Junos PyEZ Documentation

Release 2.5.1+0.gbfb6e25.dirty

Juniper Networks, Inc.

Jul 30, 2020

Contents

Junos PyEZ is a Python library to remotely manage/automate Junos devices

Contents:

CHAPTER 1

jnpr.junos

1.1 jnpr.junos.factory

1.1.1 jnpr.junos.factory.cfgtable

class jnpr.junos.factory.cfgtable.**CfgTable**(*dev=None*, *xml=None*, *path=None*, *mode=None*)

Bases: *jnpr.junos.factory.table.Table*

__init__(*dev=None*, *xml=None*, *path=None*, *mode=None*)

Dev Device instance

Xml lxml Element instance

Path file path to XML, to be used rather than :dev:

Use_filter Default usage is SAX parsing, disable this variable to use DOM

append()

It creates lxml nodes with field name as xml tag and its value given by user as text of xml node. The generated xml nodes are appended to configuration xml at appropriate hierarchy.

> **Warning:** xml node that are appended cannot be changed later hence care should be taken to assign correct value to table fields before calling append.

get(**vargs*, ***kvargs*)

Retrieve configuration data for this table. By default all child keys of the table are loaded. This behavior can be overridden by with kvargs['nameonly']=True

Parameters

- **vargs[0]** (*str*) – identifies a unique item in the table, same as calling with :kvargs['key']: value

- **namesonly** (*str*) – *OPTIONAL* True/False*, when set to True will cause only the the name-keys to be retrieved.
- **key** (*str*) – *OPTIONAL* identifies a unique item in the table
- **options** (*dict*) – *OPTIONAL* options to pass to get-configuration. By default {'inherit': 'inherit', 'groups': 'groups'} is sent.

get_table_xml()
It returns lxml object of configuration xml that is generated from table data (field=value) pairs. To get a valid xml this method should be used after append() is called.

keys_required
True/False - if this Table requires keys

load(***kvargs*)
Load configuration xml having table data (field=value) in candidate db. This method should be used after append() is called to get the desired results.

Parameters

- **overwrite** (*bool*) – Determines if the contents completely replace the existing configuration. Default is `False`.
- **merge** (*bool*) – If set to `True` will set the load-config action to merge. the default load-config action is 'replace'

Returns Class object.

Raises

ConfigLoadError: When errors detected while loading configuration. You can use the Exception errs variable to identify the specific problems

RuntimeError: If field value is set and append() is not invoked before calling this method, it will raise an exception with appropriate error message.

required_keys
return a list of the keys required when invoking :get(): and :get_keys():

reset()
Initialize fields of set table to it's default value (if mentioned in Table/View) else set to None.

set(***kvargs*)
Load configuration data in running db. It performs following operation in sequence.

- lock(): Locks candidate configuration db.
- load(): Load structured configuration xml in candidate db.
- commit(): Commit configuration to runnning db.
- unlock(): Unlock candidate db.

This method should be used after append() is called to get the desired results.

Parameters

- **overwrite** (*bool*) – Determines if the contents completely replace the existing configuration. Default is `False`.
- **merge** (*bool*) – If set to `True` will set the load-config action to merge. the default load-config action is 'replace'
- **comment** (*str*) – If provided logs this comment with the commit.

- **confirm** (*int*) – If provided activates confirm safeguard with provided value as timeout (minutes).
- **timeout** (*int*) – If provided the command will wait for completion using the provided value as timeout (seconds). By default the device timeout is used.
- **sync** (*bool*) – On dual control plane systems, requests that the candidate configuration on one control plane be copied to the other control plane, checked for correct syntax, and committed on both Routing Engines.
- **force_sync** (*bool*) – On dual control plane systems, forces the candidate configuration on one control plane to be copied to the other control plane.
- **full** (*bool*) – When true requires all the daemons to check and evaluate the new configuration.
- **detail** (*bool*) – When true return commit detail as XML

Returns Class object:

Raises

ConfigLoadError: When errors detected while loading configuration. You can use the Exception errs variable to identify the specific problems

CommitError: When errors detected in candidate configuration. You can use the Exception errs variable to identify the specific problems

RuntimeError: If field value is set and append() is not invoked before calling this method, it will raise an exception with appropriate error message.

Warning: If the function does not receive a reply prior to the timeout a RpcTimeoutError will be raised. It is possible the commit was successful. Manual verification may be required.

1.1.2 jnpr.junos.factory.factory_cls

jnpr.junos.factory.factory_cls.**FactoryCMDChildTable**(*title=None*, *regex=None*, *key='name'*, *delimiter=None*, *table_name=None*, *view=None*, *key_items=None*, *item=None*, *eval=None*)

jnpr.junos.factory.factory_cls.**FactoryCMDTable**(*cmd*, *args=None*, *item=None*, *key_items=None*, *key='name'*, *view=None*, *table_name=None*, *title=None*, *delimiter=None*, *eval=None*, *platform='juniper_junos'*, *use_textfsm=False*, ***kwargs*)

jnpr.junos.factory.factory_cls.**FactoryCMDView**(*fields*, ***kvargs*)

Fields dictionary of fields, structure of which is ~internal~ and should not be defined explicitly. use the RunstatMaker.Fields() mechanism to create theserather than hardcoding the dictionary structures; since they might change over time.

Kvargs 'view_name' to name the class. this could be useful for debug or eventual callback mechanisms.

'groups' is a dict of name/xpath assocaited to fields this technique would be used to extract fields from node-set elements like port <if-device-flags>.

'extends' names the base View class to extend. using this technique you can add to existing defined Views.

jnpr.junos.factory.factory_cls.**FactoryCfgTable**(*table_name=None*, *data_dict={}*)

jnpr.junos.factory.factory_cls.**FactoryOpTable**(*cmd*, *args=None*, *args_key=None*, *item=None*, *key='name'*, *view=None*, *table_name=None*, *use_filter=True*)

jnpr.junos.factory.factory_cls.**FactoryTable**(*item*, *key='name'*, *view=None*, *table_name=None*, *use_filter=True*)

jnpr.junos.factory.factory_cls.**FactoryView**(*fields*, ***kvargs*)

Fields dictionary of fields, structure of which is ~internal~ and should not be defined explicitly. use the RunstatMaker.Fields() mechanism to create theserather than hardcoding the dictionary structures; since they might change over time.

Kvargs 'view_name' to name the class. this could be useful for debug or eventual callback mechanisms.

'groups' is a dict of name/xpath assocaited to fields this technique would be used to extract fields from node-set elements like port <if-device-flags>.

'extends' names the base View class to extend. using this technique you can add to existing defined Views.

1.1.3 jnpr.junos.factory.factory_loader

This file contains the FactoryLoader class that is used to dynamically create Runstat Table and View objects from a <dict> of data. The <dict> can originate from any kind of source: YAML, JSON, program. For examples of YAML refer to the .yml files in this jnpr.junos.op directory.

class jnpr.junos.factory.factory_loader.**FactoryLoader**

Bases: object

Used to load a <dict> of data that contains Table and View definitions.

The primary method is :load(): which will return a <dict> of item-name and item-class definitions.

If you want to import these definitions directly into your namespace, (like a module) you would do the following:

loader = FactoryLoader() catalog = loader.load(<catalog_dict>) globals().update(catalog)

If you did not want to do this, you can access the items as the catalog. For example, if your <catalog_dict> contained a Table called MyTable, then you could do something like:

MyTable = catalog['MyTable'] table = MyTable(dev) table.get() ...

__init__()

Initialize self. See help(type(self)) for accurate signature.

load(*catalog_dict*, *envrion={}*)

1.1.4 jnpr.junos.factory.optable

class jnpr.junos.factory.optable.**OpTable**(*dev=None*, *xml=None*, *path=None*, *use_filter=True*)

Bases: *jnpr.junos.factory.table.Table*

get(**vargs*, ***kvargs*)
Retrieve the XML table data from the Device instance and returns back the Table instance - for call-chaining purposes.

If the Table was created with a :path: rather than a Device, then this method will load the XML from that file. In this case, the *vargs, and **kvargs are not used.

ALIAS: __call__

Vargs [0] is the table :arg_key: value. This is used so that the caller can retrieve just one item from the table without having to know the Junos RPC argument.

Kvargs these are the name/value pairs relating to the specific Junos XML command attached to the table. For example, if the RPC is 'get-route-information', there are parameters such as 'table' and 'destination'. Any valid RPC argument can be passed to :kvargs: to further filter the results of the :get(): operation. neato!

NOTES: If you need to create a 'stub' for unit-testing purposes, you want to create a subclass of your table and overload this methods.

jnpr.junos.factory.optable.**generate_sax_parser_input**(*obj*)
Used to generate xml object from Table/view to be used in SAX parsing Args: obj: self object which contains table/view details

Returns: lxml etree object to be used as sax parser input

1.1.5 jnpr.junos.factory.table

class jnpr.junos.factory.table.**Table**(*dev=None*, *xml=None*, *path=None*, *use_filter=True*)
Bases: object

D
the Device instance

ITEM_NAME_XPATH = 'name'

ITEM_XPATH = None

RPC
the Device.rpc instance

USE_FILTER = None

VIEW = None

__init__(*dev=None*, *xml=None*, *path=None*, *use_filter=True*)

Dev Device instance

Xml lxml Element instance

Path file path to XML, to be used rather than :dev:

Use_filter Default usage is SAX parsing, disable this variable to use DOM

get(**vargs*, ***kvargs*)

hostname

is_container
True if this table does not have records, but is a container of fields False otherwise

items()
returns list of tuple(name,values) for each table entry

key_list
the list of keys, as property for caching

keys()

savexml(*path*, *hostname=False*, *timestamp=False*, *append=None*)
Save a copy of the table XML data to a local file. The name of the output file (:path:) can include the name of the Device host, the timestamp of this action, as well as any user-defined appended value. These 'add-ons' will be added to the :path: value prior to the file extension in the order (hostname,timestamp,append), separated by underscore (_).

For example, if both hostname=True and append='BAZ1', then when :path: = '/var/tmp/foo.xml' and the Device.hostname is "srx123", the final file-path will be "/var/tmp/foo_srx123_BAZ1.xml"

Path file-path to write the XML file on the local filesystem

Hostname if True, will append the hostname to the :path:

Timestamp

if True, will append the timestamp to the :path: using the default timestamp format

if <str> the timestamp will use the value as the timestamp format as defied by strf-time()

Append any <str> value that you'd like appended to the :path: value preceding the filename extension.

to_json()

Returns JSON encoded string of entire Table contents

values()
returns list of table entry items()

view
returns the current view assigned to this table

1.1.6 jnpr.junos.factory.view

class jnpr.junos.factory.view.**View**(*table*, *view_xml*)
Bases: object

View is the base-class that makes extracting values from XML data appear as objects with attributes.

D
return the Device instance for this View

EVAL = {}

FIELDS = {}

GROUPS = None

ITEM_NAME_XPATH = 'name'

T
return the Table instance for the View

__init__(*table*, *view_xml*)

Table instance of the RunstatTable

View_xml this should be an lxml etree Elemenet object. This constructor also accepts a list with a single item/XML

asview(*view_cls*)
create a new View object for this item

items()
list of tuple(key,value)

key
return the name of view item

keys()
list of view keys, i.e. field names

name
return the name of view item

refresh()
~~~ EXPERIMENTAL ~~~ refresh the data from the Junos device. this only works if the table provides an "args_key", does not update the original table, just this specific view/item

to_json()

Returns JSON encoded string of entire View contents

updater(*fields=True*, *groups=False*, *all=True*, ***kvargs*)
provide the ability for subclassing objects to extend the definitions of the fields. this is implemented as a context manager with the form called from the subclass constructor:

with self.extend() as more: more.fields = <dict> more.groups = <dict> # optional

values()
list of view values

xml
returns the XML associated to the item

1.1.7 jnpr.junos.factory.viewfields

class jnpr.junos.factory.viewfields.**ViewFields**
Bases: object

Used to dynamically create a field dictionary used with the RunstatView class

__init__()
Initialize self. See help(type(self)) for accurate signature.

astype(*name*, *xpath=None*, *astype=<class 'int'>*, ***kvargs*)
field string value will be passed to function :astype:

This is typically used to do simple type conversions, but also works really well if you set :astype: to a function that does a basic converstion like look at the value and change it to a True/False. For example:

astype=lambda x: True if x == 'enabled' else False

end

flag(*name*, *xpath=None*, ***kvargs*)
field is a flag, results in True/False if the xpath element exists or not. Model this as a boolean type <bool>

group(*name*, *xpath=None*, ***kvargs*)
field is an apply group, results in value of group attr if the xpath element has the associated group attribute.
~~~

int (*name*, *xpath=None*, ***kvargs*)
field is an integer

str (*name*, *xpath=None*, ***kvargs*)
field is a string

table (*name*, *table*)
field is a RunstatTable

1.1.8 Module contents

jnpr.junos.factory.**loadyaml** (*path*)
Load a YAML file at :path: that contains Table and View definitions. Returns a <dict> of item-name anditem-class definition.

If you want to import these definitions directly into your namespace, (like a module) you would do the following:

globals().update(loadyaml(<path-to-yaml-file>))

If you did not want to do this, you can access the items as the <dict>. For example, if your YAML file contained a Table called MyTable, then you could do something like:

catalog = loadyaml(<path-to-yaml-file>) MyTable = catalog['MyTable']

table = MyTable(dev) table.get() ...

class jnpr.junos.factory.**FactoryLoader**
Bases: object

Used to load a <dict> of data that contains Table and View definitions.

The primary method is :load(): which will return a <dict> of item-name and item-class definitions.

If you want to import these definitions directly into your namespace, (like a module) you would do the following:

loader = FactoryLoader() catalog = loader.load(<catalog_dict>) globals().update(catalog)

If you did not want to do this, you can access the items as the catalog. For example, if your <catalog_dict> contained a Table called MyTable, then you could do something like:

MyTable = catalog['MyTable'] table = MyTable(dev) table.get() ...

__init__ ()
Initialize self. See help(type(self)) for accurate signature.

load (*catalog_dict*, *envrion={}*)

1.2 jnpr.junos.facts

A dictionary-like object of read-only facts about the Junos device.

These facts are accessed as the *facts* attribute of a *Device* object instance. For example, if *dev* is an instance of a *Device* object, the hostname of the device can be accessed with:

```
dev.facts['hostname']
```

Force a refresh of all facts with:

```
dev.facts_refresh()
```

Force a refresh of a single fact with:

```
dev.facts_refresh(keys='hostname')
```

Force a refresh of a set of facts with:

```
dev.facts_refresh(keys=('hostname','domain','fqdn'))
```

NOTE: The dictionary key for each available fact is guaranteed to exist. If there is a problem gathering the value of a specific fact/key, or if the fact is not supported on a given platform, then the fact/key will have the value None (the None object, not a string.)

Accessing a dictionary key which does not correspond to an available fact will raise a KeyError (the same behavior as accessing a non-existent key of a normal dict.)

The following dictionary keys represent the available facts and their meaning:

2RE A boolean indicating if the device has more than one Routing Engine installed.

_iri_hostname A dictionary keyed by internal routing instance ip addresses. The value of each key is the internal routing instance hostname for the ip

_iri_ip A dictionary keyed by internal routing instance hostnames. The value of each key is the internal routing instance ip for the hostname

_is_linux A boolean indicating if the device is running linuxkernel.

current_re A list of internal routing instance hostnames for the current RE. These hostnames identify things like the RE's slot ('re0' or 're1'), the RE's mastership state ('master' or 'backup'), and node in a VC ('member0' or 'member1')

domain The domain name configured at the [edit system domain-name] configuration hierarchy.

fqdn The device's hostname + domain

HOME A string indicating the home directory of the current user.

hostname A string containing the hostname of the current Routing Engine.

hostname_info A dictionary keyed on Routing Engine name. The value of each key is the hostname of the Routing Engine.

ifd_style The type of physical interface (ifd) supported by the device. Choices are 'CLASSIC' or 'SWITCH'.

junos_info A two-level dictionary providing Junos software version information for each RE in the system. The first-level key is the name of the RE. The second level key is 'text' for the version as a string and 'object' for the version as a version_info object.

master On a single chassis/node system, a string value of 'RE0' or 'RE1' indicating which RE is master. On a multi-chassis or multi-node system, the value is a list of these strings indicating whether RE0 or RE1 is master. There is one entry in the list for each chassis/node in the system.

model An uppercase string containing the model of the chassis in which the current Routing Engine resides.

model_info A dictionary keyed on Routing Engine name. The value of each key is an uppercase string containing the model of the chassis in which the Routing Engine resides.

personality A string which is generally based on the platform and indicates the behavior of the device.

RE0 A dictionary with information about RE0 (if present). The keys of the dictionary are: mastership_state, status, model, up_time, and last_reboot_reason.

RE1 A dictionary with information about RE1 (if present). The keys of the dictionary are: mastership_state, status, model, up_time, and last_reboot_reason.

RE_hw_mi (Routing Engine hardware multi-instance) A boolean indicating if this is a multi-chassis system.

re_info A three-level dictionary with information about the Routing Engines in the device. The first-level key is the chassis or node name. The second-level key is the slot number, the third-level keys are: mastership_state, status, model, and last_reboot_reason. A first-level key with a value of 'default' will always be present and represents the first chassis/node of the system (Note: the first chasis/node of the system is not necessarily the 'master' node in a VC.) A second-level key with a value of 'default' will always be present for the default chassis/node and represents the first Routing Engine on the first node/chassis. (Note: the first RE of a chassis/node is not necessarily the 'master' RE of the chassis/node. See the RE_master fact for info on the 'master' RE of each chassis/node.)

re_master A dictionary indicating which RE slot is master for each chassis/node in the system. The dictionary key is the chassis or node name. A key with a value of 'default' will always be present and represents the first node/chassis of the system. (Note: the first chassis/node of the system is not necessarily the 'master' node in a VC. See the vc_master fact to determine which chassis/node is the master of a VC.)

serialnumber A string containing the serial number of the device's chassis. If there is no chassis serial number, the serial number of the backplane or midplane is returned.

srx_cluster A boolean indicating if the device is part of an SRX cluster.

srx_cluster_id A string containing the configured cluster id

srx_cluster_redundancy_group A multi-level dictionary of information about the SRX cluster redundancy groups on the device. The first-level key is the redundancy group id. The second-level keys are: cluster_id, failover_count, node0, and node1. The node0 and node1 keys have third-level keys of priority, preempt, status, and failover_mode. The values for this fact correspond to the values of the 'show chassis cluster status' CLI command.

switch_style A string which indicates the Ethernet switching syntax style supported by the device. Possible values are: 'BRIDGE_DOMAIN', 'VLAN', 'VLAN_L2NG', or 'NONE'.

vc_capable A boolean indicating if the device is currently configured in a virtual chassis. In spite of the name, this fact does NOT indicate whether or not the device is CAPABLE of joining a VC.

vc_fabric A boolean indicating if the device is currently in fabric mode.

vc_master A string indicating the chassis/node which is currently the master of the VC.

vc_mode A string indicating the current virtual chassis mode of the device.

version A string containing the Junos version of the current Routing Engine.

version_info The Junos version of the current Routing Engine as a version_info object.

version_RE0 A string containing the Junos version of the RE in slot 0. (Assuming the system contains an RE0.)

version_RE1 A string containing the Junos version of the RE in slot 1. (Assuming the system contains an RE1)

virtual A boolean indicating if the device is virtual.

1.3 jnpr.junos.utils

1.3.1 jnpr.junos.utils.config

class jnpr.junos.utils.config.**Config**(*dev*, *mode=None*, ***kwargs*)

Bases: *jnpr.junos.utils.util.Util*

Overview of Configuration Utilities.

- *commit()*: commit changes
- *commit_check()*: perform the commit check operation
- *diff()*: return the diff string between running and candidate config
- *load()*: load changes into the candidate config
- *lock()*: take an exclusive lock on the candidate config
- *pdiff()*: prints the diff string (debug/helper)
- *rescue()*: controls "rescue configuration"
- *rollback()*: perform the load rollback command
- *unlock()*: release the exclusive lock

__init__(*dev*, *mode=None*, ***kwargs*)

Parameters

- **mode** (*str*) –

  Can be used *only* when creating Config object using context manager

  - "private" - Work in private database
  - "dynamic" - Work in dynamic database
  - "batch" - Work in batch database
  - "exclusive" - Work with Locking the candidate configuration
  - "ephemeral" - Work in default/specified ephemeral instance

- **ephemeral_instance** (*str*) – ephemeral instance name

```
# mode can be private/dynamic/exclusive/batch/ephemeral
with Config(dev, mode='exclusive') as cu:
    cu.load('set system services netconf traceoptions file xyz',
            format='set')
    print cu.diff()
    cu.commit()
```

Warning: Ephemeral databases are an advanced Junos feature which if used incorrectly can have serious negative impact on the operation of the Junos device. We recommend you consult JTAC and/or you Juniper account team before deploying the ephemeral database feature in your network.

commit(***kvargs*)

Commit a configuration.

Parameters

- **comment** (*str*) – If provided logs this comment with the commit.
- **confirm** (*int*) – If provided activates confirm safeguard with provided value as timeout (minutes).
- **timeout** (*int*) – If provided the command will wait for completion using the provided value as timeout (seconds). By default the device timeout is used.
- **sync** (*bool*) – On dual control plane systems, requests that the candidate configuration on one control plane be copied to the other control plane, checked for correct syntax, and committed on both Routing Engines.
- **force_sync** (*bool*) – On dual control plane systems, forces the candidate configuration on one control plane to be copied to the other control plane.
- **full** (*bool*) – When true requires all the daemons to check and evaluate the new configuration.
- **detail** (*bool*) – When true return commit detail as XML
- **ignore_warning** – A boolean, string or list of string. If the value is True, it will ignore all warnings regardless of the warning message. If the value is a string, it will ignore warning(s) if the message of each warning matches the string. If the value is a list of strings, ignore warning(s) if the message of each warning matches at least one of the strings in the list.

  For example:

```
cu.commit(ignore_warning=True)
cu.commit(ignore_warning='Advertisement-interval is '
                         'less than four times')
cu.commit(ignore_warning=['Advertisement-interval is '
                          'less than four times',
                          'Chassis configuration for network '
                          'services has been changed.'])
```

  Note: When the value of ignore_warning is a string, or list of strings, the string is actually used as a case-insensitive regular expression pattern. If the string contains only alpha-numeric characters, as shown in the above examples, this results in a case-insensitive substring match. However, any regular expression pattern supported by the re library may be used for more complicated match conditions.

Returns

- `True` when successful
- Commit detail XML (when detail is True)

Raises ***CommitError*** – When errors detected in candidate configuration. You can use the Exception errs variable to identify the specific problems

Warning: If the function does not receive a reply prior to the timeout a RpcTimeoutError will be raised. It is possible the commit was successful. Manual verification may be required.

commit_check(***kvargs*)

Perform a commit check. If the commit check passes, this function will return `True`. If the commit-check results in warnings, they are reported and available in the Exception errs.

Parameters **timeout** (*int*) – If provided the command will wait for completion using the provided value as timeout (seconds).

Returns `True` if commit-check is successful (no errors)

Raises

- ***CommitError*** – When errors detected in candidate configuration. You can use the Exception errs variable to identify the specific problems
- ***RpcError*** – When underlying ncclient has an error

diff(*rb_id=0*, *ignore_warning=False*)

Retrieve a diff (patch-format) report of the candidate config against either the current active config, or a different rollback.

Parameters **rb_id** (*int*) – rollback id [0..49]

Returns

- `None` if there is no difference
- ascii-text (str) if there is a difference

load(**vargs*, ***kvargs*)

Loads changes into the candidate configuration. Changes can be in the form of strings (text,set,xml, json), XML objects, and files. Files can be either static snippets of configuration or Jinja2 templates. When using Jinja2 Templates, this method will render variables into the templates and then load the resulting change; i.e. "template building".

Parameters

- **vargs[0]** (*object*) – The content to load. If the contents is a string, the framework will attempt to automatically determine the format. If it is unable to determine the format then you must specify the **format** parameter. If the content is an XML object, then this method assumes you've structured it correctly; and if not an Exception will be raised.
- **path** (*str*) – Path to file of configuration on the local server. The path extension will be used to determine the format of the contents:
  - "conf","text","txt" is curly-text-style
  - "set" - ascii-text, set-style
  - "xml" - ascii-text, XML
  - "json" - ascii-text, json

  Note: The format can specifically set using **format**.

- **format** (*str*) – Determines the format of the contents. Refer to options from the **path** description.
- **overwrite** (*bool*) – Determines if the contents completely replace the existing configuration. Default is `False`.

  Note: This option cannot be used if **format** is "set".

- **merge** (*bool*) – If set to `True` will set the load-config action to merge. the default load-config action is 'replace'
- **update** (*bool*) – If set to `True` Compare a complete loaded configuration against the candidate configuration. For each hierarchy level or configuration object that is different in the two configurations, the version in the loaded configuration replaces the version in the candidate configuration. When the configuration is later committed, only system processes that are affected by the changed configuration elements parse the new configuration.

  Note: This option cannot be used if **format** is "set".

- **patch** (*bool*) – If set to `True` will set the load-config action to load patch.
- **template_path** (*str*) – Similar to the **path** parameter, but this indicates that the file contents are `Jinja2` format and will require template-rendering.

  Note: This parameter is used in conjunction with **template_vars**. The template filename extension will be used to determine the format-style of the contents, or you can override using **format**.

- **template** (*jinja2.Template*) – A Jinja2 Template object. Same description as *template_path*, except this option you provide the actual Template, rather than a path to the template file.
- **template_vars** (*dict*) – Used in conjunction with the other template options. This parameter contains a dictionary of variables to render into the template.
- **ignore_warning** – A boolean, string or list of string. If the value is True, it will ignore all warnings regardless of the warning message. If the value is a string, it will ignore warning(s) if the message of each warning matches the string. If the value is a list of strings, ignore warning(s) if the message of each warning matches at least one of the strings in the list.

  For example:

```
cu.load(cnf, ignore_warning=True)
cu.load(cnf, ignore_warning='statement not found')
cu.load(cnf, ignore_warning=['statement not found',
                             'statement has no contents; ignored')
```

  Note: When the value of ignore_warning is a string, or list of strings, the string is actually used as a case-insensitive regular expression pattern. If the string contains only alpha-numeric characters, as shown in the above examples, this results in a case-insensitive substring match. However, any regular expression pattern supported by the re library may be used for more complicated match conditions.

- **url** (*str*) – Specify the full pathname of the file that contains the configuration data to load. The value can be a local file path, an FTP location, or a Hypertext Transfer Protocol (HTTP). Refer Doc page for more details.

  For example:

```
cu.load(url="/var/home/user/golden.conf")
cu.load(url="ftp://username@ftp.hostname.net/filename")
cu.load(url="http://username:password@hostname/path/filename")
cu.load(url="/var/home/user/golden.conf", overwrite=True)
```

Returns RPC-reply as XML object.

Raises ConfigLoadError: When errors detected while loading candidate configuration. You can use the Exception errs variable to identify the specific problems.

lock()

Attempts an exclusive lock on the candidate configuration. This is a non-blocking call.

Returns `True` always when successful

Raises ***LockError*** – When the lock cannot be obtained

pdiff(*rb_id=0*)

Helper method that calls `print` on the diff (patch-format) between the current candidate and the provided rollback.

Parameters **rb_id** (*int*) – the rollback id value [0-49]

Returns `None`

rescue(*action*, *format='text'*)

Perform action on the "rescue configuration".

Parameters

- **action** (*str*) – identifies the action as follows:
  - "get" - retrieves/returns the rescue configuration via **format**
  - "save" - saves current configuration as rescue
  - "delete" - removes the rescue configuration
  - "reload" - loads the rescue config as candidate (no-commit)
- **format** (*str*) –

  identifies the return format when action is "get":

  - "text" (default) - ascii-text format
  - "xml" - as XML object

Returns

- When **action** is 'get', then the contents of the rescue configuration is returned in the specified *format*. If there is no rescue configuration saved, then the return value is `None`.
- `True` when **action** is "save".
- `True` when **action** is "delete".

Note: `True` regardless if a rescue configuration exists.

- When **action** is 'reload', return is `True` if a rescue configuration exists, and `False` otherwise.

> **Note:** The rescue configuration is only loaded as the candidate, and not committed. You must commit to make the rescue configuration active.

Raises **ValueError** – If **action** is not one of the above

rollback(*rb_id=0*)
Rollback the candidate config to either the last active or a specific rollback number.

Parameters **rb_id** (*int*) – The rollback id value [0-49], defaults to `0`.

Returns `True` always when successful

Raises **ValueError** – When invalid rollback id is given

unlock()
Unlocks the candidate configuration.

Returns `True` always when successful

Raises ***UnlockError*** – If you attempt to unlock a configuration when you do not own the lock

1.3.2 jnpr.junos.utils.fs

class jnpr.junos.utils.fs.**FS**(*dev*)
Bases: *jnpr.junos.utils.util.Util*

Filesystem (FS) utilities:

- *cat()*: show the contents of a file
- *checksum()*: calculate file checksum (md5,sha256,sha1)
- *cp()*: local file copy (not scp)
- *cwd()*: change working directory
- *ls()*: return file/dir listing
- *mkdir()*: create a directory
- *pwd()*: get working directory
- *mv()*: local file rename
- *rm()*: local file delete
- *rmdir()*: remove a directory
- *stat()*: return file/dir information
- *storage_usage()*: return storage usage
- *directory_usage()*: return directory usage
- *storage_cleanup()*: perform storage storage_cleanup
- ***storage_cleanup_check()*: returns a list of files which will be** removed at cleanup
- *symlink()*: create a symlink
- *tgz()*: tar+gzip a directory

cat(*path*)
Returns the contents of the file **path**.

Parameters **path** (*str*) – File-path

Returns contents of the file (str) or `None` if file does not exist

checksum(*path*, *calc='md5'*)
Performs the checksum command on the given file path using the required calculation method and returns the string value. If the **path** is not found on the device, then `None` is returned.

Parameters

- **path** (*str*) – file-path on local device
- **calc** (*str*) – checksum calculation method:
  - "md5"
  - "sha256"
  - "sha1"

Returns checksum value (str) or `None` if file not found

cp(*from_path*, *to_path*)
Perform a local file copy where **from_path** and **to_path** can be any valid Junos path argument. Refer to the Junos "file copy" command documentation for details.

Parameters

- **from_path** (*str*) – source file-path
- **to_path** (*str*) – destination file-path

Returns `True` if OK, `False` if file does not exist.

cwd(*path*)
Change working directory to **path**.

Parameters **path** (*str*) – path to working directory

directory_usage(*path='.'*, *depth=0*)
Returns the directory usage, similar to the unix "du" command.

Returns dict of directory usage, including subdirectories if depth > 0

ls(*path='.'*, *brief=False*, *followlink=True*)
File listing, returns a dict of file information. If the path is a symlink, then by default **followlink** will recursively call this method to obtain the symlink specific information.

Parameters

- **path** (*str*) – file-path on local device. defaults to current working directory
- **brief** (*bool*) – when `True` brief amount of data
- **followlink** (*bool*) – when `True` (default) this method will recursively follow the directory symlinks to gather data

Returns dict collection of file information or `None` if **path** is not found

mkdir(*path*)
Executes the 'mkdir -p' command on **path**.

> **Warning:** REQUIRES SHELL PRIVILEGES

Returns `True` if OK, error-message (str) otherwise

mv(*from_path*, *to_path*)
Perform a local file rename function, same as "file rename" Junos CLI.

Returns `True` if OK, `False` if file does not exist.

pwd()

Returns The current working directory path (str)

rm(*path*)
Performs a local file delete action, per Junos CLI command "file delete".

Returns `True` when successful, `False` otherwise.

rmdir(*path*)
Executes the 'rmdir' command on **path**.

> **Warning:** REQUIRES SHELL PRIVILEGES

Parameters **path** (*str*) – file-path to directory

Returns `True` if OK, error-message (str) otherwise

stat(*path*)
Returns a dictionary of status information on the path, or `None` if the path does not exist.

Parameters **path** (*str*) – file-path on local device

Returns status information on the file

Return type dict

storage_cleanup()
Perform the 'request system storage cleanup' command to remove files from the filesystem. Return a `dict` of file name/info on the files that were removed.

Returns dict on files that were removed

storage_cleanup_check()
Perform the 'request system storage cleanup dry-run' command to return a `dict` of files/info that would be removed if the cleanup command was executed.

Returns dict of files that would be removed (dry-run)

storage_usage()
Returns the storage usage, similar to the unix "df" command.

Returns dict of storage usage

symlink(*from_path*, *to_path*)
Executes the 'ln -sf **from_path to_path**' command.

> **Warning:** REQUIRES SHELL PRIVILEGES

Returns `True` if OK, or error-message (str) otherwise

tgz (*from_path*, *tgz_path*)
Create a file called **tgz_path** that is the tar-gzip of the given directory specified **from_path**.

Parameters

- **from_path** (*str*) – file-path to directory of files
- **tgz_path** (*str*) – file-path name of tgz file to create

Returns `True` if OK, error-msg (str) otherwise

1.3.3 jnpr.junos.utils.scp

class jnpr.junos.utils.scp.**SCP** (*junos*, ***scpargs*)
Bases: `object`

The SCP utility is used to conjunction with *jnpr.junos.utils.sw.SW* when transferring the Junos image to the device. The *SCP* can be used for other secure-copy use-cases as well; it is implemented to support the python *context-manager* pattern. For example:

```
from jnpr.junos.utils.scp import SCP

with SCP(dev, progress=True) as scp:
    scp.put(package, remote_path)
```

__init__ (*junos*, ***scpargs*)
Constructor that wraps `paramiko` and `scp` objects.

Parameters

- **junos** (`Device`) – the Device object
- **scpargs** (*kvargs*) – any additional args to be passed to paramiko SCP

close()
Closes the ssh/scp connection to the device

open(***scpargs*)
Creates an instance of the scp object and return to caller for use.

Note: This method uses the same username/password authentication credentials as used by *jnpr.junos.device.Device*. It can also use `ssh_private_key_file` option if provided to the *jnpr.junos.device.Device*

Returns SCPClient object

1.3.4 jnpr.junos.utils.start_shell

class jnpr.junos.utils.start_shell.**StartShell** (*nc*, *timeout=30*)
Bases: `object`

Junos shell execution utility. This utility is written to support the "context manager" design pattern. For example:

```
def _ssh_exec(self, command):
    with StartShell(self._dev) as sh:
        got = sh.run(command)
    return got
```

__init__ (*nc*, *timeout=30*)
Utility Constructor

Parameters

- **nc** (Device) – The Device object
- **timeout** (*int*) – Timeout value in seconds to wait for expected string/pattern.

close()
Close the SSH client channel

open()
Open an ssh-client connection and issue the 'start shell' command to drop into the Junos shell (csh). This process opens a `paramiko.SSHClient` instance.

run (*command*, *this='(%|#|\\$)\\s'*, *timeout=0*)
Run a shell command and wait for the response. The return is a tuple. The first item is True/False if exit-code is 0. The second item is the output of the command.

Parameters

- **command** (*str*) – the shell command to execute
- **this** (*str*) – the expected shell-prompt to wait for. If `this` is set to None, function will wait for all the output on the shell till timeout value.
- **timeout** (*int*) – Timeout value in seconds to wait for expected string/pattern (this). If not specified defaults to self.timeout. This timeout is specific to individual run call. If `this` is provided with None value, function will wait till timeout value to grab all the content from command output.

Returns (last_ok, result of the executed shell command (str))

```
with StartShell(dev) as ss:
    print ss.run('cprod -A fpc0 -c "show version"', timeout=10)
```

Note: as a *side-effect* this method will set the `self.last_ok` property. This property is set to `True` if `$?` is "0"; indicating the last shell command was successful else False. If `this` is set to None, last_ok will be set to True if there is any content in result of the executed shell command.

send(*data*)
Send the command **data** followed by a newline character.

Parameters **data** (*str*) – the data to write out onto the shell.

Returns result of SSH channel send

wait_for (*this='(%|#|\\$)\\s'*, *timeout=0*)
Wait for the result of the command, expecting **this** prompt.

Parameters

- **this** (*str*) – expected string/pattern.
- **timeout** (*int*) – Timeout value in seconds to wait for expected string/pattern. If not specified defaults to self.timeout.

Returns resulting string of data in a list

Return type list

> **Warning:** need to add a timeout safeguard

1.3.5 jnpr.junos.utils.sw

class jnpr.junos.utils.sw.**SW**(*dev*)
Bases: *jnpr.junos.utils.util.Util*

Software Utility class, used to perform a software upgrade and associated functions. These methods have been tested on *simple deployments*. Refer to **install** for restricted use-cases for software upgrades.

Primary methods:

- *install()*: perform the entire software installation process
- *reboot()*: reboots the system for the new image to take effect
- *poweroff()*: shutdown the system

Helpers: (Useful as standalone as well)

- *put()*: SCP put package file onto Junos device
- *pkgadd()*: performs the 'request' operation to install the package
- *validate()*: performs the 'request' to validate the package

Miscellaneous:

- rollback: same as 'request software rollback'
- inventory: (property) provides file info for current and rollback images on the device

__init__(*dev*)
Initialize self. See help(type(self)) for accurate signature.

halt(*in_min=0*, *at=None*, *all_re=True*, *other_re=False*)
Perform a system halt, with optional delay (in minutes) or at a specified date and time.

Parameters

- **in_min** (*int*) – time (minutes) before halting the device.
- **at** (*str*) – date and time the halt should take place. The string must match the junos cli reboot syntax
- **all_re** (*bool*) – In case of dual re or VC setup, function by default will halt all. If all is False will only halt connected device
- **other_re** (*str*) – If the system has dual Routing Engines and this option is C(true), then the action is performed on the other REs in the system.

Returns

- rpc response message (string) if command successful

install(*package=None*, *pkg_set=None*, *remote_path='/var/tmp'*, *progress=None*, *validate=False*, *checksum=None*, *cleanfs=True*, *no_copy=False*, *issu=False*, *nssu=False*, *timeout=1800*, *cleanfs_timeout=300*, *checksum_timeout=300*, *checksum_algorithm='md5'*, *force_copy=False*, *all_re=True*, *vmhost=False*, ***kwargs*)
Performs the complete installation of the **package** that includes the following steps:

1. If :package: is a URL, or :no_copy: is True, skip to step 8.
2. computes the checksum of :package: or :pgk_set: on the local host if :checksum: was not provided.

3. performs a storage cleanup on the remote Junos device if :cleanfs: is `True`
4. Attempts to compute the checksum of the :package: filename in the :remote_path: directory of the remote Junos device if the :force_copy: argument is `False`
5. SCP or FTP copies the :package: file from the local host to the :remote_path: directory on the remote Junos device under any of the following conditions:
   a) The :force_copy: argument is `True`
   b) The :package: filename doesn't already exist in the :remote_path: directory of the remote Junos device.
   c) The checksum computed in step 2 does not match the checksum computed in step 4.
6. If step 5 was executed, computes the checksum of the :package: filename in the :remote_path: directory of the remote Junos device.
7. Validates the checksum computed in step 2 matches the checksum computed in step 6.
8. validates the package if :validate: is True
9. installs the package

Warning: This process has been validated on the following deployments.

Tested:

- Single RE devices (EX, QFX, MX, SRX).
- MX dual-RE
- EX virtual-chassis when all same HW model
- QFX virtual-chassis when all same HW model
- QFX/EX mixed virtual-chassis
- Mixed mode VC

Known Restrictions:

- SRX cluster
- MX virtual-chassis

You can get a progress report on this process by providing a **progress** callback.

Note: You will need to invoke the *reboot()* method explicitly to reboot the device.

Parameters

- **package** (*str*) – Either the full file path to the install package tarball on the local (PyEZ host's) filesystem OR a URL (from the target device's perspcective) from which the device retrieves installed. When the value is a URL, then the :no_copy: and :remote_path: values are unused. The acceptable formats for a URL value may be found at: https://www.juniper.net/documentation/en_US/junos/topics/concept/junos-software-formats-filenames-urls.html
- **pkg_set** (*list*) – A list/tuple of :package: values which will be installed on a mixed VC setup.

- **remote_path** (*str*) – If the value of :package: or :pkg_set: is a file path on the local (PyEZ host's) filesystem, then the image is copied from the local filesystem to the :remote_path: directory on the target Junos device. The default is `/var/tmp`. If the value of :package: or :pkg_set: is a URL, then the value of :remote_path: is unused.
- **progress** (*func*) – If provided, this is a callback function with a function prototype given the Device instance and the report string:

```
def myprogress(dev, report):
  print "host: %s, report: %s" % (dev.hostname, report)
```

  If set to `True`, it uses `sw.progress()` for basic reporting by default.
- **validate** (*bool*) – When `True` this method will perform a config validation against the new image
- **checksum** (*str*) – hexdigest of the package file. If this is not provided, then this method will perform the calculation. If you are planning on using the same image for multiple updates, you should consider using the *local_checksum()* method to pre calculate this value and then provide to this method.
- **cleanfs** (*bool*) – When `True` will perform a 'storage cleanup' before copying the file to the device. Default is `True`.
- **no_copy** (*bool*) – When the value of :package: or :pkg_set is not a URL, and the value of :no_copy: is `True` the software package will not be copied to the device and is presumed to already exist on the :remote_path: directory of the target Junos device. When the value of :no_copy: is `False` (the default), then the package is copied from the local PyEZ host to the :remote_path: directory of the target Junos device. If the value of :package: or :pkg_set: is a URL, then the value of :no_copy: is unused.
- **issu** (*bool*) – (Optional) When `True` allows unified in-service software upgrade (ISSU) feature enables you to upgrade between two different Junos OS releases with no disruption on the control plane and with minimal disruption of traffic.
- **nssu** (*bool*) – (Optional) When `True` allows nonstop software upgrade (NSSU) enables you to upgrade the software running on a Juniper Networks EX Series Virtual Chassis or a Juniper Networks EX Series Ethernet Switch with redundant Routing Engines with a single command and minimal disruption to network traffic.
- **timeout** (*int*) – (Optional) The amount of time (seconds) to wait for the :package: installation to complete before declaring an RPC timeout. This argument was added since most of the time the "package add" RPC takes a significant amount of time. The default RPC timeout is 30 seconds. So this :timeout: value will be used in the context of the SW installation process. Defaults to 30 minutes (30*60=1800)
- **cleanfs_timeout** (*int*) – (Optional) Number of seconds (default 300) to wait for the "request system storage cleanup" to complete.
- **checksum_timeout** (*int*) – (Optional) Number of seconds (default 300) to wait for the calculation of the checksum on the remote Junos device.
- **checksum_algorithm** (*str*) – (Optional) The algorithm to use for computing the checksum. Valid values are: 'md5', 'sha1', and 'sha256'. Defaults to 'md5'.
- **force_copy** (*bool*) – (Optional) When `True` perform the copy even if :package: is already present at the :remote_path: directory on the remote Junos device. When `False` (default) if the :package: is already present at the :remote_path:, AND the local checksum matches the remote checksum, then skip the copy to optimize time.

- **all_re** (*bool*) – (Optional) When `True` (default) install the package on all Routing Engines of the Junos device. When `False` perform the software install only on the current Routing Engine.
- **vmhost** (*bool*) – (Optional) A boolean indicating if this is a software update of the vhmhost. The default is `vmhost=False`.
- ****kwargs** (*kwargs*) – (Optional) Additional keyword arguments are passed through to the "package add" RPC.

Returns tuple(<status>, <msg>) * status : `True` when the installation is successful and `False` otherwise * msg : msg received as response or error message created

inventory

Returns dictionary of file listing information for current and rollback Junos install packages. This information comes from the /packages directory.

> **Warning:** Experimental method; may not work on all platforms. If you find this not working, please report issue.

classmethod local_checksum(*package*, *algorithm='md5'*)

Computes the checksum value on the local package file.

Parameters

- **package** (*str*) – File-path to the package (*.tgz) file on the local server
- **algorithm** (*str*) – The algorithm to use for computing the checksum. Valid values are: 'md5', 'sha1', and 'sha256'. Defaults to 'md5'.

Returns checksum (str)

Raises **IOError** – when **package** file does not exist

classmethod local_md5(*package*)

Computes the MD5 checksum value on the local package file.

Parameters **package** (*str*) – File-path to the package (*.tgz) file on the local server

Returns MD5 checksum (str)

Raises **IOError** – when **package** file does not exist

classmethod local_sha1(*package*)

Computes the SHA1 checksum value on the local package file.

Parameters **package** (*str*) – File-path to the package (*.tgz) file on the local server

Returns SHA1 checksum (str)

Raises **IOError** – when **package** file does not exist

classmethod local_sha256(*package*)

Computes the SHA-256 value on the package file.

Parameters **package** (*str*) – File-path to the package (*.tgz) file on the local server

Returns SHA-256 checksum (str)

Raises **IOError** – when **package** file does not exist

pkgadd(*remote_package*, *vmhost=False*, ***kvargs*)

Issue the RPC equivalent of the 'request system software add' command or the 'request vmhost software add' command on the package. If vhmhost=False, the <request-package-add> RPC is used and the The "no-validate" options is set. If you want to validate the image, do that using the specific *validate()* method. If vmhost=True, the <request-vmhost-package-add> RPC is used.

If you want to reboot the device, invoke the *reboot()* method after installing the software rather than passing the `reboot=True` parameter.

Parameters

- **remote_package** (*str*) – The file-path to the install package on the remote (Junos) device.
- **vhmhost** (*bool*) – (Optional) A boolean indicating if this is a software update of the vhmhost. The default is `vmhost=False`.
- **kvargs** (*dict*) – Any additional parameters to the 'request' command can be passed within **kvargs**, following the RPC syntax methodology (dash-2-underscore,etc.)

Warning: Refer to the restrictions listed in *install()*.

pkgaddISSU(*remote_package*, *vmhost=False*, ***kvargs*)

Issue the RPC equivalent of the 'request system software in-service-upgrade' command or the 'request vmhost software in-service-upgrade' command on the package. If vhmhost=False, the <request-package-in-service-upgrade> RPC is used. If vmhost=True, the <request-vmhost-package-in-service-upgrade> RPC is used.

Parameters

- **remote_package** (*str*) – The file-path to the install package on the remote (Junos) device.
- **vmhost** (*bool*) – (Optional) A boolean indicating if this is a software update of the vhmhost. The default is `vmhost=False`.

pkgaddNSSU(*remote_package*, ***kvargs*)

Issue the 'request system software nonstop-upgrade' command on the package.

Parameters **remote_package** (*str*) – The file-path to the install package on the remote (Junos) device.

poweroff(*in_min=0*, *at=None*, *on_node=None*, *all_re=True*, *other_re=False*)

Perform a system shutdown, with optional delay (in minutes) .

If the device is equipped with dual-RE, then both RE will be shut down. This code also handles EX/QFX VC.

Parameters

- **in_min** (*int*) – time (minutes) before shutting down the device.
- **at** (*str*) – date and time the poweroff should take place. The string must match the junos cli poweroff syntax
- **on_node** (*str*) – In case of linux based device, function will by default shutdown the whole device. If any specific node is mentioned, shutdown will be performed on mentioned node
- **all_re** (*bool*) – In case of dual re or VC setup, function by default will shutdown all. If all is False will only shutdown connected device

- **other_re** (*str*) – If the system has dual Routing Engines and this option is C(true), then the action is performed on the other REs in the system.

Returns

- power-off message (string) if command successful

Raises ***RpcError*** – when command is not successful.

Todo: need to better handle the exception event.

---

classmethod progress(*dev*, *report*)
simple progress report function

put(*package*, *remote_path='/var/tmp'*, *progress=None*)
SCP or FTP 'put' the package file from the local server to the remote device.

Parameters

- **package** (*str*) – File path to the package file on the local file system
- **remote_path** (*str*) – The directory on the device where the package will be copied to.
- **progress** (*func*) – Callback function to indicate progress. If set to `True` uses `scp._scp_progress()` for basic reporting by default. See that class method for details.

reboot(*in_min=0*, *at=None*, *all_re=True*, *on_node=None*, *vmhost=False*, *other_re=False*)
Perform a system reboot, with optional delay (in minutes) or at a specified date and time.

If the device is equipped with dual-RE, then both RE will be rebooted. This code also handles EX/QFX VC.

Parameters

- **in_min** (*int*) – time (minutes) before rebooting the device.
- **at** (*str*) – date and time the reboot should take place. The string must match the junos cli reboot syntax
- **all_re** (*bool*) – In case of dual re or VC setup, function by default will reboot all. If all is False will only reboot connected device
- **on_node** (*str*) – In case of linux based device, function will by default reboot the whole device. If any specific node is mentioned, reboot will be performed on mentioned node
- **vmhost** (*bool*) – (Optional) A boolean indicating to run 'request vmhost reboot'. The default is `vmhost=False`.
- **other_re** (*str*) – If the system has dual Routing Engines and this option is C(true), then the action is performed on the other REs in the system.

Returns

- reboot message (string) if command successful

remote_checksum(*remote_package*, *timeout=300*, *algorithm='md5'*)
Computes a checksum of the remote_package file on the remote device.

Parameters

- **remote_package** (*str*) – The file-path on the remote Junos device

- **timeout** (*int*) – The amount of time (seconds) before declaring an RPC timeout. The default RPC timeout is generally around 30 seconds. So this :timeout: value will be used in the context of the checksum process. Defaults to 5 minutes (5*60=300)
- **algorithm** (*str*) – The algorithm to use for computing the checksum. Valid values are: 'md5', 'sha1', and 'sha256'. Defaults to 'md5'.

Returns

- The checksum string
- `None` when the **remote_package** is not found.

Raises ***RpcError*** – RPC errors other than **remote_package** not found.

rollback()
Issues the 'request' command to do the rollback and returns the string output of the results.

Returns Rollback results (str)

safe_copy(*package*, *remote_path='/var/tmp'*, *progress=None*, *cleanfs=True*, *cleanfs_timeout=300*, *checksum=None*, *checksum_timeout=300*, *checksum_algorithm='md5'*, *force_copy=False*)
Copy the install package safely to the remote device. By default this means to clean the filesystem to make space, perform the secure-copy, and then verify the checksum.

Parameters

- **package** (*str*) – file-path to package on local filesystem
- **remote_path** (*str*) – file-path to directory on remote device
- **progress** (*func*) – call-back function for progress updates. If set to `True` uses `sw.progress()` for basic reporting by default.
- **cleanfs** (*bool*) – When `True` (default) perform a "request system storage cleanup" on the device.
- **cleanfs_timeout** (*int*) – Number of seconds (default 300) to wait for the "request system storage cleanup" to complete.
- **checksum** (*str*) – This is the checksum string as computed on the local system. This value will be used to compare the checksum on the remote Junos device.
- **checksum_timeout** (*int*) – Number of seconds (default 300) to wait for the calculation of the checksum on the remote Junos device.
- **checksum_algorithm** (*str*) – The algorithm to use for computing the checksum. Valid values are: 'md5', 'sha1', and 'sha256'. Defaults to 'md5'.
- **force_copy** (*bool*) – When `True` perform the copy even if the package is already present at the remote_path on the device. When `False` (default) if the package is already present at the remote_path, and the local checksum matches the remote checksum, then skip the copy to optimize time.

Returns

- `True` when the copy was successful
- `False` otherwise

validate(*remote_package*, *issu=False*, *nssu=False*, ***kwargs*)
Issues the 'request' operation to validate the package against the config.

Returns

- `True` if validation passes. i.e return code (rc) value is 0
- – `False` otherwise

zeroize(*all_re=False*, *media=None*)
Restore the system (configuration, log files, etc.) to a factory default state. This is the equivalent of the C(request system zeroize) CLI command.

Parameters

- **all_re** (*bool*) – In case of dual re or VC setup, function by default will halt all. If all is False will only halt connected device
- **media** (*str*) – Overwrite media when performing the zeroize operation.

Returns

- rpc response message (string) if command successful

1.3.6 jnpr.junos.utils.util

Junos PyEZ Utility Base Class

class jnpr.junos.utils.util.**Util**(*dev*)
Bases: `object`

Base class for all utility classes

__init__(*dev*)
Initialize self. See help(type(self)) for accurate signature.

dev

Returns the Device object

rpc

Returns Device RPC meta object

1.3.7 jnpr.junos.utils.ftp

FTP utility

class jnpr.junos.utils.ftp.**FTP**(*junos*, ***ftpargs*)
Bases: `ftplib.FTP`

FTP utility can be used to transfer files to and from device.

__init__(*junos*, ***ftpargs*)

Parameters

- **junos** (Device) – Device object
- **ftpargs** (*kvargs*) – any additional args to be passed to ftplib FTP

Supports python *context-manager* pattern. For example:

```
from jnpr.junos.utils.ftp import FTP
with FTP(dev) as ftp:
    ftp.put(package, remote_path)
```

get(*remote_file*, *local_path='/home/docs/checkouts/readthedocs.org/user_builds/junos-pyez/checkouts/2.5.1/docs'*)
This function is used to download file from router to local execution server/shell.

Parameters

- **local_path** – path in which to receive files locally
- **remote_file** – Full path along with filename on the router. If ignored FILE will be copied to "tmp"

Returns True if the transfer succeeds, else False

open()

put(*local_file*, *remote_path=None*)
This function is used to upload file to the router from local execution server/shell.

Parameters

- **local_file** – Full path along with filename which has to be copied to router
- **remote_path** – path in which to receive the files on the remote host. If ignored FILE will be copied to "tmp"

Returns True if the transfer succeeds, else False

1.4 jnpr.junos.device

class jnpr.junos.device.**Device**(**vargs*, ***kvargs*)
Bases: jnpr.junos.device._Connection

Junos Device class.

ON_JUNOS: READ-ONLY - Auto-set to `True` when this code is running on a Junos device, vs. running on a local-server remotely connecting to a device.

auto_probe: When non-zero the call to *open()* will probe for NETCONF reachability before proceeding with the NETCONF session establishment. If you want to enable this behavior by default, you could do the following in your code:

```
from jnpr.junos import Device

# set all device open to auto-probe with timeout of 10 sec
Device.auto_probe = 10

dev = Device( ... )
dev.open()   # this will probe before attempting NETCONF connect
```

__init__(**vargs*, ***kvargs*)
Device object constructor.

Parameters

- **vargs[0]** (*str*) – host-name or ipaddress. This is an alternative for **host**
- **host** (*str*) – **REQUIRED** host-name or ipaddress of target device, unless sock_fd is provided
- **sock_fd** (*str*) – **REQUIRED** file descriptor of an existing socket instead of providing a host. Used for outbound ssh.

- **user** (*str*) – *OPTIONAL* login user-name, uses $USER if not provided
- **passwd** (*str*) – *OPTIONAL* if not provided, assumed ssh-keys are enforced
- **port** (*int*) – *OPTIONAL* NETCONF port (defaults to 830)
- **gather_facts** (*bool*) – *OPTIONAL* For ssh mode default is `True`. In case of console connection over telnet/serial it defaults to `False`. If `False` and old-style fact gathering is in use then facts are not gathered on call to *open()*. This argument is a no-op when new-style fact gathering is in use (the default.)
- **fact_style** (*str*) – *OPTIONAL* The style of fact gathering to use. Valid values are: 'new', 'old', or 'both'. The default is 'new'. The value 'both' is only present for debugging purposes. It will be removed in a future release. The value 'old' is only present to workaround bugs in new-style fact gathering. It will be removed in a future release.
- **mode** (*str*) – *OPTIONAL* mode, mode for console connection (telnet/serial)
- **baud** (*int*) – *OPTIONAL* baud, Used during serial console mode, default baud rate is 9600
- **attempts** (*int*) – *OPTIONAL* attempts, for console connection. default is 10
- **auto_probe** (*bool*) – *OPTIONAL* if non-zero then this enables auto_probe at time of *open()* and defines the amount of time(sec) for the probe timeout
- **ssh_private_key_file** (*str*) – *OPTIONAL* The path to the SSH private key file. This can be used if you need to provide a private key rather than loading the key into the ssh-key-ring/environment. if your ssh-key requires a password, then you must provide it via **passwd**
- **ssh_config** (*str*) – *OPTIONAL* The path to the SSH configuration file. This can be used to load SSH information from a configuration file. By default ~/.ssh/config is queried.
- **normalize** (*bool*) – *OPTIONAL* default is `False`. If `True` then the XML returned by `execute()` will have whitespace normalized
- **use_filter** (*bool*) – *OPTIONAL* To choose between SAX and DOM parsing. default is `False` to use DOM. Select `True` to use SAX (if SAX input is provided).
- **huge_tree** (*bool*) – *OPTIONAL* parse XML with very deep trees and long text content. default is `False`.

close()
Closes the connection to the device only if connected.

connected

open(**vargs*, ***kvargs*)
Opens a connection to the device using existing login/auth information.

Parameters

- **gather_facts** (*bool*) – If set to `True`/`False` will override the device instance value for only this open process
- **auto_probe** (*bool*) – If non-zero then this enables auto_probe and defines the amount of time/seconds for the probe timeout
- **normalize** (*bool*) – If set to `True`/`False` will override the device instance value for only this open process

Returns Device Device instance (*self*).

Raises

- ***ProbeError*** – When **auto_probe** is `True` and the probe activity exceeds the timeout
- ***ConnectAuthError*** – When provided authentication credentials fail to login
- ***ConnectRefusedError*** – When the device does not have NETCONF enabled
- ***ConnectTimeoutError*** – When the the `Device.timeout()` value is exceeded during the attempt to connect to the remote device
- ***ConnectError*** – When an error, other than the above, occurs. The originating `Exception` is assigned as `err._orig` and re-raised to the caller.

transform

Returns the current RPC XML Transformation.

class jnpr.junos.device.**DeviceSessionListener**(*device*)
Bases: ncclient.transport.session.SessionListener

Listens to Session class of Netconf Transport and detects errors in the transport.

__init__(*device*)
Initialize self. See help(type(self)) for accurate signature.

callback(*root*, *raw*)
Required by implementation but not used here.

errback(*ex*)
Called when an error occurs. Set the device's connected status to False. :type ex: `Exception`

1.5 jnpr.junos.exception

exception jnpr.junos.exception.**CommitError**(*rsp*, *cmd=None*, *errs=None*)
Bases: *jnpr.junos.exception.RpcError*

Generated in response to a commit-check or a commit action.

__init__(*rsp*, *cmd=None*, *errs=None*)

Cmd is the rpc command

Rsp is the rpc response (after <rpc-reply>)

Errs is a list of dictionaries of extracted <rpc-error> elements.

Dev is the device rpc was executed on

Timeout is the timeout value of the device

Re is the RE or member exception occured on

exception jnpr.junos.exception.**ConfigLoadError**(*rsp*, *cmd=None*, *errs=None*)
Bases: *jnpr.junos.exception.RpcError*

Generated in response to a failure when loading a configuration.

__init__(*rsp*, *cmd=None*, *errs=None*)

Cmd is the rpc command

Rsp is the rpc response (after <rpc-reply>)

Errs is a list of dictionaries of extracted <rpc-error> elements.

Dev is the device rpc was executed on

Timeout is the timeout value of the device

Re is the RE or member exception occured on

exception jnpr.junos.exception.**ConnectAuthError**(*dev*, *msg=None*)
Bases: *jnpr.junos.exception.ConnectError*

Generated if the user-name, password is invalid

exception jnpr.junos.exception.**ConnectClosedError**(*dev*)
Bases: *jnpr.junos.exception.ConnectError*

Generated if connection unexpectedly closed

__init__(*dev*)
Initialize self. See help(type(self)) for accurate signature.

exception jnpr.junos.exception.**ConnectError**(*dev*, *msg=None*)
Bases: Exception

Parent class for all connection related exceptions

__init__(*dev*, *msg=None*)
Initialize self. See help(type(self)) for accurate signature.

host
login host name/ipaddr

msg
login SSH port

port
login SSH port

user
login user-name

exception jnpr.junos.exception.**ConnectNotMasterError**(*dev*, *msg=None*)
Bases: *jnpr.junos.exception.ConnectError*

Generated if the connection is made to a non-master routing-engine. This could be a backup RE on an MX device, or a virtual-chassis member (linecard), for example

exception jnpr.junos.exception.**ConnectRefusedError**(*dev*, *msg=None*)
Bases: *jnpr.junos.exception.ConnectError*

Generated if the specified host denies the NETCONF; could be that the services is not enabled, or the host has too many connections already.

exception jnpr.junos.exception.**ConnectTimeoutError**(*dev*, *msg=None*)
Bases: *jnpr.junos.exception.ConnectError*

Generated if the NETCONF session fails to connect, could be due to the fact the device is not ip reachable; bad ipaddr or just due to routing

exception jnpr.junos.exception.**ConnectUnknownHostError**(*dev*, *msg=None*)
Bases: *jnpr.junos.exception.ConnectError*

Generated if the specific hostname does not DNS resolve

exception jnpr.junos.exception.**FactLoopError**
Bases: RuntimeError

Generated when there is a loop in fact gathering.

exception jnpr.junos.exception.**JSONLoadError**(*exception*, *rpc_content*)
Bases: Exception

Generated if json content of rpc reply fails to load

__init__(*exception*, *rpc_content*)
Initialize self. See help(type(self)) for accurate signature.

exception jnpr.junos.exception.**LockError**(*rsp*)
Bases: *jnpr.junos.exception.RpcError*

Generated in response to attempting to take an exclusive lock on the configuration database.

__init__(*rsp*)

Cmd is the rpc command

Rsp is the rpc response (after <rpc-reply>)

Errs is a list of dictionaries of extracted <rpc-error> elements.

Dev is the device rpc was executed on

Timeout is the timeout value of the device

Re is the RE or member exception occured on

exception jnpr.junos.exception.**PermissionError**(*rsp*, *cmd=None*, *errs=None*)
Bases: *jnpr.junos.exception.RpcError*

Generated in response to invoking an RPC for which the auth user does not have user-class permissions.

PermissionError.message gives you the specific RPC that cause the exceptions

__init__(*rsp*, *cmd=None*, *errs=None*)

Cmd is the rpc command

Rsp is the rpc response (after <rpc-reply>)

Errs is a list of dictionaries of extracted <rpc-error> elements.

Dev is the device rpc was executed on

Timeout is the timeout value of the device

Re is the RE or member exception occured on

exception jnpr.junos.exception.**ProbeError**(*dev*, *msg=None*)
Bases: *jnpr.junos.exception.ConnectError*

Generated if auto_probe is enabled and the probe action fails

exception jnpr.junos.exception.**RpcError**(*cmd=None*, *rsp=None*, *errs=None*, *dev=None*, *timeout=None*, *re=None*)
Bases: Exception

Parent class for all junos-pyez RPC Exceptions

__init__(*cmd=None*, *rsp=None*, *errs=None*, *dev=None*, *timeout=None*, *re=None*)

Cmd is the rpc command

Rsp is the rpc response (after <rpc-reply>)

Errs is a list of dictionaries of extracted <rpc-error> elements.

Dev is the device rpc was executed on

Timeout is the timeout value of the device

Re is the RE or member exception occured on

exception jnpr.junos.exception.**RpcTimeoutError**(*dev*, *cmd*, *timeout*)

Bases: *jnpr.junos.exception.RpcError*

Generated in response to a RPC execution timeout.

__init__(*dev*, *cmd*, *timeout*)

Cmd is the rpc command

Rsp is the rpc response (after <rpc-reply>)

Errs is a list of dictionaries of extracted <rpc-error> elements.

Dev is the device rpc was executed on

Timeout is the timeout value of the device

Re is the RE or member exception occured on

exception jnpr.junos.exception.**SwRollbackError**(*rsp*, *re=None*)

Bases: *jnpr.junos.exception.RpcError*

Generated in response to a SW rollback error.

__init__(*rsp*, *re=None*)

Cmd is the rpc command

Rsp is the rpc response (after <rpc-reply>)

Errs is a list of dictionaries of extracted <rpc-error> elements.

Dev is the device rpc was executed on

Timeout is the timeout value of the device

Re is the RE or member exception occured on

exception jnpr.junos.exception.**UnlockError**(*rsp*)

Bases: *jnpr.junos.exception.RpcError*

Generated in response to attempting to unlock the configuration database.

__init__(*rsp*)

Cmd is the rpc command

Rsp is the rpc response (after <rpc-reply>)

Errs is a list of dictionaries of extracted <rpc-error> elements.

Dev is the device rpc was executed on

Timeout is the timeout value of the device

Re is the RE or member exception occured on

1.6 jnpr.junos.jxml

jnpr.junos.jxml.**INSERT**(*cmd*)

jnpr.junos.jxml.**NAME**(*name*)

jnpr.junos.jxml.**cscript_conf**(*reply*)

jnpr.junos.jxml.**remove_namespaces**(*xml*)

jnpr.junos.jxml.**remove_namespaces_and_spaces**(*xml*)

jnpr.junos.jxml.**rpc_error**(*rpc_xml*)
extract the various bits from an <rpc-error> element into a dictionary

1.7 jnpr.junos.rpcmeta

class jnpr.junos.rpcmeta.**_RpcMetaExec**(*junos*)
Bases: object

__init__(*junos*)
~PRIVATE CLASS~ creates an RPC meta-executor object bound to the provided ez-netconf :junos: object

cli(*command*, *format='text'*, *normalize=False*)

get(*filter_select=None*, *ignore_warning=False*, ***kwargs*)
Retrieve running configuration and device state information using <get> rpc

```
dev.rpc.get()
dev.rpc.get(ignore_warning=True)
dev.rpc.get(filter_select='bgp') or dev.rpc.get('bgp')
dev.rpc.get(filter_select='bgp/neighbors')
dev.rpc.get("/bgp/neighbors/neighbor[neighbor-address='10.10.0.1']"
            "/timers/state/hold-time")
dev.rpc.get('mpls', ignore_warning=True)
```

Parameters

- **filter_select** (*str*) – The select attribute will be treated as an XPath expression and used to filter the returned data.
- **ignore_warning** – A boolean, string or list of string. If the value is True, it will ignore all warnings regardless of the warning message. If the value is a string, it will ignore warning(s) if the message of each warning matches the string. If the value is a list of strings, ignore warning(s) if the message of each warning matches at least one of the strings in the list.

  For example:

```
dev.rpc.get(ignore_warning=True)
dev.rpc.get(ignore_warning='vrrp subsystem not running')
dev.rpc.get(ignore_warning=['vrrp subsystem not running',
                            'statement not found'])
```

Note: When the value of ignore_warning is a string, or list of strings, the string is actually used as a case-insensitive regular expression pattern. If the string contains only alpha-numeric characters, as shown in the above examples, this results in a case-insensitive substring match. However, any regular expression pattern supported by the re library may be used for more complicated match conditions.

Returns xml object

get_config(*filter_xml=None*, *options={}*, *model=None*, *namespace=None*, *remove_ns=True*, ***kwargs*)

retrieve configuration from the Junos device

```
dev.rpc.get_config()
dev.rpc.get_config(filter_xml='<system><services/></system>')
dev.rpc.get_config(filter_xml='system/services')
dev.rpc.get_config(
    filter_xml=etree.XML('<system><services/></system>'),
    options={'format': 'json'})
# to fetch junos as well as yang model configs
dev.rpc.get_config(model=True)
# openconfig yang example
dev.rpc.get_config(filter_xml='bgp', model='openconfig')
dev.rpc.get_config(filter_xml='<bgp><neighbors></neighbors></bgp>',
                   model='openconfig')
# custom yang example
dev.rpc.get_config(filter_xml='l2vpn', model='custom',
                namespace="http://yang.juniper.net/customyang/l2vpn")
# ietf yang example
dev.rpc.get_config(filter_xml='interfaces', model='ietf')
# ietf-softwire yang example
dev.rpc.get_config(filter_xml='softwire-config', model='ietf',
                     namespace="urn:ietf:params:xml:ns:yang:ietf-softwire",
                     options={'format': 'json'})
```

Filter_xml fully XML formatted tag which defines what to retrieve, when omitted the entire configuration is returned; the following returns the device host-name configured with "set system host-name"

```
config = dev.rpc.get_config(filter_xml=etree.XML('''
    <configuration>
        <system>
            <host-name/>
        </system>
    </configuration>'''))
```

Options is a dictionary of XML attributes to set within the <get-configuration> RPC; the following returns the device host-name either configured with "set system host-name" and if unconfigured, the value inherited from apply-group re0|re1, typical for multi-RE systems

```
config = dev.rpc.get_config(filter_xml=etree.XML('''
                <configuration>
                    <system>
                        <host-name/>
                    </system>
                </configuration>'''),
      options={'database':'committed','inherit':'inherit'})
```

Parameters

- **model** (*str*) – Can provide yang model openconfig/custom/ietf. When model is True and filter_xml is None, xml is enclosed under <data> so that we get junos as well as other model configurations

- **namespace** (*str*) – User can have their own defined namespace in the custom yang models, In such cases they need to provide that namespace so that it can be used to fetch yang modeled configs
- **remove_ns** (*bool*) – remove namespaces, if value assigned is False, function will return xml with namespaces. The same xml returned can be loaded back to devices. This comes handy in case of yang based configs

```
dev.rpc.get_config(filter_xml='bgp', model='openconfig',
             remove_ns=False)
```

load_config(*contents*, *ignore_warning=False*, ***options*)

loads :contents: onto the Junos device, does not commit the change.

Parameters **ignore_warning** – A boolean, string or list of string. If the value is True, it will ignore all warnings regardless of the warning message. If the value is a string, it will ignore warning(s) if the message of each warning matches the string. If the value is a list of strings, ignore warning(s) if the message of each warning matches at least one of the strings in the list.

For example:

```
dev.rpc.load_config(cnf, ignore_warning=True)
dev.rpc.load_config(cnf,
                    ignore_warning='vrrp subsystem not running')
dev.rpc.load_config(cnf,
                    ignore_warning=['vrrp subsystem not running',
                                    'statement not found'])
dev.rpc.load_config(cnf, ignore_warning='statement not found')
```

Note: When the value of ignore_warning is a string, or list of strings, the string is actually used as a case-insensitive regular expression pattern. If the string contains only alphanumeric characters, as shown in the above examples, this results in a case-insensitive substring match. However, any regular expression pattern supported by the re library may be used for more complicated match conditions.

Options is a dictionary of XML attributes to set within the <load-configuration> RPC.

The :contents: are interpreted by the :options: as follows:

format='text' and action='set', then :contents: is a string containing a series of "set" commands

format='text', then :contents: is a string containing Junos configuration in curly-brace/text format

format='json', then :contents: is a string containing Junos configuration in json format

url='path', then :contents: is a None

<otherwise> :contents: is XML structure

CHAPTER 2

Table & View

Tutorial page for PyEZ table and view

2.1 Table & View for Structured output

Understanding Junos PyEZ Tables and Views for structured (xml) output

2.2 Table & View for UnStructured output

PyEZ table/view is extended to parse unstructured data. for example VTY commands and CLI output for which we don't have structured (xml) output and convert them to JSON.

2.2.1 Table Keywords

Table 1: YAML definition to fetch command output from Junos device.

Parameter	Description
Table name	User-defined Table identifier.
command	CLI/VTY command to be executed.
target	(Optional) If the command is vty, provide the targeted fpc.
args	(Optional) Commmand can be a jinja template. args takes key/value pair, value associated with a given key is used to render command template.
key	(Optional) Defines key to be used while forming JSON.
key_items	(Optional) Only get the data for given keys in this list.
item	(Optional) This is used to split whole data in different sections which become the iterable reference for the associated View. If item is '*', parsing is done on whole string blob and not on each line. Item can also be regular expression, where regular expression is used to get key.
title	(Optional) Title help in defining from which section the data should be parsed.
delimiter	(Optional) delimiter to be used to split data of each line and store them as key value pair in the dictionary.
eval	(Optional) Mathematical expression which can be evaluated using python eval function. This can be used on *data* dictionary which is returned by table view.
view	(Optional) View that is used to extract field data from the Table items.

Table name

The Table name is a user-defined identifier for the Table. The YAML file or string can contain one or more Tables. The start of the YAML document must be left justified. For example:

```
---
FPCMemory:
    command: show memory
```

command

Command to be executed on CLI or VTY:

```
# CLI command
---
EthernetSwitchStatistics:
    command: show chassis ethernet-switch statistics

# VTY command to be exectued on target FPC1
---
CMErrorTable:
  command: show cmerror module brief
  target: fpc1
```

target

Target FPC on which given command is to get executed:

```
# VTY command to be exectued on target FPC1
---
FPCMemory:
    command: show memory
    target: fpc1
```

Given FPC target in Table can be overridden through get API:

```
from jnpr.junos.command.fpcmemory import FPCMemory

stats = FPCMemory(dev)
stats = stats.get(target='fpc2')
```

args

CLI/VTY command can take parameter(s). Variable parameters in the command can be Jinja template, dictionary under args is used to render command template:

```
---
XMChipIfdListTable:
  command: show xmchip {{ XM_instance }} ifd list {{ direction }}
  target: fpc1
  args:
    XM_instance: 0
    direction: 0
```

key

The optional key property is a tag or tags that are used to uniquely identify a Table:

```
# Value of task_name key will be taken from the dictionary created from view
---
TaskIOTable:
  command: show task io
  key: task_name
  view: TaskIOView

TaskIOView:
  columns:
    task_name: Task Name
    reads: Reads
    writes: Writes
    rcvd: Rcvd
    sent: Sent
    dropped: Dropped


# Key can be a list also
---
FPCIPV4AddressTable:
  command: show ipv4 address
```

(continues on next page)

(continued from previous page)

```
  target: fpc1
  key:
    - name
    - addr
  view: FPCIPV4AddressView

FPCIPV4AddressView:
  columns:
    index: Index
    addr: Address
    name: Name
```

key_items

The optional key_items property is used to select only certain key data in JSON:

```
---
FPCMemory:
    command: show memory
    target: fpc1
    key: ID
    key_items:
      - 1
      - 2
    view: FPCMemoryView

FPCMemoryView:
    columns:
        id: ID
        base: Base
        total: Total(b)
        free: Free(b)
        used: Used(b)
        perc: "%"
        name: Name
```

Output for **show memory** is:

```
ID        Base      Total(b)       Free(b)       Used(b)   %   Name
--  ----------  ----------  ----------  ----------  ---  -----------
0    4d9ad8e8   1726292636  1514622708   211669928   12  Kernel
1    b47ffb88     67108860    53057404    14051456   20  LAN buffer
2    bcdfffe0     52428784    52428784           0    0  Blob
3    b87ffb88     73400316    73400316           0    0  ISSU scratch
```

Even though we have four ID row here, data returned will be for just 1 & 2 as provided in key_items:

```
{1: {'base': 'b47ffb88',
     'free': 53057404,
     'id': 1,
     'name': 'LAN buffer',
     'perc': 20,
     'total': 67108860,
     'used': 14051456},
 2: {'base': 'bcdfffe0',
```

(continues on next page)

(continued from previous page)

```
    'free': 52428784,
    'id': 2,
    'name': 'Blob',
    'perc': 0,
    'total': 52428784,
    'used': 0}}
```

item

The item value is a string or regular expression to split the output in sections.

Say the out of the command **show devices local** is:

```
TSEC Ethernet Device Driver: .le1, Control 0x4296c218, (1000Mbit)
HW reg base 0xff724000
  [0]: TxBD base 0x7853ce20, RxBD Base 0x7853d640
  [1]: TxBD base 0x7853d860, RxBD Base 0x7853e080
  [2]: TxBD base 0x7853e2a0, RxBD Base 0x785422c0
  [3]: TxBD base 0x785426e0, RxBD Base 0x78544700
Receive:
  185584608 bytes, 2250212 packets, 0 FCS errors, 0 multicast packets
  107271 broadcast packets, 0 control frame packets
  0 PAUSE frame packets, 0 unknown OP codes
  0 alignment errors, 0 frame length errors
  0 code errors, 0 carrier sense errors
  0 undersize packets, 0 oversize packets
  0 fragments, 0 jabbers, 0 drops
Receive per queue:
  [0]: 0 bytes, 0 packets, 0 dropped
       0 jumbo, 0 truncated jumbo
  [1]: 0 bytes, 0 packets, 0 dropped
       0 jumbo, 0 truncated jumbo
  [2]: 0 bytes, 0 packets, 0 dropped
       0 jumbo, 0 truncated jumbo
  [3]: 203586808 bytes, 2250219 packets, 0 dropped
       0 jumbo, 0 truncated jumbo
Transmit:
  288184646 bytes, 2038370 packets, 0 multicast packets
  106531 broadcast packets, 0 PAUSE control frames
  0 deferral packets, 0 excessive deferral packets
  0 single collision packets, 0 multiple collision packets
  0 late collision packets, 0 excessive collision packets
  0 total collisions, 0 drop frames, 0 jabber frames
  0 FCS errors, 0 control frames, 0 oversize frames
  0 undersize frames, 0 fragments frames
Transmit per queue:
  [0]:   10300254 bytes,        72537 packets
                0 dropped,          0 fifo errors
  [1]:    4474302 bytes,       106531 packets
                0 dropped,          0 fifo errors
  [2]:  260203538 bytes,      1857137 packets
                0 dropped,          0 fifo errors
  [3]:     199334 bytes,         2179 packets
                0 dropped,          0 fifo errors
TSEC status counters:
```

(continues on next page)

(continued from previous page)

```
kernel_dropped:0, rx_large:0 rx_short: 0
rx_nonoctet: 0, rx_crcerr: 0, rx_overrun: 0
rx_bsy: 0,rx_babr:0, rx_trunc: 0
rx_length_errors: 0, rx_frame_errors: 0 rx_crc_errors: 0
rx_errors: 0, rx_ints: 2250110, collisions: 0
eberr:0, tx_babt: 0, tx_underrun: 0
tx_timeout: 0, tx_timeout: 0,tx_window_errors: 0
tx_aborted_errors: 0, tx_ints: 2038385, resets: 1


TSEC Ethernet Device Driver: .le3, Control 0x42979220, (1000Mbit)
HW reg base 0xff726000
  [0]: TxBD base 0x78545720, RxBD Base 0x78545f40
  [1]: TxBD base 0x78546160, RxBD Base 0x78546980
  [2]: TxBD base 0x78546ba0, RxBD Base 0x7854abc0
  [3]: TxBD base 0x7854afe0, RxBD Base 0x7854d000
Receive:
  0 bytes, 0 packets, 0 FCS errors, 0 multicast packets
  0 broadcast packets, 0 control frame packets
  0 PAUSE frame packets, 0 unknown OP codes
  0 alignment errors, 0 frame length errors
  0 code errors, 0 carrier sense errors
  0 undersize packets, 0 oversize packets
  0 fragments, 0 jabbers, 0 drops
Receive per queue:
  [0]: 0 bytes, 0 packets, 0 dropped
       0 jumbo, 0 truncated jumbo
  [1]: 0 bytes, 0 packets, 0 dropped
       0 jumbo, 0 truncated jumbo
  [2]: 0 bytes, 0 packets, 0 dropped
       0 jumbo, 0 truncated jumbo
  [3]: 0 bytes, 0 packets, 0 dropped
       0 jumbo, 0 truncated jumbo
Transmit:
  6817984 bytes, 106531 packets, 0 multicast packets
  106531 broadcast packets, 0 PAUSE control frames
  0 deferral packets, 0 excessive deferral packets
  0 single collision packets, 0 multiple collision packets
  0 late collision packets, 0 excessive collision packets
  0 total collisions, 0 drop frames, 0 jabber frames
  0 FCS errors, 0 control frames, 0 oversize frames
  0 undersize frames, 0 fragments frames
Transmit per queue:
  [0]:          0 bytes,            0 packets
                0 dropped,          0 fifo errors
  [1]:    4474302 bytes,       106531 packets
                0 dropped,          0 fifo errors
  [2]:          0 bytes,            0 packets
                0 dropped,          0 fifo errors
  [3]:          0 bytes,            0 packets
                0 dropped,          0 fifo errors
TSEC status counters:
kernel_dropped:0, rx_large:0 rx_short: 0
rx_nonoctet: 0, rx_crcerr: 0, rx_overrun: 0
rx_bsy: 0,rx_babr:0, rx_trunc: 0
rx_length_errors: 0, rx_frame_errors: 0 rx_crc_errors: 0
rx_errors: 0, rx_ints: 0, collisions: 0
```

(continues on next page)

(continued from previous page)

```
eberr:0, tx_babt: 0, tx_underrun: 0
tx_timeout: 0, tx_timeout: 0,tx_window_errors: 0
tx_aborted_errors: 0, tx_ints: 106531, resets: 1
```

And the table to parse above output, item is used to split them into sections.:

```
---
DevicesLocalTable:
  command: show devices local
  target: fpc1
  item: 'TSEC Ethernet Device Driver: (\.?\w+),'
  key: name
  view: DevicesLocalView

DevicesLocalView:
  fields:
    TSEC_status_counters: _TSECStatusCountersTable
    receive_counters: _ReceiveTable
    transmit_per_queue: _TransmitQueueTable
```

key in above table is fetched from the regex group used in item.

item Can also be provided as '*' if we dont want each line matching but from whole output.:

```
_ReceiveTable:
  item: '*'
  title: 'Receive:'
  view: _ReceiveView

_ReceiveView:
  regex:
    bytes: '(\d+) bytes'
    packets: '(\d+) packets'
    FCS_errors: '(\d+) FCS errors'
    broadcast_packets: '(\d+) broadcast packets'
```

title

Title helps in deciding the data to be parsed starting point.:

```
_TSECStatusCountersTable:
  item: '*'
  title: 'TSEC status counters:'
  view: _TSECStatusCountersView

_TSECStatusCountersView:
  regex:
    kernel_dropped: 'kernel_dropped:(\d+)'
    rx_large: 'rx_large:(\d+)'
```

helps to parse data from:

```
TSEC status counters:
kernel_dropped:0, rx_large:0 rx_short: 0
rx_nonoctet: 0, rx_crcerr: 0, rx_overrun: 0
```

(continues on next page)

(continued from previous page)

```
rx_bsy: 0,rx_babr:0, rx_trunc: 0
rx_length_errors: 0, rx_frame_errors: 0 rx_crc_errors: 0
rx_errors: 0, rx_ints: 2250110, collisions: 0
eberr:0, tx_babt: 0, tx_underrun: 0
tx_timeout: 0, tx_timeout: 0,tx_window_errors: 0
tx_aborted_errors: 0, tx_ints: 2038385, resets: 1
```

Note: In above table '*' consider whole data as one paragraph.

delimiter

There are some command output which are just key value pairs. They can be split using given delimiter and converted to JSON. Consider below table:

```
---
FPCLinkStatTable:
    command: show link stats
    target: fpc1
    delimiter: ":"
```

Output for command **show links stats**:

```
PPP LCP/NCP: 0
HDLC keepalives: 0
RSVP: 0
ISIS: 0
OSPF Hello: 539156
OAM:  0
BFD:  15
UBFD:  0
LMI:  0
LACP: 0
ETHOAM: 0
SYNCE:  0
PTP:  0
L2TP:  0
LNS-PPP:  0
ARP:  4292
ELMI:  0
VXLAN MRESOLVE: 0
Unknown protocol: 42
```

Using given delimiter ":" output is parsed to get:

```
{'ARP': 4292, 'ELMI': 0, 'SYNCE': 0, 'ISIS': 0, 'BFD': 15, 'PPP LCP/NCP': 0,
'OAM': 0, 'ETHOAM': 0, 'LACP': 0, 'LMI': 0, 'Unknown protocol': 42,
'UBFD': 0, 'L2TP': 0, 'HDLC keepalives': 0, 'LNS-PPP': 0,
'OSPF Hello': 539156, 'RSVP': 0, 'VXLAN MRESOLVE': 0, 'PTP': 0}
```

eval

Using eval keyword, we can add extra key/value to the final data returned from Table/View. value is evaluated from the Mathematical expression provided by the user. eval expression can use **data** which can be considered as the dictionary returned from table/view. data should be kept under Jinja template so that PyEZ can replace data with dictonary.

Note: For more details about python eval function. check this Link

Examples:

```
---
CChipLiInterruptStatsTable:
  command: show mqss {{ chip_instance }} li interrupt-stats
  target: NULL
  args:
    chip_instance: 0
  key:
    - li_block
    - name
  view: CChipLiInterruptStatsView
  eval:
    cchip_errors_from_lkup_chip: "reduce(lambda x,y: x+y, [v['interrupts'] for k,v in
↪{{ data }}.items()])"

CChipLiInterruptStatsView:
  columns:
    li_block: LI Block
    name: Interrupt Name
    interrupts: Number of Interrupts
    last_occurance: Last Occurrence
```

```
---
CChipLiInterruptStatsTable:
  command: show xmchip {{ chip_instance }} li interrupt-stats
  target: NULL
  args:
    chip_instance: 0
  key:
    - li_block
    - name
  eval:
    cchip_errors_from_lkup_chip: "reduce(lambda x,y: x+y, [v['interrupts'] for k,v in
↪{{ data }}.items()])"
  view: CChipLiInterruptStatsView

CChipLiInterruptStatsView:
  columns:
    li_block: LI Block
    name: Interrupt Name
    interrupts: Number of Interrupts
    last_occurance: Last Occurrence
```

eval can be used to calculate multile values:

```
---
CChipDRDErrTable:
  command: show mqss {{ instance }} drd error-stats
  args:
    instance: 0
  target: NULL
  key: Interrupt Name
  item: '*'
  view: CChipDRDErrView
  eval:
    cchip_drd_wan_errors: sum([v['interrupt_count_wan'] for k, v in {{ data }}.
↪items() if isinstance(v, dict)])
    cchip_drd_fab_errors: sum([v['interrupt_count_fab'] for k, v in {{ data }}.
↪items() if isinstance(v, dict)])

CChipDRDErrView:
  regex:
    cchip_drd_wan_timeouts: 'Total WAN reorder ID timeout errors:\s+(\d+)'
    cchip_drd_fab_timeouts: 'Total fabric reorder ID timeout errors:\s+(\d+)'
  columns:
    interrupt_name: Interrupt Name
    interrupt_count_wan:
      -          Number of
      - Reorder Engine 0
    interrupt_count_fab:
      - Interrupts
      -   Reorder Engine 1
```

```
---
CChipDRDErrTable:
  command: show xmchip {{ instance }} drd error-stats
  args:
    instance: 0
  target: NULL
  key: Interrupt Name
  item: '*'
  view: CChipDRDErrView
  eval:
    cchip_drd_wan_errors: sum([v['interrupt_count'] for k, v in {{ data }}.items() if␣
↪k.endswith('_0')])
    cchip_drd_fab_errors: sum([v['interrupt_count'] for k, v in {{ data }}.items() if␣
↪k.endswith('_1')])

CChipDRDErrView:
  regex:
    cchip_drd_wan_timeouts: 'Total WAN reorder ID timeout errors:\s+(\d+)'
    cchip_drd_fab_timeouts: 'Total fabric reorder ID timeout errors:\s+(\d+)'
  columns:
    interrupt_name: Interrupt Name
    interrupt_count: Number of Interrupts
  filters:
```

(continues on next page)

(continued from previous page)

```
    - interrupt_count
```

view

View is defined how the output from the table to be parsed. Different keyword which can be used with view is defined in next section. Every view will be associated with a table.

Example:

```
---
CMErrorTable:
  command: show cmerror module brief
  target: fpc1
  key: module
  view: CMErrorView

CMErrorView:
  columns:
    module: Module
    name: Name
    errors: Active Errors
```

2.2.2 View Keywords

Junos PyEZ Tables select specific data from the command reply from devices running Junos OS. A Table is associated with a View, which is used to access fields in the Table items. You associate a Table with a particular View by including the view property in the Table definition, which takes the View name as its argument.

A View maps your user-defined field names to string elements in the selected Table items. A View enables you to access specific fields in the output as variables with properties that can be manipulated in Python. Junos PyEZ handles the extraction of the data into JSON for unstructured command output.

Table 2: YAML definition to parse command output

Parameter	Description
columns	(Optional) List of column title as seen in command output
regex	(Optional) List of regular expression to match desired content
fields	(Optional) List of nested tables.
exists	(Optional) If the given statement exists, sets True else False
filters	(Optional) list of column item which should only go to dictionary data

columns

Consider the case where the output of the command is in row/column format. For Example **show lkup-asic wedge-client**:

```
Wedge poll thread state  : 'Started'
Total registered clients : 4
CID         Name PfeInst AscIdx            PPEMask    ZoneMask  RordChk     Mode
------------------------------------------------------------------------------
  0    LUCHIP(0)       0      0 0x0000000000000000 0x000000000000ffff 0x0000f000␣
↪Disabled   NORMAL
```

(continues on next page)

(continued from previous page)

```
  1    LUCHIP(4)       0      1 0x0000000000000000 0x000000000000ffff 0x0000f000␣
↪Disabled   NORMAL
  2    LUCHIP(8)       0      2 0x0000000000000000 0x000000000000ffff 0x0000f000␣
↪Disabled   NORMAL
  3   LUCHIP(12)       0      3 0x0000000000000000 0x000000000000ffff 0x0000f000␣
↪Disabled   NORMAL

Client ID       Curr State    Prev State   Last read
-------------------------------------------------------
    0             NORMAL        NORMAL     6294337620
    1            DISABLED       NORMAL     6294337620
    2            DISABLED       NORMAL     6294337620
    3            DISABLED       NORMAL     6294337620
```

And we want to parse the data of second section consisting of client id, curr state, previous state and last read. We will define table/view with view declares all columns:

```
---
LUChipStatusTable:
  command: show lkup-asic wedge-client
  target: fpc1
  key: Client ID
  view: LUChipStatusView

LUChipStatusView:
  columns:
    client_id: Client ID
    curr_state: Curr State
    prev_state: Prev State
    last_read: Last read
```

Output received will be:

```
{0: {'client_id': 0,
     'curr_state': 'NORMAL',
     'last_read': 6294337620,
     'prev_state': 'NORMAL'},
 1: {'client_id': 1,
     'curr_state': 'DISABLED',
     'last_read': 6294337620,
     'prev_state': 'NORMAL'},
 2: {'client_id': 2,
     'curr_state': 'DISABLED',
     'last_read': 6294337620,
     'prev_state': 'NORMAL'},
 3: {'client_id': 3,
     'curr_state': 'DISABLED',
     'last_read': 6294337620,
     'prev_state': 'NORMAL'}}
```

Note: In columns we need to provide all column title to help parse the data.

There are situation when column title are spread across multiple line. In such cases columns keys element should be list of words corresponding to their columns.

For command `show cmerror module brief` with cli output:

```
------------------------------------------------------------------------
Module  Name               Active Errors  PFE       Callback    ModuleData
                                          Specific  Function
------------------------------------------------------------------------
1       PQ3 Chip           0              Yes       0x00000000  0x00000000
2       Host Loopback      0              No        0x00000000  0x464295b0
3       CM[0]              0              No        0x41f550f0  0x462f767c
```

Table/View to parse above output:

```
---
CMErrorTable:
  command: show cmerror module brief dummy multiline
  target: Null
  key: module
  view: CMErrorView

CMErrorView:
  columns:
    module: Module
    name: Name
    errors: Active Errors
    pfe:
      - PFE
      - Specific
    callback:
      - Callback
      - Function
    data: ModuleData
```

Similarly for command `show mqss 0 fi interrupt-stats` with cli output:

```
FI interrupt statistics
-----------------------

------------------------------------------------------------------------------------------
Stream  Total RLIM  Total       Cell timeout  Total Reorder  Total cell   Total number
        request     PT/MALLOC   Ignored       cell timeout   drops in     of times
        counter     Usemeter                  errors         secure mode  entered into
        saturation  Drops                                                 secure mode
------------------------------------------------------------------------------------------
36      0           0           1             1              0            0
128     0           0           1             49             0            0
142     0           0           1             53             0            0
------------------------------------------------------------------------------------------
------------------------------------------------------------------------
Stream  Stream reconfiguration  Total Error  Total Late  Total CRC Errored
        count due to pointers   Cells        Cells       Packets
        stalled in secure mode
------------------------------------------------------------------------
36      0                       0            1           0
------------------------------------------------------------------------
```

Table/View to parse above output:

```
CChipFiStatsTable:
  command: show mqss {{ chip_instance }} fi interrupt-stats
```

(continues on next page)

(continued from previous page)

```
  target: fpc8
  args:
    chip_instance: 0
  key: Stream
  view: CChipFiStatsView

CChipFiStatsView:
  columns:
    stream: Stream
    req_sat:
      - Total RLIM
      - request
      - counter
      - saturation
    cchip_fi_malloc_drops:
      - Total
      - PT/MALLOC
      - Usemeter
      - Drops
    cell_timeout_ignored:
      - Cell timeout
      - Ignored
    cchip_fi_cell_timeout:
      - Total Reorder
      - cell timeout
      - errors
    drops_in_secure:
      - Total cell
      - drops in
      - secure mode
    times_in_secure:
      - Total number
      - of times
      - entered into
      - secure mode
```

regex

list of regular expression which combined together should match one line. consider command output for `show icmp statistics`

```
ICMP Statistics:
          0 requests
          0 throttled
          0 network unreachables
          0 ttl expired
          0 redirects
          0 mtu exceeded
          0 source route denials
          0 filter prohibited
          0 other unreachables
          0 parameter problems
          0 ttl captured
          0 icmp/option handoffs
          0 igmp v1 handoffs
```

(continues on next page)

(continued from previous page)

```
          0 tag te requests
          0 tag te to RE

ICMP Errors:
          0 unknown unreachables
          0 unsupported ICMP type
          0 unprocessed redirects
          0 invalid ICMP type
          0 invalid protocol
          0 bad input interface
          0 bad route lookup
          0 bad nh lookup
          0 bad cf mtu
          0 runts

ICMP Discards:
          0 multicasts
          0 bad source addresses
          0 bad dest addresses
          0 IP fragments
          0 ICMP errors
          0 unknown originators

ICMP Debug Messages:
          0 throttled

ICMP Rate Limit Settings:
        500 pps per iff
        1000 pps total
```

Here datas are under different title and data is numbers + word(s). Hence the below table/view use regular expression to parse key (numbers) and value (words).

Table/View:

```
---
ICMPStatsTable:
  command: show icmp statistics
  target: fpc1
  view: ICMPStatsView

ICMPStatsView:
  fields:
    discards: _ICMPDiscardsTable
    errors: _ICMPErrorsTable
    rate: _ICMPRateTable

_ICMPDiscardsTable:
  title: ICMP Discards
  key: name
  view: _ICMPDiscardsView

_ICMPDiscardsView:
  regex:
    value: \d+
    name: '(\w+(\s\w+)*)'
```

(continues on next page)

(continued from previous page)

```
_ICMPErrorsTable:
  title: ICMP Errors
  key: name
  view: _ICMPErrorsView

_ICMPErrorsView:
  regex:
    error: numbers
    name: words

_ICMPRateTable:
  title: ICMP Rate Limit Settings
  key: name
  view: _ICMPRateView

_ICMPRateView:
  regex:
    rate: numbers
    name: words
```

Check _ICMPDiscardsTable and _ICMPDiscardsView. *title* (ICMP Discards) under _ICMPDiscardsTable is used to get to the starting point for parsing. value and name are the keys and corresponding regex value is combined to parse each line. Also in such data key cannot be value, so we can select right hand side data *name* as the key.

We also define some inbuilt keywords which can be used in place of regular expression. Check _ICMPErrorsView. Below are the list of inbuilt keywords and corresponding expression used.

- **numbers** = (pp.Word(pp.nums) + pp.Optional(pp.Literal('.') + pp.Word(pp.nums))).setParseAction(lambda i: ''.join(i))
- **hex_numbers** = pp.OneOrMore(pp.Word(pp.nums, min=1)) & pp.OneOrMore(pp.Word('abcdefABCDEF', min=1))
- **word** = pp.Word(pp.alphanums) | pp.Word(pp.alphas)
- **words** = (pp.OneOrMore(word)).setParseAction(lambda i: ' '.join(i))
- **percentage** = pp.Word(pp.nums) + pp.Literal('%')
- **printables** = pp.OneOrMore(pp.Word(pp.printables))

Here **pp** is coming from (import pyparsing as pp)

When table *item* is proved, regex is used on the splitted items. Say when item is * regex is used on whole string blob and not combined to parse each line. For example check the output of command `show ithrottle id 0`:

```
SENT: Ukern command: show ithrottle id 0

ID  Usage %  Cfg State  Oper State     Name
--  -------  ---------  ----------   --------
  0     50.0       1           1      TOE ithrottle

Throttle Times:             In hptime ticks     In ms
                            ---------------     ------
  Timer Interval                     333333     5.000
  Allowed time                       166666     2.500
  Allowed excess                       8333     0.125
  Start time                      488655082     n/a
  Run time this interval                 0     0.000
```

(continues on next page)

(continued from previous page)

```
  Deficit                                    0      0.000
  Run time max                           17712      0.266
  Run time total                 144154525761     2162317

Min Usage Perc:    25.0
Max Usage Perc:    50.0
AdjustUsageEnable: 1

Throttle Stats:
  Starts    : 65708652
  Stops     : 65708652
  Checks    : 124149442
  Enables   : 0
  Disables  : 0
  AdjUp     : 6
  AdjDown   : 4
```

Table/View used for parsing above output:

```
IthrottleIDTable:
  command: show ithrottle id {{ id }}
  args:
    id: 0
  item: '*'
  target: fpc1
  view: IthrottleIDView

IthrottleIDView:
  regex:
    min_usage: 'Min Usage Perc:    (\d+\.\d+)'
    max_usage: 'Max Usage Perc:    (\d+\.\d+)'
    usg_enable: 'AdjustUsageEnable: (\d)'
  fields:
    throttle_stats: _ThrottleStatsTable

_ThrottleStatsTable:
    title: Throttle Stats
    delimiter: ":"
```

Here item is *, hence whole string blob is used to search for given regular expressions.

output:

```
{'max_usage': 50.0,
'min_usage': 25.0,
'throttle_stats': {'AdjDown': 4,
                   'AdjUp': 6,
                   'Checks': 124149442,
                   'Disables': 0,
                   'Enables': 0,
                   'Starts': 65708652,
                   'Stops': 65708652},
'usg_enable': 1}
```

Note: grouping is used to get specific data from regex expression. For example. (d+.d+) is used to get the float value

from string search expression. if we change expression to **Max Usage Perc: (d+).d+** we will get integer part only.

fields

Where the command output has different sections of data which need different logic to parse those subset of data, we can define nested tables under fields section.

For command `show xmchip 0 pt stats` we have 2 section of data:

```
SENT: Ukern command: show xmchip 0 pt stats


WAN PT statistics (Index 0)
---------------------------

PCT entries used by all WI-1 streams         : 0
PCT entries used by all WI-0 streams         : 0
PCT entries used by all LI streams           : 0
CPT entries used by all multicast packets    : 0
CPT entries used by all WI-1 streams         : 0
CPT entries used by all WI-0 streams         : 0
CPT entries used by all LI streams           : 0

Fabric PT statistics (Index 1)
------------------------------

PCT entries used by all FI streams           : 0
PCT entries used by all WI (Unused) streams  : 0
PCT entries used by all LI streams           : 0
CPT entries used by all multicast packets    : 0
CPT entries used by all FI streams           : 0
CPT entries used by all WI (Unused) streams  : 0
CPT entries used by all LI streams           : 0
```

So we defined fields with two nested table each used to parse different subset of data:

```
---
XMChipStatsTable:
  command: show xmchip 0 pt stats
  target: fpc1
  view: XMChipStatsView

XMChipStatsView:
  fields:
    wan_pt_stats: _WANPTStatTable
    fabric_pt_stats: _FabricPTStatTable

_WANPTStatTable:
  title: WAN PT statistics (Index 0)
  delimiter: ":"

_FabricPTStatTable:
  title: Fabric PT statistics (Index 1)
  delimiter: ":"
```

Output:

```
{'fabric_pt_stats': {'CPT entries used by all FI streams': 0,
                     'CPT entries used by all LI streams': 0,
                     'CPT entries used by all WI (Unused) streams': 0,
                     'CPT entries used by all multicast packets': 0,
                     'PCT entries used by all FI streams': 0,
                     'PCT entries used by all LI streams': 0,
                     'PCT entries used by all WI (Unused) streams': 0},
'wan_pt_stats': {'CPT entries used by all LI streams': 0,
                 'CPT entries used by all WI-0 streams': 0,
                 'CPT entries used by all WI-1 streams': 0,
                 'CPT entries used by all multicast packets': 0,
                 'PCT entries used by all LI streams': 0,
                 'PCT entries used by all WI-0 streams': 0,
                 'PCT entries used by all WI-1 streams': 0}}
```

Another example using command output for `show ttp statistics`:

```
SENT: Ukern command: show ttp statistics

TTP Statistics:
                   Receive    Transmit
                ----------  ----------
 L2 Packets            4292     1093544
 L3 Packets          542638           0
 Drops                    0           0
 Netwk Fail               0           0
 Queue Drops              0           0
 Unknown                  0           0
 Coalesce                 0           0
 Coalesce Fail            0           0

TTP Transmit Statistics:
                   Queue 0     Queue 1     Queue 2     Queue 3
                ----------  ----------  ----------  ----------
 L2 Packets        1093544           0           0           0
 L3 Packets              0           0           0           0

TTP Receive Statistics:
                   Control        High      Medium         Low     Discard
                ----------  ----------  ----------  ----------  ----------
 L2 Packets              0           0        4292           0           0
 L3 Packets              0      539172        3466           0           0
 Drops                   0           0           0           0           0
 Queue Drops             0           0           0           0           0
 Unknown                 0           0           0           0           0
 Coalesce                0           0           0           0           0
 Coalesce Fail           0           0           0           0           0

TTP Receive Queue Sizes:
 Control Plane : 0 (max is 4473)
 High          : 0 (max is 4473)
 Medium        : 0 (max is 4473)
 Low           : 0 (max is 2236)

TTP Transmit Queue Size: 0 (max is 6710)
```

Table/View used to parse above output using nested table/view under fields:

```
---
FPCTTPStatsTable:
  command: show ttp statistics
  target: fpc2
  view: FPCTTPStatsView

FPCTTPStatsView:
  fields:
    TTPStatistics: _FPCTTPStatisticsTable
    TTPTransmitStatistics: _FPCTTPTransmitStatisticsTable
    TTPReceiveStatistics: _FPCTTPReceiveStatisticsTable
    TTPQueueSizes: _FPCTTPQueueSizesTable

_FPCTTPStatisticsTable:
  title: TTP Statistics
  view: _FPCTTPStatisticsView

_FPCTTPStatisticsView:
  columns:
    rcvd: Receive
    tras: Transmit

_FPCTTPTransmitStatisticsTable:
  title: TTP Transmit Statistics
  view: _FPCTTPTransmitStatisticsView

_FPCTTPTransmitStatisticsView:
  columns:
    queue0: Queue 0
    queue1: Queue 1
    queue2: Queue 2
    queue3: Queue 3
  filters:
    - queue2

_FPCTTPReceiveStatisticsTable:
  title: TTP Receive Statistics
  key: name
  key_items:
    - Coalesce
  view: _FPCTTPReceiveStatisticsView

_FPCTTPReceiveStatisticsView:
  columns:
    control: Control
    high: High
    medium: Medium
    low: Low
    discard: Discard

_FPCTTPQueueSizesTable:
  title: TTP Receive Queue Sizes
  delimiter: ":"
```

Output:

```
{'TTPQueueSizes': {'Control Plane': '0 (max is 4473)',
```

(continues on next page)

(continued from previous page)

```
                  'High': '0 (max is 4473)',
                  'Low': '0 (max is 2236)',
                  'Medium': '0 (max is 4473)'},
 'TTPReceiveStatistics': {'Coalesce': {'control': 0,
                                       'discard': 0,
                                       'high': 0,
                                       'low': 0,
                                       'medium': 0,
                                       'name': 'Coalesce'}},
 'TTPStatistics': {'Coalesce': {'name': 'Coalesce', 'rcvd': 0, 'tras': 0},
                   'Coalesce Fail': {'name': 'Coalesce Fail',
                                     'rcvd': 0,
                                     'tras': 0},
                   'Drops': {'name': 'Drops', 'rcvd': 0, 'tras': 0},
                   'L2 Packets': {'name': 'L2 Packets',
                                  'rcvd': 0,
                                  'tras': 7468},
                   'L3 Packets': {'name': 'L3 Packets', 'rcvd': 0, 'tras': 0},
                   'Netwk Fail': {'name': 'Netwk Fail',
                                  'rcvd': 0,
                                  'tras': 173},
                   'Queue Drops': {'name': 'Queue Drops',
                                   'rcvd': 0,
                                   'tras': 0},
                   'Unknown': {'name': 'Unknown', 'rcvd': 0, 'tras': 0}},
 'TTPTransmitStatistics': {'L2 Packets': {'queue2': 0},
                           'L3 Packets': {'queue2': 0}}}
```

Note: fields in unstructured table/view is different compared to rpc based table/view. For RPC, fields are xpath tags.

exists

If we just need to check if the given string statement is present or not in the command output. we can use `exists` option. For example

For command output of `show host_loopback status-summary`:

```
SENT: Ukern command: show host_loopback status-summary

Host Loopback Toolkit Status Summary:

No detected wedges

No toolkit errors
```

Table/View defined:

```
---
HostlbStatusSummaryTable:
  command: show host_loopback status-summary
  target: fpc1
  view: HostlbStatusSummaryView
```

(continues on next page)

(continued from previous page)

```
HostlbStatusSummaryView:
  exists:
    no_detected_wedges: No detected wedges
    no_toolkit_errors: No toolkit errors
```

Output:

```
{'no_detected_wedges': True, 'no_toolkit_errors': True}
```

filters

When we are interested in only few keys from the parsed output, we can filter filter out the desired key/value using filters. Filters takes a list of keys from columns items. Final output dictionary should only consist of items listed in filters per iteration of view output.

Consider below table/view:

```
---
CMErrorTable:
  command: show cmerror module brief
  target: fpc1
  key:
    - module
  view: CMErrorView

CMErrorView:
  columns:
    module: Module
    name: Name
    errors: Active Errors
  filters:
    - errors
```

Output from command `show cmerror module brief`:

```
---------------------------------------
Module  Name               Active Errors
---------------------------------------
1       PQ3 Chip           0
2       Host Loopback      0
3       CM[0]              0
4       LUCHIP(0)          0
5       TOE-LU-0:0:0       0
6       LUCHIP(4)          0
7       TOE-LU-0:1:0       0
8       LUCHIP(8)          0
9       TOE-LU-0:2:0       0
10      LUCHIP(12)         0
11      TOE-LU-0:3:0       0
12      XMCHIP(0)          0
13      TOE-XM-0:0:0       0
14      MPC                0
15      GE Switch          0
16      PMB                0
17      JNH                0
```

(continues on next page)

(continued from previous page)

```
18      PRECL:0:XM:0      0
19      PRECL:1:XM:0      0
```

Output:

```
{1: {'errors': 0}, 2: {'errors': 0}, 3: {'errors': 0}, 4: {'errors': 0},
5: {'errors': 0}, 6: {'errors': 0}, 7: {'errors': 0}, 8: {'errors': 0},
9: {'errors': 0}, 10: {'errors': 0}, 11: {'errors': 0}, 12: {'errors': 0},
13: {'errors': 0}, 14: {'errors': 0}, 15: {'errors': 0}, 16: {'errors': 0},
17: {'errors': 0}, 18: {'errors': 0}, 19: {'errors': 0}}
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

j

Index

Symbols

A

C

D

E

F

G

H

I

J

T

U

V

W

X

Z