Junos PyEZ Documentation

Release 1.2.1

Juniper Networks, Inc.

June 30, 2015

Contents

Contents:

CHAPTER 1

jnpr.junos

1.1 jnpr.junos.cfg

1.1.1 jnpr.junos.cfg.phyport

jnpr.junos.cfg.phyport.base

class jnpr.junos.cfg.phyport.base.**PhyPortBase**(*junos*, *namevar=None*, ***kvargs*)
Bases: *jnpr.junos.cfg.resource.Resource*

[edit interfaces <name>]

Resource name: str <name> is the interface-name (IFD), e.g. 'ge-0/0/0'

PORT_DUPLEX = {'full': 'full-duplex', 'half': 'half-duplex'}

PROPERTIES = ['admin', 'description', 'speed', 'duplex', 'mtu', 'loopback', '$unit_count']

jnpr.junos.cfg.phyport.classic

class jnpr.junos.cfg.phyport.classic.**PhyPortClassic**(*junos*, *namevar=None*, ***kvargs*)
Bases: *jnpr.junos.cfg.phyport.base.PhyPortBase*

jnpr.junos.cfg.phyport.switch

class jnpr.junos.cfg.phyport.switch.**PhyPortSwitch**(*junos*, *namevar=None*, ***kvargs*)
Bases: *jnpr.junos.cfg.phyport.base.PhyPortBase*

PORT_SPEED = {'auto': 'auto-negotiation', '100m': 'ethernet-100m', '1g': 'ethernet-1g', '10m': 'ethernet-10m'}

Module contents

class jnpr.junos.cfg.phyport.**PhyPort**
Bases: object

1.1.2 jnpr.junos.cfg.resource

class jnpr.junos.cfg.resource.**Resource**(*junos*, *namevar=None*, ***kvargs*)
Bases: object

D
returns the Device object bound to this resource/manager

M
returns the :Resource: manager associated to this resource

P
returns the parent of the associated Junos object

PROPERTIES = ['_exists', '_active']

R
returns the Device RPC meta object

__init__(*junos*, *namevar=None*, ***kvargs*)
Resource or Resource-Manager constructor. All managed resources and resource-managers inherit from this class.

junos Instance of Device, this is bound to the Resource for device access

namevar If not None, identifies a specific resource by 'name'. The format of the name is resource dependent. Most resources take a single string name, while others use tuples for compound names. refer to each resource for the 'namevar' definition

If namevar is None, then the instance is a Resource-Manager (RM). The RM is then used to select specific resources by name using the __getitem__ overload.

kvargs['P'] or kvargs['parent'] Instance to the resource parent. This is set when resources have hierarchical relationships, like rules within rulesets

kvargs['M'] Instance to the resource manager.

activate()
activate resource in Junos config the same as the Junos config-mode "activate" command

active
is this resource configuration active on the Junos device?

RuntimeError if invoked on a manager object

catalog
returns a dictionary of resources

catalog_refresh()
reloads the resource catalog from the Junos device

classmethod copyifexists(*klass*, *xml*, *ele_name*, *to_py*, *py_name=None*)

deactivate()
activate resource in Junos config the same as the Junos config-mode "deactivate" command

delete()
remove configuration from Junos device the same as the Junos config-mode "delete" command

classmethod diff_list(*klass*, *has_list*, *should_list*)

exists
does this resource configuration exist on the Junos device?

RuntimError if invoked on a manager

is_mgr
is this a resource manager?

is_new
is this a new resource? that is, it does not exist on the Junos device when it was initally retrieved

RuntimeError if invoked on a manager

list
returns a list of named resources

list_refresh()
reloads the managed resource list from the Junos device

manages
a resource may contain sub-managers for hierarchical oriented resources. this method will return a list of manager names attached to this resource, or :None: if there are not any

name
the name of the resource

RuntimeError if invoked on a manager

propcopy(*p_name*)
proptery from :has: to :should:

performs a 'deepcopy' of the property; used to make changes to list, dict type properties

read()
read resource configuration from device

refresh()

rename(*new_name*)
rename resource in Junos configuration the same as the Junos config-mode "rename" command

reorder(***kvargs*)
move the configuration within the Junos hierarcy the same as the Junos config-mode "insert" command

Kvargs after="<name>" before="<name>"

write(***kvargs*)
write resource configuration stored in :should: back to device

kvargs['touch'] if True, then write() will skip the check to see if any items exist in :should:

xml
for debugging the resource XML configuration that was read from the Junos device

classmethod xml_set_or_delete(*klass*, *xml*, *ele_name*, *value*)
HELPER function to either set a value or remove the element

classmethod xmltag_set_or_del(*klass*, *ele_name*, *value*)
HELPER function creates an XML element tag read-only that includes the DEL attribute depending on :value:

1.1.3 jnpr.junos.cfg.user

class jnpr.junos.cfg.user.**User**(*junos*, *namevar=None*, ***kvargs*)
Bases: *jnpr.junos.cfg.resource.Resource*

[edit system login user <name>]

Resource name: str <name> is the user login name

Manages resources: sshkey, UserSSHKey

MANAGES = {'sshkey': <class 'jnpr.junos.cfg.user_ssh_key.UserSSHKey'>}

PROPERTIES = ['uid', 'fullname', 'userclass', 'password', '$password', '$sshkeys']

1.1.4 jnpr.junos.cfg.user_ssh_key

class jnpr.junos.cfg.user_ssh_key.**UserSSHKey**(*junos*, *namevar=None*, ***kvargs*)
Bases: *jnpr.junos.cfg.resource.Resource*

[edit system login user <name> authentication <key-type> <key-value>]

Resource name: tuple(<key-type>, <key-value>) <key-type> : ['ssh-dsa', 'ssh-rsa'] <key-value> : SSH public key string (usually something very long)

Resource manager utilities: load_key - allows you to load an ssh-key from a file or str

PROPERTIES = []

load_key(*path=None*, *key_value=None*)
Adds a new ssh-key to the user authentication. You can provide either the path to the ssh-key file, or the contents of they key (useful for loading the same key on many devices)

Path (optional) path to public ssh-key file on the local server,

Key_value (optional) the contents of the ssh public key

1.2 jnpr.junos.factory

1.2.1 jnpr.junos.factory.cfgtable

class jnpr.junos.factory.cfgtable.**CfgTable**(*dev=None*, *xml=None*, *path=None*)
Bases: *jnpr.junos.factory.table.Table*

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

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.

keys_required
True/False - if this Table requires keys

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

1.2.2 jnpr.junos.factory.factory_cls

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*)

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

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.

1.2.3 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__()

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

1.2.4 jnpr.junos.factory.optable

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

1.2.5 jnpr.junos.factory.table

class jnpr.junos.factory.table.**Table**(*dev=None*, *xml=None*, *path=None*)

Bases: object

D

the Device instance

ITEM_NAME_XPATH = 'name'

ITEM_XPATH = None

RPC

the Device.rpc instance

VIEW = None

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

Dev Device instance

Xml lxml Element instance

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

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 strftime()

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

1.2.6 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

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(**args*, ***kwds*)
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

1.2.7 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__()

astype(*name*, *xpath=None*, *astype=<type '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
~~~

1.2.8 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__()

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

1.3 jnpr.junos.facts

1.3.1 jnpr.junos.facts.chassis

jnpr.junos.facts.chassis.**facts_chassis**(*junos*, *facts*)

The following facts are assigned: facts['2RE'] : designates if the device can support two RE, not that it has them facts['RE_hw_mi'] : designates if the device is multi-instance-routing-engine facts['model'] : product model facts['serialnumber'] : serial number

NOTES:

1. if in a 2RE system, this routine will only load the information from the first chassis item.
2. hostname, domain, and fqdn are retrieved from configuration data; inherited configs are checked.

1.3.2 jnpr.junos.facts.domain

jnpr.junos.facts.domain.**facts_domain**(*junos*, *facts*)

The following facts are required: facts['hostname']

The following facts are assigned: facts['domain'] facts['fqdn']

1.3.3 jnpr.junos.facts.ifd_style

jnpr.junos.facts.ifd_style.**facts_ifd_style**(*junos*, *facts*)

1.3.4 jnpr.junos.facts.personality

jnpr.junos.facts.personality.**facts_personality**(*junos*, *facts*)

1.3.5 jnpr.junos.facts.routing_engines

jnpr.junos.facts.routing_engines.**facts_routing_engines**(*junos*, *facts*)

1.3.6 jnpr.junos.facts.session

facts['HOME'] = login home directory

jnpr.junos.facts.session.**facts_session**(*dev*, *facts*)

1.3.7 jnpr.junos.facts.srx_cluster

jnpr.junos.facts.srx_cluster.**facts_srx_cluster**(*junos*, *facts*)

1.3.8 jnpr.junos.facts.switch_style

jnpr.junos.facts.switch_style.**facts_switch_style**(*junos*, *facts*)

1.3.9 jnpr.junos.facts.swver

jnpr.junos.facts.swver.**facts_software_version**(*junos*, *facts*)

The following facts are required: facts['master']

The following facts are assigned: facts['hostname'] facts['version'] facts['version_<RE#>'] for each RE in dual-RE, cluster or VC system facts['version_info'] for master RE

class jnpr.junos.facts.swver.**version_info**(*verstr*)

Bases: object

__init__(*verstr*)

verstr - version string

jnpr.junos.facts.swver.**version_yaml_representer**(*dumper*, *version*)

1.4 jnpr.junos.op

1.4.1 jnpr.junos.op.arp

Pythonifier for ARP Table/View

1.4.2 jnpr.junos.op.bfd

Pythonifier for BFD Table/View

1.4.3 jnpr.junos.op.ethport

Pythonifier for EthPort Table/View

1.4.4 jnpr.junos.op.isis

Pythonifier for ISIS Table/View

1.4.5 jnpr.junos.op.lacp

Pythonifier for LACP Table/View

1.4.6 jnpr.junos.op.ldp

Pythonifier for LDP Table/View

1.4.7 jnpr.junos.op.lldp

Pythonifier for LLDP Table/View

1.4.8 jnpr.junos.op.phyport

Pythonifier for PhyPort Table/View

1.4.9 jnpr.junos.op.routes

Pythonifier for Route Table/View

1.4.10 jnpr.junos.op.xcvr

Pythonifier for Xcvr Table/View

1.5 jnpr.junos.utils

1.5.1 jnpr.junos.utils.config

class jnpr.junos.utils.config.**Config**(*dev*)

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

Overivew 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

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

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()

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.

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*)

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

Parameters **rollback** (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), 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

  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'
- **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.

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) – identifes 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** (str) – 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

1.5.2 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
- *storage_cleanup()*: perform storage storage_cleanup
- *storage_cleanup_check()*: returns a list of files to remove 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

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

1.5.3 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=_scp_progress ) as scp:
    scp.put( package, remote_path )
```

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

Warning: The *jnpr.junos.device.Device* `ssh_private_key_file` option is currently **not** supported.

Todo

add support for `ssh_private_key_file`.

Returns SCPClient object

1.5.4 jnpr.junos.utils.start_shell

class `jnpr.junos.utils.start_shell.`**StartShell**(*nc*)
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)
        ok = sh.last_ok
    return (ok, got)
```

__init__(*nc*)
Utility Constructor

Parameters **nc** (Device) – The Device object

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='% '*)
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 exected shell-prompt to wait for

Returns result of the executed shell command (str)

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.

---

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='% '*)
Wait for the result of the command, expecting **this** prompt.

Parameters **this** (str) – expected string/pattern.

Returns resulting string of data

Return type *str*

Warning: need to add a timeout safeguard

1.5.5 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*)

install(*package=None*, *pkg_set=None*, *remote_path='/var/tmp'*, *progress=None*, *validate=False*, *checksum=None*, *cleanfs=True*, *no_copy=False*, *timeout=1800*, ***kwargs*)
Performs the complete installation of the **package** that includes the following steps:

1.computes the local MD5 checksum if not provided in :checksum:

2.performs a storage cleanup if :cleanfs: is True

3.SCP copies the package to the :remote_path: directory

4.computes remote MD5 checksum and matches it to the local value

5.validates the package if :validate: is True

6.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

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) – The file-path to the install package tarball on the local filesystem
- **pkg_set** (list) – The file-paths as list/tuple of the install package tarballs on the local filesystem which will be installed on mixed VC setup.
- **remote_path** (str) – The directory on the Junos device where the package file will be SCP'd to or where the package is stored on the device; the default is `/var/tmp`.
- **validate** (*bool*) – When `True` this method will perform a config validation against the new image
- **checksum** (str) – MD5 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_md5()* method to pre calculate this value and then provide to this method.
- **cleanfs** (*bool*) – When `True` will perform a 'storeage cleanup' before SCP'ing the file to the device. Default is `True`.
- **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)
```

- **no_copy** (*bool*) – When `True` the software package will not be SCP'd to the device. Default is `False`.
- **timeout** (int) – The amount of time (seconds) 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 generally around 30 seconds. So this :timeout: value will be used in the context of the SW installation process. Defaults to 30 minutes (30*60=1800)
- **force_host** (*bool*) – (Optional) Force the addition of host software package or bundle (ignore warnings) on the QFX5100 device.

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_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*, ***kvargs*)
Issue the 'request system software add' command on the package. The "no-validate" options is set by default. If you want to validate the image, do that using the specific *validate()* method. Also, if you want to reboot the device, suggest using the *reboot()* method rather `reboot=True`.

Parameters

- **remote_package** (str) – The file-path to the install package on the remote (Junos) device.
- **kvargs** (*dict*) – Any additional parameters to the 'request' command can be passed within **kvargs**, following the RPC syntax methodology (dash-2-underscore,etc.)

Todo

Add way to notify user why installation failed.

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

poweroff (*in_min=0*)
Perform a system shutdown, with optional delay (in minutes) .

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.

Returns

- reboot 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 '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. You can use *SW.progress()* for basic reporting. See that class method for details.

reboot(*in_min=0*, *at=None*)
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

Returns

- reboot message (string) if command successful

Raises RpcError when command is not successful.

Todo

need to better handle the exception event.

---

remote_checksum(*remote_package*, *timeout=300*)
Computes the MD5 checksum 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)

Returns

- The MD5 checksum string
- `False` 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*, ***kvargs*)
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 MD5 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
- **cleanfs** (*bool*) – When `True` (default) this method will perform the "storage cleanup" on the device.
- **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.

Returns

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

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

Returns

- `True` if validation passes
- error (str) otherwise

1.5.6 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*)

dev

Returns the Device object

rpc

Returns Device RPC meta object

1.6 jnpr.junos.device

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

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
```

ON_JUNOS = False

Template (*filename*, *parent=None*, *gvars=None*)
Used to return a Jinja2 *Template*.

Parameters **filename** (str) – file-path to Jinja2 template file on local device

Returns Jinja2 *Template* give **filename**.

__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
- **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* default is `True`. If `False` then the facts are not gathered on call to *open()*
- **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

auto_probe = 0

bind (**vargs*, ***kvargs*)
Used to attach things to this Device instance and make them a property of the :class:Device instance. The most common use for bind is attaching Utility instances to a :class:Device. For example:

```
from jnpr.junos.utils.config import Config

dev.bind( cu=Config )
dev.cu.lock()
# ... load some changes
dev.cu.commit()
dev.cu.unlock()
```

Parameters

- **vargs** (list) – A list of functions that will get bound as instance methods to this Device instance.

  Warning: Experimental.

- **new_property** – name/class pairs that will create resource-managers bound as instance attributes to this Device instance. See code example above

cli(*command*, *format='text'*, *warning=True*)
Executes the CLI command and returns the CLI text output by default.

Parameters

- **command** (str) – The CLI command to execute, e.g. "show version"
- **format** (str) – The return format, by default is text. You can optionally select "xml" to return the XML structure.

Note: You can also use this method to obtain the XML RPC command for a given CLI command by using the pipe filter `| display xml rpc`. When you do this, the return value is the XML RPC command. For example if you provide as the command `show version | display xml rpc`, you will get back the XML Element `<get-software-information>`.

Warning: This function is provided for **DEBUG** purposes only! **DO NOT** use this method for general automation purposes as that puts you in the realm of "screen-scraping the CLI". The purpose of the PyEZ framework is to migrate away from that tooling pattern. Interaction with the device should be done via the RPC function.

Warning: You cannot use "pipe" filters with **command** such as `| match` or `| count`, etc. The only value use of the "pipe" is for the `| display xml rpc` as noted above.

close()
Closes the connection to the device.

display_xml_rpc(*command*, *format='xml'*)
Executes the CLI command and returns the CLI text output by default.

Parameters

- **command** (str) – The CLI command to retrieve XML RPC for, e.g. "show version"
- **format** (str) – The return format, by default is XML. You can optionally select "text" to return the XML structure as a string.

execute(**args*, ***kwargs*)
Executes an XML RPC and returns results as either XML or native python

Parameters

- **rpc_cmd** – can either be an XML Element or xml-as-string. In either case the command starts with the specific command element, i.e., not the <rpc> element itself
- **to_py'** (*func*) – Is a caller provided function that takes the response and will convert the results to native python types. all kvargs will be passed to this function as well in the form:

```
to_py( self, rpc_rsp, **kvargs )
```

Raises

- **ValueError** – When the **rpc_cmd** is of unknown origin
- **PermissionError** – When the requested RPC command is not allowed due to user-auth class privilege controls on Junos
- **RpcError** – When an `rpc-error` element is contained in the RPC-reply

Returns RPC-reply as XML object. If **to_py** is provided, then that function is called, and return of that function is provided back to the caller; presumably to convert the XML to native python data-types (e.g. `dict`).

facts

Returns Device fact dictionary

facts_refresh()
Reload the facts from the Junos device into *facts* property.

hostname

Returns the host-name of the Junos device.

logfile

Returns exsiting logfile `file` object.

manages

Returns `list` of Resource Managers/Utilities attached to this instance using the *bind()* method.

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.

password

Returns `None` - do not provide the password

probe(*timeout=5*, *intvtimeout=1*)

Probe the device to determine if the Device can accept a remote connection. This method is meant to be called *prior* to :open(): This method will not work with ssh-jumphost environments.

Parameters

- **timeout** (int) – The probe will report `True`/`False` if the device report connectivity within this timeout (seconds)
- **intvtimeout** (int) – Timeout interval on the socket connection. Generally you should not change this value, but you can if you want to twiddle the frequency of the socket attempts on the connection

Returns `True` if probe is successful, `False` otherwise

timeout

Returns the current RPC timeout value (int) in seconds.

transform

Returns the current RPC XML Transformation.

user

Returns the login user (str) accessing the Junos device

1.7 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*)

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*)

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*)

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

Parent class for all connection related exceptions

__init__(*dev*, *msg=None*)

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

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

__init__(*rsp*)

exception jnpr.junos.exception.**PermissionError**(*rsp*, *cmd=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*)

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: exceptions.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 <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*)

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*)

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

Generated in response to attempting to unlock the configuration database.

__init__(*rsp*)

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

1.9 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'*)

get_config(*filter_xml=None*, *options={}*)
retrieve configuration from the Junos device

Filter_xml is options, defines what to retrieve. if omitted then the entire configuration is returned

Options is a dict, creates attributes for the RPC

load_config(*contents*, ***options*)

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

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

<otherwise> :contents: is XML structure

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

j

Symbols

A

B

C

K

L

M

N

O

P

R

S

T

U

V

W

X