the config object that will be pushed has successfully been staged.

The next step is to execute the change.

```
>>> response = device.execute_staged()
>>> vlan.get_config()
{'name': 'VLAN 0020', 'vlanid': '20', 'descr': 'VLAN 0020'}
```

Setting the vlan name or description could easily be sent in as additional key value pairs (or with a dictionary) when using `build`.

```
>>> args = dict(name='NEWV20', descr='DESCR_20')
>>> vlan.build(stage=True, **args)
True
>>> rsp = device.execute_staged()
>>> vlan.get_config()
{'name': 'NEWV20', 'vlanid': '20', 'descr': 'DESCR_20'}
```

To remove the VLAN, the `remove` method is used.

```
>>> vlan.remove(stage=True)
True
>>> rsp = device.execute_staged()
>>> vlan.get_config()
{}
```

5.2.3 Get Interface Information

```
>>> from pyhpecw7.features.interface import Interface
>>> interface = Interface(device, 'FortyGigE1/0/4')
>>> interface.get_config()
{'admin': 'up', 'duplex': 'auto', 'speed': 'auto', 'description': 'DESCR104', 'type':
↪'routed'}
```

5.2.4 Default an Interface

```
>>> interface.default(stage=True)
True
>>> response = device.execute_staged()
>>> interface.get_config()
{'admin': 'up', 'duplex': 'auto', 'speed': 'auto', 'description': 'FortyGigE1/0/4␣
↪Interface', 'type': 'switched'}
```

5.2.5 Configure an Interface

```
>>> interface.build(stage=True, admin='down', description='TEST_DESCR')
True
>>> rsp = device.execute_staged()
>>> interface.get_config()
{'admin': 'down', 'duplex': 'auto', 'speed': 'auto', 'description': 'TEST_DESCR',
 ↪'type': 'switched'}
```

CHAPTER 6

Contents

6.1 pyhpecw7 package

6.1.1 Subpackages

6.1.1.1 pyhpecw7.features package

Submodules

pyhpecw7.features.cleanerase module

Factory default HPCOM7 devices.

class pyhpecw7.features.cleanerase.**CleanErase**(*device*)
Bases: object

Factory default a HP Comware 7 switch.

Parameters **device** (HPCOM7) – connected instance of a pyhpecw7.comware.HPCOM7 object.

device
connected instance of a pyhpecw7.comware.HPCOM7 object.

Type *HPCOM7*

build(*stage=False*, *factory_default=False*)
Build cmd list to factory default switch and immediately reboot.

Parameters

- **factory_default** (*bool*) – determines if the switch will be reset to factory defaults. It is a safety measure and must be set to for the factory default to take place.
- **stage** (*bool*) – whether to stage the command for later execution, or execute immediately.

Returns True if stage=True and staging is successful. The output of restore command if stage=False

pyhpecw7.features.config module

Mange config files on HPCOM7 devices.

class pyhpecw7.features.config.**Config**(*device*, *filename*)
Bases: object

This class is used to activate a new running config in real-time.

Parameters

- **device** (HPCOM7) – connected instance of a pyhpecw7.comware.HPCOM7 object.
- **filename** (*str*) – absolute path to the file that will be compared and/or activated on the device.

device
connected instance of a pyhpecw7.comware.HPCOM7 object.

Type *HPCOM7*

filename
absolute path to the file that will be compared and/or activated on the device.

Type str

build(*stage=False*)
Stage or execute configuration required to activate new config file.

This method stores the existing running configuration as flash:/safety_file.cfg, loads the new config file, and then stores the newly loaded file to flash:/startup.cfg.

Parameters **stage** (*bool*) – whether to stage the commands or execute immediately

Returns True if stage=True and staging is successful List of etree.Element XML responses if immediate execution

compare_config()
Compare new config file to the existing current running config

Returns

This returns a tuple of two elements that are both lists.

The first element has a summary of diffs (self._diffs) and the second element is the exact output from the 'display diff' command, but as a list (self._original__diffs).

pyhpecw7.features.errors module

Feature-specific errors.

exception pyhpecw7.features.errors.**AggregationGroupError**(*if_name*)
Bases: *pyhpecw7.features.errors.PortChannelError*

exception pyhpecw7.features.errors.**ConfigError**
Bases: *pyhpecw7.features.errors.FeatureError*

exception pyhpecw7.features.errors.**FeatureError**
Bases: *pyhpecw7.errors.PYHPError*

exception pyhpecw7.features.errors.**FileError**(*src=None*, *dst=None*)
Bases: *pyhpecw7.features.errors.FeatureError*

exception pyhpecw7.features.errors.**FileHashMismatchError**(*src*, *dst*, *src_hash*, *dst_hash*)
Bases: *pyhpecw7.features.errors.FileError*

exception pyhpecw7.features.errors.**FileNotEnoughSpaceError**(*src*, *file_size*, *flash_size*)
Bases: *pyhpecw7.features.errors.FileError*

exception pyhpecw7.features.errors.**FileNotReadableError**(*src=None*, *dst=None*)
Bases: *pyhpecw7.features.errors.FileError*

exception pyhpecw7.features.errors.**FileRemoteDirDoesNotExist**(*remote_dir*)
Bases: *pyhpecw7.features.errors.FileError*

exception pyhpecw7.features.errors.**FileTransferError**(*src=None*, *dst=None*)
Bases: *pyhpecw7.features.errors.FileError*

exception pyhpecw7.features.errors.**IRFError**
Bases: *pyhpecw7.features.errors.FeatureError*

exception pyhpecw7.features.errors.**IRFMemberDoesntExistError**(*member_id*)
Bases: *pyhpecw7.features.errors.IRFError*

exception pyhpecw7.features.errors.**InterfaceAbsentError**(*if_name*)
Bases: *pyhpecw7.features.errors.InterfaceError*

exception pyhpecw7.features.errors.**InterfaceCreateError**(*if_name*)
Bases: *pyhpecw7.features.errors.InterfaceError*

exception pyhpecw7.features.errors.**InterfaceError**(*if_name*)
Bases: *pyhpecw7.features.errors.FeatureError*

exception pyhpecw7.features.errors.**InterfaceParamsError**(*if_name*, *params*)
Bases: *pyhpecw7.features.errors.InterfaceError*

exception pyhpecw7.features.errors.**InterfaceRemoveError**(*if_name*)
Bases: *pyhpecw7.features.errors.InterfaceError*

exception pyhpecw7.features.errors.**InterfaceTypeError**(*if_name*, *if_types=None*)
Bases: *pyhpecw7.features.errors.InterfaceError*

exception pyhpecw7.features.errors.**InterfaceVlanMustExist**(*if_name*, *number*)
Bases: *pyhpecw7.features.errors.InterfaceError*

exception pyhpecw7.features.errors.**InvalidConfigFile**
Bases: *pyhpecw7.features.errors.ConfigError*

exception pyhpecw7.features.errors.**InvalidIPAddress**(*ipaddr*)
Bases: *pyhpecw7.features.errors.FeatureError*

exception pyhpecw7.features.errors.**InvalidPortType**(*if_name*, *config_type*, *pc_type*)
Bases: *pyhpecw7.features.errors.PortChannelError*

exception pyhpecw7.features.errors.**IpIfaceMissingData**
Bases: *pyhpecw7.features.errors.IpInterfaceError*

exception pyhpecw7.features.errors.**IpInterfaceError**
Bases: *pyhpecw7.features.errors.FeatureError*

exception pyhpecw7.features.errors.**LengthOfStringError**(*param_name*)
Bases: *pyhpecw7.features.errors.FeatureError*

exception pyhpecw7.features.errors.**PortChannelError**
Bases: *pyhpecw7.features.errors.FeatureError*

exception pyhpecw7.features.errors.**RebootDateError**
Bases: *pyhpecw7.features.errors.RebootError*

exception pyhpecw7.features.errors.**RebootError**
Bases: *pyhpecw7.features.errors.FeatureError*

exception pyhpecw7.features.errors.**RebootTimeError**
Bases: *pyhpecw7.features.errors.RebootError*

exception pyhpecw7.features.errors.**VlanError**
Bases: *pyhpecw7.features.errors.FeatureError*

exception pyhpecw7.features.errors.**VlanIDError**
Bases: *pyhpecw7.features.errors.VlanError*

pyhpecw7.features.facts module

Gather device facts on HPCOM7 devices.

class pyhpecw7.features.facts.**Facts**(*device*)
Bases: object

Gather device facts from a HP Comware 7 device.

Parameters **device** (HPCOM7) – connected instance of a pyhpecw7.comware.HPCOM7 object.

device
connected instance of a pyhpecw7.comware.HPCOM7 object.

Type *HPCOM7*

facts
this is a read-only attribute. Details can be seen in get_facts.

Type dict

facts

get_facts()
Gather facts from the HP Comware 7 device

Returns

This returns a dictionary with several key/value pairs describing the device. See example below.

Example:

```
{
    'hostname': 'HP5930_1',
    'interface_list': ["FortyGigE1/0/1", "FortyGigE1/0/2",
        "FortyGigE1/0/3", "FortyGigE1/0/4"],
    'localtime': '2016-01-30T01:47:07',
    'model': 'HP FF 5930-32QSFP+ Switch',
    'os': '7.1.045 Release 2418P01',
    'serial_number': 'CN43G9800T',
    'uptime': '6d 23hr 25min 18sec',
    'vendor': 'hp',
```

(continues on next page)

(continued from previous page)

```
    'hardware': 'Ver.A'
}
```

pyhpecw7.features.file_copy module

Manage file transfer to HPCOM7 devices.

class pyhpecw7.features.file_copy.**FileCopy**(*device*, *src*, *dst=None*, *port=22*)
Bases: object

This class is used to copy local files to a HPCOM7 device.

Note: SCP should first be enabled on the device.

Note: When using this class, the passed in HPCOM7 object should be constructed with the timeout equal to at least 60 seconds. Remote MD5 sum calculations can take some time.

Note: If the remote directory doesn't exist (check remote_dir_exists), it's the responsibility of the user to call create_remote_dir() before calling transfer_file(). Otherwise, a FileRemoteDirDoesNotExist exception will be raised.

Parameters

- **device** (HPCOM7) – connected instance of a pyhpecw7.comware.HPCOM7 object.
- **src** (*str*) – Full path to local file to be copied.
- **dst** (*str*) – OPTIONAL - Full path or filename of remote file. If just a filename is supplied, 'flash:/' will be prepended. If nothing is supplied, the source filename will be used, and 'flash:/' will be prepended.
- **port** (*int*) – OPTIONAL - The SSH port over which the SCP connection is made. Defaults to 22.

device
connected instance of a pyhpecw7.comware.HPCOM7 object.

Type *HPCOM7*

src
Full path to local file to be copied.

Type str

dst
Full path of remote file.

Type str

port
The SSH port over which the SCP connection is made.

Type int

remote_dir_exists
Whether there remote directory exists.

Type bool

create_remote_dir()
Create the remote directory.

Raises **FileCreateDirectoryError** – if the directory could not be created.

file_already_exists()
Check to see if there is a remote file with the same name and md5 sum.

Returns `True` if exists, `False` otherwise.

transfer_file(*hostname=None*, *username=None*, *password=None*)
Transfer the file to the remote device over SCP.

Note: If any arguments are omitted, the corresponding attributes of the `self.device` will be used.

Parameters

- **hostname** (*str*) – OPTIONAL - The name or IP address of the remote device.
- **username** (*str*) – OPTIONAL - The SSH username for the remote device.
- **password** (*str*) – OPTIONAL - The SSH password for the remote device.

Raises

- ***FileTransferError*** – if an error occurs during the file transfer.
- ***FileHashMismatchError*** – if the source and destination hashes don't match.
- ***FileNotReadableError*** – if the local file doesn't exist or isn't readable.
- ***FileNotEnoughSpaceError*** – if there isn't enough space on the device.
- ***FileRemoteDirDoesNotExist*** – if the remote directory doesn't exist.

pyhpecw7.features.install_os module

Install an operating system on HPCOM7 devices.

class pyhpecw7.features.install_os.**InstallOs**(*device*)
Bases: `object`

This class is used to get and build the startup software image. It is often used in conjunction with `file_copy.FileCopy`, which can transfer an image.

Parameters **device** (HPCOM7) – connected instance of a `pyhpecw7.comware.HPCOM7` object.

device
connected instance of a `pyhpecw7.comware.HPCOM7` object.

Type *HPCOM7*

build(*os_type*, *ipe=None*, *boot=None*, *system=None*, *delete_ipe=False*, *stage=False*)
Stage or execute the configuration commands for changing the primary startup image.

Parameters

- **os_type** (*str*) – REQUIRED - 'ipe' (for IPE packages) or 'bootsys' (for separate boot and system files.
- **ipe** (*str*) – REQUIRED if os_type is 'ipe' - The full path of the remote IPE file.
- **boot** (*str*) – REQUIRED if os_type is 'bootsys' - The full path of the remote boot .bin file.
- **system** (*str*) – REQUIRED if os_type is 'bootsys' - The full path of the remote system .bin file.
- **delete_ipe** (*bool*) – OPTIONAL - Whether to delete the remote IPE file after config change. Defaults to False.
- **stage** (*bool*) – whether to stage the commands or execute immediately

Returns True if stage=True and successfully staged etree.Element XML response if immediate execution

get_config()

Return a dictionary of current and startup image names.

Returns

A dictionary of 'current', 'startup-primary', and 'startup-backup' image file names. For each, there is a 'boot' image and a 'system' image:

```
{
    'current': {
        'boot' : <current boot image>,
        'system': <current system image>,
    }
    'startup-primary': {
        'boot' : <primary startup boot image>,
        'system': <primary startup system image>,
    }
    'startup-backup': {
        'boot': <backup startup boot image>,
        'system': <backup startup system image>
    }
}
```

pyhpecw7.features.interface module

Manage interfaces on HPCOM7 devices.

class pyhpecw7.features.interface.**Interface**(*device*, *interface_name*)

Bases: object

This class is used to get and build interface configurations on HPCOM7 devices.

Parameters

- **device** (HPCOM7) – connected instance of a phyp.comware.HPCOM7 object.
- **interface_name** (*str*) – The name of the interface.

device

connected instance of a phyp.comware.HPCOM7 object.

Type *HPCOM7*

interface_name
The name of the interface.

Type str

iface_index
The device's internal number representation of an interface.

Type str

iface_type
The type of interface, for example: 'LoopBack', 'FortyGigE'.

Type str

is_ethernet
Whether the interface is ethernet.

Type bool

is_routed
Whether the interface is in layer 3 mode. If this is `False`, the interface is either in bridged mode or does not exist.

Type bool

iface_exists
Whether the interface exists. Physical interfaces should always exist. Logical interfaces may or may not exist.

Type bool

build(*stage=False*, ***params*)
Stage or execute the configuration to modify an interface.

Parameters

- **stage** (*bool*) – whether to stage the commands or execute immediately
- ****params** – see Keyword Args.

Keyword Arguments

- **admin** (*str*) – The up/down state of the interface. 'up' or 'down'.
- **speed** (*str*) – The speed of the interface, in Mbps.
- **duplex** (*str*) – The duplex of the interface. 'full', 'half', or 'auto'.
- **description** (*str*) – The textual description of the interface.
- **type** (*str*) – Whether the interface is in layer 2 or layer 3 mode. 'bridged' or 'routed'.

Raises ***InterfaceCreateError*** – if a logical interface cannot be created.

Returns True if stage=True and staging is successful etree.Element XML response if immediate execution

create_logical(*stage=False*)
Stage or execute the configuration to create a logical interface.

Supported types include 'LoopBack', 'Vlan-interface', 'Bridge-Aggregation', and 'Route-Aggregation'

Note: When stage=True, it's the caller's responsibility to call `execute()` on this class's `device` object after this method is called.

Note: After execution, the caller must call `update()` on this class.

Returns True if successful.

Raises ***InterfaceCreateError*** – if the logical interface cannot be created.

default(*stage=False*)
Stage or execute the configuration to default an interface.

stage (bool): whether to stage the commands or execute immediately

Returns True if stage=True and staging is successful etree.Element XML response if immediate execution

get_config()
Return the currently configured parameters for the interface.

Returns

A dictionary of currently configured parameters for the interface, including:

admin (str) The up/down state of the interface. 'up' or 'down'.

speed (str) The speed of the interface, in Mbps.

duplex (str) The duplex of the interface. 'full', 'half', or 'auto'.

description (str) The textual description of the interface.

type (str) Whether the interface is in layer 2 or layer 3 mode. 'bridged' or 'routed'.

get_default_config()
Return the default configuration of an interface.

Returns

A dictionary of default interface configuration parameters, depending on the type of interface.

For example, for ethernet interfaces:

```
{
    'description': 'FortyGigE1/0/1 Interface',
    'admin': 'up',
    'speed': 'auto',
    'duplex': 'auto',
    'type': 'bridged'
}
```

param_check(***params*)
Checks given parameters against the interface for various errors.

Parameters ****params** – see Keyword Args

Keyword Arguments

- **admin** (*str*) – The up/down state of the interface. 'up' or 'down'.
- **speed** (*str*) – The speed of the interface, in Mbps.
- **duplex** (*str*) – The duplex of the interface. 'full', 'half', or 'auto'.

- **description** (*str*) – The textual description of the interface.
- **type** (*str*) – Whether the interface is in layer 2 or layer 3 mode. 'bridged' or 'routed'.

Raises

- ***InterfaceTypeError*** – if the given interface isn't a valid type.
- ***InterfaceAbsentError*** – if the given interface is of type is_ethernet and doesn't exist.
- ***InterfaceParamsError*** – if 'speed' or 'duplex' are supplied for a non ethernet interface.
- ***InterfaceVlanMustExist*** – if the interface is of type 'Vlan-interface' and the the associated vlan doesn't exist.

remove_logical(*stage=False*)

Stage or execute the configuration to remove a logical interface.

Supported types include 'LoopBack', 'Vlan-interface', 'Bridge-Aggregation', and 'Route-Aggregation'

Parameters **stage** (*bool*) – whether to stage the commands or execute immediately

---

Note: It's the caller's responsibility to call `execute()` on this class's `device` object after this method is called.

Returns True if stage=True and staging is successful etree.Element XML response if immediate execution

Raises ***InterfaceCreateError*** – if the logical interface cannot be removed.

update()

Update `self.iface_index` and `self.iface_exists`.

Usually called after a logical interface is created.

Raises ***InterfaceCreateError*** – if the interface hasn't yet been successfully created.

Note: It is the responsibility of the caller to call `update()` after staging (``create_logical()) *and* executing (`execute()` on this class's `device` object) of commands to create an interface.

---

pyhpecw7.features.ipinterface module

Manage layer 3 interfaces on HPCOM7 devices.

class pyhpecw7.features.ipinterface.**IpInterface**(*device*, *interface_name*, *version='v4'*)

Bases: `object`

This class is used to get and build layer 3 interface configurations on `HPCOM7` devices.

Parameters

- **device** (`HPCOM7`) – connected instance of a `pyhpecw7.comware.HPCOM7` object.
- **interface_name** (*str*) – The name of the interface.

- **version** (*str*) – V4 for IPv4, V6 for IPv6

device
connected instance of a `pyhpecw7.comware.HPCOM7` object.

Type *HPCOM7*

interface_name
The name of the interface.

Type str

version
V4 for IPv4, V6 for IPv6

Type str

interface
The associated `Interface` configuration object.

Type pyhpecw7.features.Interface

is_routed
`True` if the interface is in layer 3 mode, `False` otherwise.

Type bool

build(*stage=False*, ***params*)
Stage or execute a configuration to configure an IP address on an interface.

Parameters

- **stage** (*bool*) – whether to stage the commands or execute immediately
- ****params** – see Keyword Args

Keyword Arguments

- **addr** (*str*) – The IP address to add.
- **mask** (*str*) – The network mask for the IP address, in dotted decimal or prefix length notation.

Returns True if stage=True and staging is successful etree.Element XML response if immediate execution

gen_ipv4_config(*params*, *key_map*, *operation*)

gen_ipv4_top()

gen_ipv6_config(*params*, *key_map*, *operation*)

gen_ipv6_top()

get_config()
Return a list of currently configured IP addresses on the interface.

Note: Either only IPv4 or only IPv6 addresses will be returned depending on the version stored in `self.version`.

Returns A list of currently configured IP addresses on the interface.

remove(*stage=False*, ***params*)
Stage or execute a configuration to remove an IP address on an interface.

Parameters

- **stage** (*bool*) – whether to stage the commands or execute immediately
- ****params** – see Keyword Args

Keyword Arguments

- **addr** (*str*) – The IP address to remove.
- **mask** (*str*) – The network mask for the IP address, in dotted decimal or prefix length notation.

Returns True if stage=True and staging is successful etree.Element XML response if immediate execution

pyhpecw7.features.irf module

This module is used to configure IRF for a `HPCOM7` device. The `IRFMember` class is used to change the IRF member numbers on the device, and can force a reboot. The `IRFPort` class is used to bind physical ports to IRF ports. `IRFMember` should be used first and the `IRFPort`.

class pyhpecw7.features.irf.**IrfMember**(*device*)

Bases: `object`

This class is used to get and build IRF port configurations on a `HPCOM7` device.

Parameters **device** (`HPCOM7`) – connected instance of a `pyhpecw7.comware.HPCOM7` object.

device

connected instance of a `pyhpecw7.comware.HPCOM7` object.

Type *HPCOM7*

build(***params*)

Build the IRF membership configuration.

Parameters ****params** – See Keyword Args.

Keyword Arguments

- **auto_update** (*str*) – OPTIONAL - Enables or disables the auto-upgrade of software after an IRF convergence. Should be 'enable' or 'disable'.
- **mad_exclude** (*list*) – OPTIONAL - A list of interfaces to be excluded from MAD mechanisms.
- **member_id** (*str*) – REQUIRED - The current IRF member ID of the device.
- **new_member_id** (*str*) – ID of the device.
- **domain_id** (*str*) –
- **descr** (*str*) – OPTIONAL - A textual description of the IRF member.
- **priority** (*str*) – OPTIONAL - The desired priority of the IRF member.

Returns True if successful, False otherwise.

get_config(*member_id*)

Return the entire IRF membership configuration for a given member ID.

Parameters **member_id** (*str*) – A current IRF member ID for the device.

Returns

A dictionary with IRF membership configuration parameters. Including:

auto_update (str) 'enable' or 'disable'.

mad_exclude (list) A list of interfaces to be excluded from MAD mechanisms.

new_member_id (str) The new IRF member ID of the device to applied after reboot.

descr (str) A textual description of the IRF member.

priority (str) The priority of the IRF member.

domain_id (str) The domain id of the member.

Raises ***IRFMemberDoesntExistError*** – if the IRF member doesn't exist.

remove_mad_exclude(*iface_list*)
Stage the configuration to remove interfaces from the mad excluded list.

class pyhpecw7.features.irf.**IrfPort**(*device*)
Bases: object

This class is used to get and build IRF port configurations on a HPCOM7 device.

Parameters **device** (HPCOM7) – connected instance of a pyhpecw7.comware.HPCOM7 object.

device
connected instance of a pyhpecw7.comware.HPCOM7 object.

Type *HPCOM7*

build(*member_id*, *old_p1=[]*, *old_p2=[]*, *irf_p1=[]*, *irf_p2=[]*, *filename='startup.cfg'*, *activate=True*)
Stage all of the commands to configure IRF ports, including: 1. Bringing down interfaces to be removed or added 2. Binding physical ports to IRF ports 3. Bringing up interfaces to be added. 4. Saving the config. 5. (Optionally) activating the IRF port configuration.

Parameters

- **member_id** (*string*) – The member ID of the switch.
- **old_p1** (*list*) – REQUIRED - A list of current interfaces bound to IRF port 1.
- **old_p2** (*list*) – REQUIRED - A list of current interfaces bound to IRF port 2.
- **irf_p1** (*list*) – REQUIRED - A list of desired interfaces to be bound to IRF port 1.
- **irf_p2** (*list*) – REQUIRED - A list of desired interfaeces to be bound to IRF port 2.
- **filename** (*str*) – OPTIONAL - The filename in which to save the current configuration. Defaults to 'startup.cfg'.
- **activate** (*bool*) – OPTIONAL - Whether to immediately apply the IRF port configuration. Defaults to True.

Note: The irf_p1 and irf_p2 should be the complete physical interface list for IRF port 1 and IRF port 2, respectively. Interfaces not in the list will be removed.

Note: If `old_p1` and `old_p2` are not accurate, behavior is undefined. These values can be obtained from `get_config()`.

Returns A string representation of the list of staged configurations on the device.

get_config()
Get the current configuration of IRF ports on the device.

Returns

A dictionary of IRF port bindings.

It has the following format:

```
{
    <member_id>: {
        'irf_p1' : <iface_list>,
        'irf_p2' : <iface_list>
    }
}
```

pyhpecw7.features.l2vpn module

Manage L2VPN on HPCOM7 devices.

class pyhpecw7.features.l2vpn.**L2VPN**(*device*)
Bases: object

Enable/Disable L2VPN globally on a HP Comware 7 switch.

Parameters **device** (HPCOM7) – connected instance of a pyhpecw7.comware.HPCOM7 object.

device
connected instance of a pyhpecw7.comware.HPCOM7 object.

Type *HPCOM7*

disable(*stage=False*)
Stage or execute a config object to disable L2VPN

Parameters **stage** (*bool*) – whether to stage the commands or execute immediately

Returns True if stage=True and successfully staged etree.Element XML response if immediate execution

enable(*stage=False*)
Stage or execute a config object to enable L2VPN

Parameters **stage** (*bool*) – whether to stage the commands or execute immediately

Returns True if stage=True and successfully staged etree.Element XML response if immediate execution

get_config()
Get current L2VPN global configuration state.

pyhpecw7.features.neighbor module

Gather LLDP neighbor information from HPCOM7 devices.

class pyhpecw7.features.neighbor.**Neighbors**(*device*)
Bases: object

Gather LLDP neighbor information from a HP COM7 switch.

Parameters **device** (HPCOM7) – connected instance of a pyhpecw7.comware.HPCOM7 object.

device
connected instance of a pyhpecw7.comware.HPCOM7 object.

Type *HPCOM7*

lldp
dictionary containing LLDP neighbors

Type lldp

cdp
dictionary containing CDP neighbors

Type cdp

refresh()
Refreshes the "ldp" and "cdp" attributes of the class

pyhpecw7.features.ping module

Ping another device from HPCOM7 devices.

class pyhpecw7.features.ping.**Ping**(*device*, *host*, *vrf=''*, *v6=False*, *detail=False*)
Bases: object

Ping another device from an HPCOM7 device.

Parameters

- **device** (HPCOM7) – connected instance of a pyhpecw7.comware.HPCOM7 object.
- **host** (*str*) – IP address or name to ping from the switch
- **vrf** (*vrf*) – source VRF on the switch the ping will come from
- **v6** (*bool*) – set to true if dest is v6 target
- **detail** (*bool*) – set to true if you want to see per ping (ICMP echo request) response details

---

Note: If an IPv6 address is provided for host, there is no need to set v6 to True, but it doesn't hurt either way.

If a name is used for host and that resolves to a v6 address, then v6 must be set to True.

---

device
connected instance of a pyhpecw7.comware.HPCOM7 object.

Type *HPCOM7*

host
IP address or name to ping from the switch

Type str

vrf
source VRF on the switch the ping will come from

Type vrf

v6
set to true if dest is v6 target

Type bool

detail
set to true if you want to see per ping (ICMP echo request) response details

Type bool

response
see `_ping`

Type dict

param_check(***kvargs*)
Basic param validation for v4 and v6 addresses

pyhpecw7.features.portchannel module

Manage portchannels on HPCOM7 devices.

class pyhpecw7.features.portchannel.**Portchannel**(*device*, *groupid*, *pc_type*)
Bases: object

This class is used to collect data or configure a specific portchannel.

Parameters

- **device** (HPCOM7) – connected instance of a pyhpecw7.comware.HPCOM7 object.
- **groupid** (*str*) – group # of the RAGG/BAGG interface
- **pc_type** (*str*) – must be "bridged" or "routed"

device
connected instance of a pyhpecw7.comware.HPCOM7 object.

Type *HPCOM7*

groupid
group # of the RAGG/BAGG interface

Type str

pc_type
must be "bridged" or "routed"

Type str

members_to_remove
interface names to remove from the portchannel

Type list

desired_lacp_mode
set to "active" or "passive". Should be set when `mode` is set to the value of `lacp` in `build_config`.

Type str

build(*stage=False*, ***portchannel*)
Stage or execute a config object to add/update portchannel

Parameters

- **state** (*str*) – must be "present" or "absent"
- **portchannel** – see Keyword Args
- **stage** (*bool*) – whether to stage the commands or execute immediately

Keyword Arguments

- **members** (*list*) – OPTIONAL - list of members by interface name being configured
- **min_ports** (*str*) – OPTIONAL - number that represents selected-port minimum
- **max_ports** (*str*) – OPTIONAL - number that represents selected-port maximum
- **lacp_to_change** (*list*) – OPTIONAL - list of interfaces that need have their lacp mode changed

---

Note: `desired_lacp_mode` needs to be set for the members in `portchannel['lacp_to_change']` to take effect.

`members_to_remove` can be set to remove members during the build process. This should also be a list of interface names.

Returns True if stage=True and successfully staged List of etree.Element XML responses if immediate execution

get_all_members(*list_type='name'*, *asdict=False*)
Gets ports that are a member to any port channel

Parameters

- **list_type** (*str*) – must be "name" or "ifindex"
- **asdict** (*bool*) – determines if a dict should be returned this overrides list_type (see Returns)

Returns

if asdict=True regardless of other Args, a dict

is returned that has interface names as keys and the group of the port-channel that is config'd on that interface as the key

if list_type is name (default), a list of interface names is returned. The names of interfaces that have any portchannel config'd.

if list_type is set to "ifindex", the list has all ifindexes instead of names of interfaces that have a portchannel config'd.

if list_type is misconfigured, an error string is returned.

Return type 1 of 4 objects can be returned based on input args

get_config()
Get current configuration for a given portchannel

Returns

This returns a dictionary that has the following k/v pairs if the portchannel exists:

groupid (str) group ID of the portchannel

ncgroupid (str) INTERNAL group ID used by the switch to differentiate between bridged, routed, and other types of LAGGs. Kept to assist in troubleshooting.

mode (str) will be "static" or "dynamic"

members (list) list of current members by interface name

min_ports (str) number that represents selected-port minimum

max_ports (str) number that represents selected-port maximum

lacp_modes_by_interface (dict) list of dicts that have two key/value pairs. sample_dict=(interface='FortyGigE1/0/1', lacp_mode='passive')

It returns an empty dictionary if the portchannel group does not exist.

get_index_from_interface(*interface*)
Get IfIndex from interface name

Parameters **interface** (*str*) – name of the interface

Returns This returns the IfIndex for an interface.

get_interface_from_index(*index*)
Return interface name based on a given ifindex

get_lacp_mode_by_name(*name*)
Get current LACP mode for a given interface

Parameters **name** (*str*) – full name of the interface

Returns "active" or "passive"

Return type mode (str)

get_portchannels()
Get a list of portchannel groups that exist on the switch

Returns This returns a list of numbers represented as strings that are the portchannel groups that exist on the switch.

get_selected_port_max()
Get selected port max configuration

Returns This returns the selected-port maximum configured value on the switch, else it returns None

get_selected_port_min()
Get selected port min configuration

Returns This returns the selected-port minimum configured value on the switch, else it returns None

param_check(***portchannel*)
Param validation for portchannel

Parameters

- **state** (*str*) – present or absent
- **portchannel** – see Keyword Args

Keyword Arguments **members** (*list*) – members by interface name being configured

Raises

- ***InvalidPortType*** – when existing port type does not match desired type of portchannel
- ***AggregationGroupError*** – when an interface is already a member of a different portchannel than the one being configured.

remove(*stage=False*)
Stage or execute a config object to remove portchannel

Parameters **stage** (*bool*) – whether to stage the commands or execute immediately

Returns True if stage=True and successfully staged List of etree.Element XML responses if immediate execution

pyhpecw7.features.reboot module

Reboot HPCOM7 devices.

class pyhpecw7.features.reboot.**Reboot**(*device*)
Bases: object

This class is used to reboot a HP COM7 switch.

Parameters **device** (HPCOM7) – connected instance of a pyhpecw7.comware.HPCOM7 object.

device
connected instance of a pyhpecw7.comware.HPCOM7 object.

Type *HPCOM7*

build(*stage=False*, ***reboot*)
Build command list to reboot the switch and send to staging

Parameters

- **stage** (*bool*) – whether to stage the commands or execute immediately
- **reboot** – see Keyword Args

Keyword Arguments

- **reboot** (*bool*) – REQUIRED - set to True to reboot (safety)
- **time** (*str*) – OPTIONAL - must be in HH:MM format
- **date** (*str*) – OPTIONAL - must be in MM/DD/YYYY format
- **delay** (*str*) – OPTIONAL - number representing delay in minutes

Returns True if stage=True and successfully staged etree.Element XML response if immediate execution

param_check(***reboot*)
Param validation for time & date.

Parameters **reboot** – see Keyword Args

Keyword Arguments

- **time** (*str*) – OPTIONAL - must be in HH:MM format
- **date** (*str*) – OPTIONAL - must be in MM/DD/YYYY format

pyhpecw7.features.switchport module

Manage switchports on HPCOM7 devices.

class pyhpecw7.features.switchport.**Switchport**(*device*, *interface_name*)
Bases: object

This class is used to get and build layer 2 interface configurations on HPCOM7 devices.

Parameters

- **device** (HPCOM7) – connected instance of a pyhpecw7.comware.HPCOM7 object.
- **interface_name** (*str*) – The name of the interface.

device
connected instance of a pyhpecw7.comware.HPCOM7 object.

Type *HPCOM7*

interface_name
The name of the interface.

Type str

interface
The associated Interface configuration object.

Type pyhpecw7.features.Interface

build(*stage=False*, ***params*)
Stage a layer 2 configuration with given parameters on switchport.

Parameters **stage** (*bool*) – whether to stage the command or execute immediately

Keyword Arguments

- **link_type** (*str*) – 'access' or 'trunk'.
- **pvid** (*str*) – The access VLAN if link_type is 'access', the native VLAN if link_type is 'trunk'.
- **permitted_vlans** (*str*) – A comma and/or hyphen delimited list of VLAN numbers. Used when link_type is 'trunk'. For example: *1,3-5,7*

Returns True if stage=True and successfully staged etree.Element XML response if immediate execution

convert_interface(*link_type*, *stage=False*)
Stage or execute the commands to toggle an interface between trunk/access.

Parameters

- **link_type** (*str*) – 'access' or 'trunk'.
- **stage** (*bool*) – whether to stage the command or execute immediately

Note: If *link_type* does not equal 'access' or 'trunk', no commands are staged.

Returns True if stage=True and successfully staged etree.Element XML responses if immediate execution

default(*stage=False*)
Stage or execute a layer 2 default configuration.

Parameters **stage** (*bool*) – whether to stage the command or execute immediately

Returns True if stage=True and successfully staged etree.Element XML response if immediate execution

get_config()
Return the current layer 2 settings on the switchport.

Returns

A dictionary of current configuration parameters.

For example:

```
{
    'pvid': '2',
    'link_type': 'trunk',
    'permitted_vlans': '1-5'
}
```

get_default()
Return the default layer 2 settings for a switchport.

Returns

A dictionary of default configuration parameters.

For example:

```
{
    'pvid': '1',
    'link_type': 'access'
}
```

pyhpecw7.features.vlan module

Manage VLANS on HPCOM7 devices.

class pyhpecw7.features.vlan.**Vlan**(*device*, *vlanid=None*)
Bases: object

This class is used to get data and configure a specific VLAN.

Parameters

- **device** (HPCOM7) – connected instance of a pyhpecw7.comware.HPCOM7 object.
- **vlanid** (*str*) – VLAN ID

device
connected instance of a pyhpecw7.comware.HPCOM7 object.

Type *HPCOM7*

vlanid
VLAN ID

Type str

build(*stage=False*, ***vlan*)
Stage or execute XML object for VLAN configuration and send to staging

Parameters

- **stage** (*bool*) – whether to stage the command or execute immediately
- **vlan** – see Keyword Args

Keyword Arguments

- **name** (*str*) – OPTIONAL - VLAN name
- **descr** (*str*) – OPTOINAL - VLAN description

Returns True if stage=True and successfully staged etree.Element XML response if immediate execution

gen_top()

get_config()
Gets current configuration for a given VLAN ID

Parameters **vlanid** (*str*) – REQUIRED - VLAN ID

Returns

This returns a dictionary with the following k/v pairs:

vlanid (str) VLAN ID of the vlan requested

name (str) configured name of the vlan

descr (str) configured descr of the vlan

It returns an empty dictionary if the vlan does not exist

get_vlan_list()
Get a list of VLAN IDs that exist on the switch.

Returns It returns a list of VLAN IDs as strings.

param_check(***vlan*)
Basic param validation for vlan

Parameters

- **state** (*str*) – REQUIRED must be "present" or "absent"
- **vlan** – see Keyword Args
- **Args** (*Keyword*) – vlanid (str): OPTIONAL - VLAN ID name (str): OPTIONAL - VLAN name descr (str): OPTIONAL - VLAN description

remove(*stage=False*)
Stage or execute XML object for VLAN removal and send to staging

Parameters **stage** (*bool*) – whether to stage the command or execute immediately

Returns True if stage=True and successfully staged etree.Element XML response if immediate execution

pyhpecw7.features.vrrp module

Manage VRRP groups on HPCOM7 devices.

class pyhpecw7.features.vrrp.**VRRP**(*device*, *interface*, *vrid*)
Bases: object

This class is used to collect data or configure a VRRP group on a given interface.

Parameters

- **device** (HPCOM7) – connected instance of a pyhpecw7.comware.HPCOM7 object.
- **interface** (*str*) – name of the Layer 3 interface
- **vrid** (*str*) – virtual router ID (group number)

device
connected instance of a pyhpecw7.comware.HPCOM7 object.

Type *HPCOM7*

interface
name of the Layer 3 interface

Type str

vrid
virtual router ID (group number)

Type str

build(*stage=False*, ***kvargs*)
Execute or stage XML VRRP configuration and send to staging

Parameters

- **stage** (*bool*) – whether to stage the command or execute immediately
- **kvargs** – see Keyword Args

Note: If auth is being configured, all three auth Keyword Args are required.

Keyword Arguments

- **vip** (*str*) – REQUIRED - virtual IP address
- **auth_mode** (*str*) – OPTIONAL - “simple” or “md5”
- **key_type** (*str*) – OPTIONAL - “cipher” or “plain”
- **key** (*str*) – OPTIONAL - text string if key_type is “plain” or cipher if key_type is “cipher”
- **priority** (*str*) – OPTIONAL - VRRP priority
- **preempt** (*str*) – OPTIONAL - “yes” or “no” (STRING)

Returns True if stage=True and successfully staged CLI response if immediate execution

get_auth_type()
Get auth type for a given VRID on a given interface

Returns

auth_mode (str) "simple" or "md5" :key_type (str): "cipher" :key (str): it will be a cipher

It will return an empty dictionary if it VRID does not exist.

Return type This returns a dictionary with the following k/v pairs

get_config()
Get the config of a given vrid on a given interface

Returns

auth_mode (str) "simple" or "md5"

key_type (str) "cipher"

key (str) it will be a cipher

priority (str) VRRP priority

preempt (str) "true" or "false" (STRING)

vip (str) virtual IP address

Return type This returns a dictionary with the following k/v pairs

get_vrrp_groups()
Get list of VRIDs for a given Layer 3 interface

Returns This returns a list of VRIDs configured on a given interface.

remove(*stage=False*)
Stage or execute commands to remove VRRP group.

Parameters **stage** (*bool*) – whether to stage the command or execute immediately

Returns True if stage=True and successfully staged CLI response string if immediate execution

shutdown(*stage=False*)
Stage or execute commands to shutdown VRRP group.

Parameters **stage** (*bool*) – whether to stage the command or execute immediately

Returns True if stage=True and successfully staged CLI response string if immediate execution

undoshutdown(*stage=False*)
Stage or execute commands to undo shutdown of VRRP group.

Parameters **stage** (*bool*) – whether to stage the command or execute immediately

Returns True if stage=True and successfully staged CLI response string if immediate execution

pyhpecw7.features.vxlan module

Manage VXLAN configurations on HPCOM7 devices.

class pyhpecw7.features.vxlan.**L2EthService**(*device*, *interface*, *instance*, *vsi*)
Bases: object

This class is used to get data and configure Ethernet Service Instances on Layer 2 interfaces, map to VSI, and perform equiv of xconnect.

Parameters

- **device** (HPCOM7) – connected instance of a pyhpecw7.comware.HPCOM7 object.
- **interface** (*str*) – name of a Layer 2 interface
- **instance** (*str*) – service instance ID to be configured on the interface
- **vsi** (*str*) – name of the VSI being mapped to the instance

device
connected instance of a pyhpecw7.comware.HPCOM7 object.

Type *HPCOM7*

interface
name of a Layer 2 interface

Type str

instance
service instance ID to be configured on the interface

Type str

vsi
name of the VSI being mapped to the instance

Type str

RV_VALUE_MAP = None
supported value definition when using Ansible encapsulation default - 1 encapsulation s-vid VLAN_ID - 4, but adds SVlanRange key as VLAN_ID encapsulation s-vid VLAN_ID only-tagged - 5,

but adds SVlanRange key as VLAN_ID

encapsulation tagged - 3 encapsulation untagged - 2

build(*stage=False*, ***kvargs*)
Builds service instance (and xconn) config object for an interface :param state: "present" or "absent" :type state: str :param kvargs: see Keyword Args

Keyword Arguments

- **vsi** (*str*) – OPTIONAL - name of VSI
- **instance** (*str*) – OPTIONAL - instance ID
- **encap** (*str*) – REQUIRED - ['default', 'untagged', 'tagged', 's-vid', 'only-tagged']
- **vlanid** (*str*) – REQUIRED when encap set to "only-tagged" or "s-vid"
- **access_mode** (*str*) – OPTIONAL - "vlan" or "ethernet"

Note: when encap is set to only-tagged, it also ensures s-vid

get_config()

Get config of a service instance on a given interface for a given VSI

Returns

If the mapping exists, it returns a dictionary with the following key/value pairs:

index (str) value of IfIndex

interface (str) name of interface

vsi (str) name of VSI

instance (str) instance ID

encap (str) ['default', 'untagged', 'tagged', 's-vid', 'only-tagged']

vlanid (str) vlanid PRESENT when `encap` set to "only-tagged" or "s-vid"

access_mode (str) "vlan" or "ethernet"

get_vsi_encap()

Gets encap configuration for a given VXLAN ID

Returns

If a config exists, it returns a dictionary with the following key/value pairs:

index (str) value of IfIndex

instance (str) instance ID

encap (str) ['default', 'untagged', 'tagged', 's-vid', 'only-tagged']

vlanid (str) vlanid PRESENT when `encap` set to "only-tagged" or "s-vid"

get_vsi_map()

Get xconnect config for given interface and service instance

Returns

If the mapping exists, it returns a dictionary with the following key/value pairs:

index (str) value of IfIndex

vsi (str) name of VSI

interface (str) name of interface

instance (str) instance ID

access_mode (str) "vlan" or "ethernet"

remove(*stage=False*)

Stage or execute object to remove service instance configuration

Parameters **stage** (*bool*) – whether to stage the command or execute immediately

vsi_exist()

Check to see if the VSI exists

Returns If returns True if the VSI exists, else false)

class pyhpecw7.features.vxlan.**Tunnel**(*device*, *tunnel*)

Bases: `object`

This class is used to get data and configure VXLAN tunnel interfaces.

Parameters

- **device** (HPCOM7) – connected instance of a `pyhpecw7.comware.HPCOM7` object.

- **tunnel** (*str*) – Tunnel ID

device
connected instance of a pyhpecw7.comware.HPCOM7 object.

Type *HPCOM7*

tunnel
Tunnel ID

Type str

build(*stage=False*, ***kvargs*)
Stage or execute config object to create/update tunnel

Parameters **stage** (*bool*) – whether to stage the command or execute immediately

Returns True if stage=True and successfully staged CLI response strings if immediate execution

get_config()
Get running config for a tunnel interface

Returns

src (str): source IP addr of tunnel dest (str): destination IP addr of tunnel mode (str): mode of tunnel

If the tunnel does not exist, an empty dictionary is returned.

Return type A dictionary is returned with the following k/v pairs

get_global_source()
Get global source address for tunnel interfaces

Returns String that is the global source IP address on the switch

remove(*stage=False*)
Build config object to remove tunnel interface

Parameters **stage** (*bool*) – whether to stage the command or execute immediately

Returns True if stage=True and successfully staged CLI response strings if immediate execution

class pyhpecw7.features.vxlan.**Vxlan**(*device*, *vxlan*, *vsi=None*)
Bases: object

This class is used to get data and configure VXLAN/VSI mappings.

Parameters

- **device** (HPCOM7) – connected instance of a pyhpecw7.comware.HPCOM7 object.
- **vxlan** (*str*) – VXLAN ID
- **vsi** (*str*) – name of the VSI

device
connected instance of a pyhpecw7.comware.HPCOM7 object.

Type *HPCOM7*

vxlan
VXLAN ID

Type str

vsi
name of the VSI

Type str

build(*stage=False*, ***kvargs*)
Stage or execute config for managing tunnels

Parameters

- **state** (*str*) – "present" or "absent"
- **kvargs** – see Keyword Args
- **stage** (*bool*) – whether to stage the command or execute immediately

Keyword Arguments

- **tunnels_to_add** (*list*) – OPTIONAL - tunnels to add to the VXLAN/VSI mapping
- **tunnels_to_remove** (*list*) – OPTIONAL - tunnels to remove to the VXLAN/VSI mapping

Returns True if stage=True and successfully staged List of etree.Element XML responses if immediately executed

create(*stage=False*)
Stage or execute a config for creating a VSI

Parameters **stage** (*bool*) – whether to stage the command or execute immediately

Returns True if stage=True and successfully staged List of etree.Element XML responses if immediately executed

get_config()
Get associated VSI for a given VXLAN ID along with configured tunnels for that given VXLAN/VSI mapping.

Returns

vxlan (str) vxlan id :vsi (str): name of vsi

If the mapping does not exist, an empty dictionary is returned.

Return type Dictionary with the following key/value pairs

get_tunnels()
Get a list of tunnel interface that are mapped to a given VXLAN ID

Returns List of tunnel IDs

remove_vsi(*stage=False*, *vsi=None*)
Stage or execute a config for removing a VSI

Parameters **stage** (*bool*) – whether to stage the command or execute immediately

Returns True if stage=True and successfully staged etree.Element XML response if immediately executed

remove_vxlan(*stage=False*)
Stage or execute a config for removing a VXLAN

Parameters **stage** (*bool*) – whether to stage the command or execute immediately

Returns True if stage=True and successfully staged etree.Element XML response if immediately executed

Module contents

6.1.1.2 pyhpecw7.utils package

Subpackages

pyhpecw7.utils.network package

Module contents

pyhpecw7.utils.templates package

Subpackages

Submodules

pyhpecw7.utils.templates.cli module

pyhpecw7.utils.templates.cli.**get_structured_data**(*template*, *rawtxt*)
Returns structured data given raw text using TextFSM templates

Module contents

pyhpecw7.utils.xml package

Submodules

pyhpecw7.utils.xml.lib module

This module provides several utility functions for dealing with XML text and `etree.Element` XML objects.

pyhpecw7.utils.xml.lib.**action_element_maker**()

pyhpecw7.utils.xml.lib.**config_element_maker**()

pyhpecw7.utils.xml.lib.**config_params**(*pmap*, *key_map*, *value_map={}*, *E=<lxml.builder.ElementMaker object>*, *fill_in=True*)

pyhpecw7.utils.xml.lib.**data_elem_to_dict**(*elem*, *key_map*, *value_map={}*)

pyhpecw7.utils.xml.lib.**data_element_maker**()

pyhpecw7.utils.xml.lib.**elem_to_dict**(*elem*, *ns*, *key_map*, *value_map={}*)
Convert an XML etree.Element to a desired dictionary as specified by the key map and value map. :param elem: An ancestor element

of the tags specified in the key map.

Parameters

- **ns** (*string*) – The namespace to use when searching for XML tags.
- **key_map** (*dict*) – A mapping from desired dictionary keys to XML tag names.

- **value_map** (*dict*) – A mapping from XML tag names to dictionaries of mappings from XML text values to desired dictionary values.

Returns The desired dictionary.

pyhpecw7.utils.xml.lib.**find_in_action**(*query*, *ele*)

pyhpecw7.utils.xml.lib.**find_in_config**(*query*, *ele*)

pyhpecw7.utils.xml.lib.**find_in_data**(*query*, *ele*)

pyhpecw7.utils.xml.lib.**findall_in_action**(*query*, *ele*)

pyhpecw7.utils.xml.lib.**findall_in_data**(*query*, *ele*)

pyhpecw7.utils.xml.lib.**get_text**(*xml*, *tag*)
Return the text from a given tag and XML element.

pyhpecw7.utils.xml.lib.**nc_element_maker**()

pyhpecw7.utils.xml.lib.**operation_kwarg**(*operation=None*)

pyhpecw7.utils.xml.lib.**remove_namespaces**(*xml*)
Remove the namespaces from an `etree.Element` object and return the modified object.

pyhpecw7.utils.xml.lib.**reverse_value_map**(*key_map*, *value_map*)
Utility function for creating a "reverse" value map from a given key map and value map.

pyhpecw7.utils.xml.namespaces module

Module contents

Submodules

pyhpecw7.utils.validate module

pyhpecw7.utils.validate.**valid_ip_network**(*network*)
Take a v4 or v6 network, e.g. '192.168.3.5/24' or '192.168.3.5/255.255.255.0' and return whether it is valid.

Parameters **network** (*str*) – IP address and mask, e.g. '192.168.3.5/24'.

Returns True if valid, False otherwise.

Module contents

6.1.2 Submodules

6.1.2.1 pyhpecw7.comware module

Manage connections to HPCOM7 devices.

class pyhpecw7.comware.**HPCOM7**(***kvargs*)

Bases: object

This class manages the NETCONF connection to an HP Comware switch, and provides methods to execute various NETCONF operations.

Parameters

- **host** – REQUIRED - hostname or IP address of the HP Comware Switch
- **username** – REQUIRED - username used to login to the switch. Requires network-admin access
- **password** – REQUIRED - password used to login to the switch. Requires network-admin access
- **port** – OPTIONAL - integer that represents the NETCONF port being used on the switch. Default is 830.
- **timeout** – OPTIONAL - How long a single RPC rquest should wait before timing out.
- **ssh_config** – OPTIONAL - enables parsing of a OpenSSH configuration file, either file in string, or defaults to ~/.ssh/config if True

staged

Dictionary that stores XML objects prior to being sent to the device. It is a "holding" area. The stage_config method is are used to build this attribute. Keys are user defined to make it possible to recall/view the specified object if more than one are being prepared to send.

staged_cli

Dictionary that stores XML objects when pushing raw CLI commands via NETCONF. It is a "holding" area. The stage_cli_display and stage_cli_config methods are used to build this attribute. Keys are user defined to make it possible to recall/view the specified object if more than one are being prepared to send.

action(*element*)

Wrapper for ncclient.manger.action

Parameters **element** – etree.Element sent to ncclient.manager.action

Returns The etree.Element returned from ncclient.manager.action

cli_config(*command*)

Immediately push config commands to the device and returns text.

Parameters **command**(*list or string*) – config commands

Returns raw text CLI output

cli_display(*command*)

Immediately push display commands to the device and returns text.

Parameters **command**(*list or string*) – display commands

Returns raw text CLI output

close()

Close the NETCONF connection to the HP switch.

connected

`True` if the NETCONF session to the device is open else it is `False`.

edit_config(*config*, *target='running'*)

Send a NETCONF edit_config XML object to the device.

Parameters

- **config** – etree.Element sent to ncclient.manager.edit_config
- **target** – Name of configuration on the remote device. Defaults to 'running'

Returns The etree.Element returned from ncclient.manager.edit_config

execute(*run_cmd_func*, *args=[]*, *kwargs={}*)
Safely execute the supplied function with args and kwargs.

Parameters **run_cmd_func** (*executable*) – Function to be run.

Returns The return value of the supplied function.

Raises

- ***NCError*** – if there is an error in the NETCONF protocol.
- ***NCTimeoutError*** – if a client-side timeout has occured.
- ***ConnectionClosedError*** – if the NETCONF session is closed.

execute_staged(*target='running'*)
Execute/Push the XML object(s) or CLI strings in the staging area (self.staged) to the device.

Parameters **target** (*str*) – must be set to running. It *could* change in the future if HP supports candidate configurations, etc. Only used for 'edit_config' API calls. Defaults to 'running'.

Returns A list of responses received from the device. Responses with CLI information are extracted from the XML response.

facts
A dictionary of a facts about the device. None if not connected.

get(*get_tuple=None*)
Wrapper for ncclient.manger.get

Parameters **get_tuple** – The tuple sent to ncclient.manager.get, e.g: ('subtree', <etree.Element>)

Returns The etree.Element returned from ncclient.manager.get

lock(*target='running'*)
Attempt to lock the NETCONF connection.

Raises

- ***NCError*** – if there is an error in the NETCONF protocol.
- ***LockConflictError*** – if another process hold the NETCONF lock.

open(*hostkey_verify=False*, *allow_agent=True*, *look_for_keys=False*)
Open the NETCONF connection to the HP switch.

Parameters

- **hostkey_verify** (*bool*) – OPTIONAL - enables hostkey verification from ~/.ssh/known_hosts
- **allow_agent** (*bool*) – OPTIONAL - enables querying SSH agent (if found) for keys
- **look_for_keys** (*bool*) – OPTIONAL - enables looking in the usual locations for ssh keys (e.g. ~/.ssh/id_*)

Returns Connection to the device using ncclient's connect method.

Raises

- ***ConnectionAuthenticationError*** – if there is an error authenticating to the device.
- ***ConnectionSSHError*** – if NETCONF isn't enabled on the device, or the device isn't reachable
- ***ConnectionUnkownHostError*** – if the device's network name cannot be resolved to an IP address.
- ***ConnectionError*** – if an unkown error occurs during connection

reboot()
Attempt an immediate reboot of the device.

rollback(*filename*)
Wrapper for ncclient.manger.rollback

Parameters **element** – etree.Element sent to ncclient.manager.rollback

Returns The etree.Element returned from ncclient.manager.rollback

save(*filename=None*)
Wrapper for ncclient.manger.save

Parameters **element** – etree.Element sent to ncclient.manager.save

Returns The etree.Element returned from ncclient.manager.save

stage_config(*config*, *cfg_type*)
Append config object to the staging area.

Parameters

- **config** – The config payload. Could be a partial etree.Element XML object or raw text if using a CLI config type
- **cfg_type** (*string*) – The type of config payload. Permitted options: "edit_config", "action", "cli_config", "cli_display", "save", "rollback"

Returns True if config object was successfully staged.

Raises **ValueError** – if an invalid config type is supplied.

staged_to_string()
Convert the staging area to a list of strings.

Returns A list of string representing the configuration in the staging area.

unlock(*target='running'*)
Attempt to unlock the NETCONF connection.

Raises

- ***NCError*** – if there is an error in the NETCONF protocol.
- ***LockConflictError*** – if another process hold the NETCONF lock.

6.1.2.2 pyhpecw7.errors module

Errors for pyhpecw7 library.

(c) Copyright 2016 Hewlett Packard Enterprise Development LP Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License

at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

exception pyhpecw7.errors.**ConnectionAuthenticationError**(*dev*, *msg=None*)
Bases: *pyhpecw7.errors.ConnectionError*

When there's an authentication error.

exception pyhpecw7.errors.**ConnectionClosedError**(*dev*, *msg=None*)
Bases: *pyhpecw7.errors.ConnectionError*

When there's a connection closed error.

exception pyhpecw7.errors.**ConnectionError**(*dev*, *msg=None*)
Bases: *pyhpecw7.errors.PYHPError*

When there is an error in the SSH/NETCONF connection.

exception pyhpecw7.errors.**ConnectionSSHError**(*dev*, *msg=None*)
Bases: *pyhpecw7.errors.ConnectionError*

When there's an SSH error.

exception pyhpecw7.errors.**ConnectionUnkownHostError**(*dev*, *msg=None*)
Bases: *pyhpecw7.errors.ConnectionError*

When there's an uknown host error.

exception pyhpecw7.errors.**FeatureError**
Bases: *pyhpecw7.errors.PYHPError*

Base class for all errors related to features.

exception pyhpecw7.errors.**LockConflictError**
Bases: *pyhpecw7.errors.PYHPError*

When there's an attempt to lock, and NETCONF lock is not available.

exception pyhpecw7.errors.**NCError**(*rpc_error=None*)
Bases: *pyhpecw7.errors.PYHPError*

Wrapper for ncclient RPC errors.

exception pyhpecw7.errors.**NCTimeoutError**
Bases: *pyhpecw7.errors.PYHPError*

When there is no response from the device within the timeout range.

exception pyhpecw7.errors.**PYHPError**
Bases: Exception

General base class for all errors in the pyhpecw7 library.

exception pyhpecw7.errors.**UnlockConflictError**
Bases: *pyhpecw7.errors.PYHPError*

When there's an attempt to unlock, and NETCONF lock is not available.

6.1.2.3 pyhpecw7.execkeys module

6.1.3 Module contents

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V