
CCUtils Documentation

Release 0.2.18-beta

ccutils

Jul 09, 2021

Contents

1	CC Parser	1
1.1	ConfigParser	1
1.1.1	Quick Start	1
1.2	BaseConfigParser	1
1.2.1	Quick Start	1
1.3	CiscoIosParser	4
1.3.1	Quick Start	4
1.4	BaseConfigLine	9
1.5	BaseInterfaceLine	12
1.6	CiscoIosInterfaceLine	18
1.7	ConfigToJson	25
1.8	ConfigMigration	26
2	CC Templater	27
2.1	CC Templater	27
3	Templates	29
3.1	Placeholder	29
4	Utils	31
4.1	Common Utils	31
4.2	CiscoRange	32
5	Indices and tables	33
	Python Module Index	35
	Index	37

1.1 ConfigParser

1.1.1 Quick Start

```
from ccutils.ccparser import ConfigParser
import pathlib

# Optinal - Create pathlib object pointing to your config file
path_to_file = pathlib.Path("/path/to/config_file.txt")

config = ConfigParser(config=path_to_file, device_type="ios")

# Print number of config lines
print(len(config.lines))
```

1.2 BaseConfigParser

1.2.1 Quick Start

```
from ccutils.ccparser import BaseConfigParser
import pathlib

# Optinal - Create pathlib object pointing to your config file
path_to_file = pathlib.Path("/path/to/config_file.txt")

config = BaseConfigParser(config=path_to_file)

# Print number of config lines
print(len(config.lines))
```

class BaseConfigParser (*config=None, verbosity=4, name='BaseConfigParser', **kwargs*)

Bases: object

Base Configuration Parser object used for loading configuration files.

Base class for parsing Cisco-like configs

Parameters

- **config** (*pathlib.Path* or *str* or *list*) – Config file in a form of *pathlib.Path*, or *string* containing the entire config or list of lines of the config file
- **verbosity** (*int*, optional) – Determines the verbosity of logging output, defaults to 4: Info

lines

Contains list of all config lines stored as objects (see *ccutils.ccparser.BaseConfigLine*)

Type list

config_lines_str

Contains list of all config lines stored as strings

Type list

Examples

Possible config inputs:

```
# Using pathlib
config_file = pathlib.Path("/path/to/config_file.txt")
config = BaseConfigParser(config=config_file)

# Using string
config_string = '''
hostname RouterA
!
interface Ethernet0/0
 description Test Interface
 ip address 10.0.0.1 255.255.255.0
!
end
'''
config = BaseConfigParser(config=config_string)

# Using list
config_list = [
"hostname RouterA",
"! ",
"interface Ethernet0/0",
" description Test Interface",
" ip address 10.0.0.1 255.255.255.0",
"! ",
"end"
]
config = BaseConfigParser(config=config_list)
```

INTERFACE_LINE_CLASS

alias of *ccutils.ccparser.BaseInterfaceLine.BaseInterfaceLine*

PATTERN_TYPE

alias of `re.Pattern`

_check_path (*filepath*)

_compile_regex (*regex*, *flags*=<*RegexFlag.MULTILINE*: 8>)

Helper function for compiling *re* patterns from string.

Parameters

- **regex** (*str*) – Regex string
- **flags** – Flags for regex pattern, default is `re.MULTILINE`

Returns

_create_cfg_line_objects ()

Function for generating `self.lines`.

_device_tracking_attach_policy_regex = `re.compile('^ device-tracking attach-policy (?P`

_get_clean_config (*first_line_regex*=`^version \d+\.\d+`, *last_line_regex*=`^end`)

_get_indent (*line*)

_vlan_configuration_regex = `re.compile('^vlan configuration (?P<vlan_range>[\\d\\- ,]+)`

`cdp`

config_lines_obj

Kept for backwards compatibility, will be removed in future versions.

Returns `BaseConfigParser.lines`

Return type list

domain_name

find_objects (*regex*, *flags*=<*RegexFlag.MULTILINE*: 8>)

Function for filtering Config Lines Objects based on given regex.

Parameters

- **regex** (`re.Pattern` or *str*) – Regex based on which the search is done
- **flags** (*int*, optional) – Set custom flags for regex, defaults to `re.MULTILINE`

Examples

Example:

```
# Initialize the object
config = BaseConfigParser(config="/path/to/config_file.cfg")

# Define regex for matching config lines
interface_regex = r"^ interface"

# Apply the filter
interface_lines = config.find_objects(regex=interface_regex)

# Returns subset of `self.lines` which match specified regex
```

fix_indents ()

Function for fixing the indentation level of config lines.

Returns**get_section_by_parents** (*parents*)**hostname****interface_lines****lines = None**

This is a URI.

match_to_dict (*line, patterns*)**Parameters**

- **line** – Instance of *BaseConfigLine* object
- **patterns** – List of compiled *re* patterns
- **minimal_result** – Bool, if True, omits keys with value *None*

Returns Dictionary containing named groups across all provided patterns**Return type** dict**name_servers****parse** ()

Entry function which triggers the parsing process. Called automatically when instantiating the object.

Returns None**property_autoparse** (*candidate_pattern, patterns*)Function for searching multiple patterns across all occurrences of lines that matched *candidate_pattern*
:param candidate_pattern: :param patterns:

Returns:

section_property_autoparse (*parent, patterns, return_with_line=False*)**static_routes****vlan_groups****vlangs****vrrfs**

1.3 CiscosParser

1.3.1 Quick Start

```
from ccutils.ccparser import CiscoIosParser
import pathlib

# Optinal - Create pathlib object pointing to your config file
path_to_file = pathlib.Path("/path/to/config_file.txt")

config = CiscoIosParser(config=path_to_file)

# Print number of config lines
print(len(config.lines))
```


class CiscoIosParser (*config=None, verbosity=4, **kwargs*)

Bases: `ccutils.ccparser.BaseConfigParser.BaseConfigParser`

INTERFACE_LINE_CLASS

alias of `ccutils.ccparser.CiscoIosInterfaceLine.CiscoIosInterfaceLine`

aaa_authorization_exec_methods

aaa_login_methods

all_ipv4_physical_addresses

Get all physical IPv4 addresses of device config.

Returns

List IPv4 Addresses

Example:

```
[
    "192.168.100.2",
    "192.168.101.2"
]
```

Returns empty list ([]) if absent

Return type list

all_ipv4_standby_addresses

List IPv4 Addresses

Example:

```
[
    "192.168.100.1",
    "192.168.101.1"
]
```

Returns empty list ([]) if absent

Type Returns

Type list

cdp

domain_name

hostname

isis

type: Returns

logging

logging_global_params

logging_servers

name_servers

ntp

ntp_access_groups

ntp_authentication_keys

ntp_global_params

ntp_peers

ntp_servers

Property containing DNS servers related data

Returns

List of name server IP addresses

Example:

```
[
  "10.0.0.1",
  "10.0.0.2"
]
```

Returns None if absent

Return type list

ntp_trusted_keys

ospf

type: Returns

radius_groups

List of RADIUS Server Groups Entries

Example:

```
[
  {
    "name": "RADIUS-GROUP",
    "source_interface": "Vlan100",
    "servers": [
      {
        "name": "RADIUS-Primary"
      }
    ]
  }
]
```

Returns None if absent

Type Returns

Type list

radius_servers

List of RADIUS Servers

Example:

```
[
  {
    "name": "RADIUS-Primary",
    "address_version": "ipv4",
    "server": "10.0.0.1",
    "encryption_type": null,
    "hash": "Test123",
    "timeout": "2",
  }
]
```

(continues on next page)

(continued from previous page)

```

    "retransmit": "1",
    "auth_port": "1812",
    "acct_port": "1813"
  },
  {
    "name": "RADIUS-Secondary",
    "address_version": "ipv4",
    "server": "10.0.1.1",
    "encryption_type": null,
    "hash": "Test123",
    "timeout": "2",
    "retransmit": "1",
    "auth_port": "1812",
    "acct_port": "1813"
  }
]

```

Returns None if absent

Type Returns

Type list

section_unprocessed_lines (*parent, check_patterns*)

tacacs_groups

List of TACACS Server entries

Example:

```

[
  {
    "name": "ISE-TACACS",
    "source_interface": "Loopback0",
    "servers": [
      {
        "name": "ISE-1"
      },
      {
        "name": "ISE-2"
      }
    ]
  }
]

```

Returns None if absent

Type Returns

Type list

tacacs_servers

List of TACACS Servers

Example:

```

[
  {
    "name": "ISE-1",
    "address_version": "ipv4",

```

(continues on next page)

(continued from previous page)

```

    "server": "10.0.0.1",
    "encryption_type": "7",
    "hash": "36A03A8A4C00E81F03D62D8B04BBBF4D",
    "timeout": "10",
    "single_connection": true
  },
  {
    "name": "ISE-2",
    "address_version": "ipv4",
    "server": "10.0.1.1",
    "encryption_type": "7",
    "hash": "36A03A8A4C00E81F03D62D8B04BBBF4D",
    "timeout": "10",
    "single_connection": true
  }
]

```

Returns None if absent

Type Returns

Type list

vlan_groups

vlangs

vrf_ipv4_physical_addresses

Get all physical IPv4 addresses in particular VRF. By default returns all IP addresses from global routing table

Parameters **vrf** – Name of the VRF (default is “*global*”)

Returns

List IPv4 Addresses

Example:

```

[
  "192.168.100.2",
  "192.168.101.2"
]

```

Returns empty list ([]) if absent

Return type list

vrf_ipv4_standby_addresses

Name of the VRF (default is “*global*”)

Returns

List IPv4 Addresses

Example:

```

[
  "192.168.100.1",
  "192.168.101.1"
]

```

Returns empty list ([]) if absent

Return type list

Type Args

Type vrf

vrfs

1.4 BaseConfigLine

class BaseConfigLine (*number, text, config, verbosity=3, name='BaseConfigLine'*)

Bases: object

This class is not meant to be instantiated directly, but only from BaseConfigParser instance.

Parameters

- **number** (*int*) – Index of line in config
- **text** (*str*) – Text of the config line
- **config** (*BaseConfigParser*) – Reference to the parent BaseConfigParser object
- **verbosity** (*int*, optional) – Logging output level, defaults to 3: Warning

PATTERN_TYPE

alias of `re.Pattern`

comment_regex = `re.compile('^(\\s+)?!.*', re.MULTILINE)`

get_children()

Return all children lines (all following lines with larger indent)

Returns List of child config lines (objects)

Return type list

get_parent

get_parents

get_type

Return *types* of config line. Used mostly for filtering purposes.

Currently available values are:

- parent
- child
- interface
- comment

Returns List of types

Return type list

is_child

Check whether this line is a child

Returns True if line is a child line, False otherwise

Return type bool

is_interface

is_parent

Check whether this line is a parent

Returns True if line is a parent line, False otherwise

Return type bool

re_match (*regex*, *group=None*)

re_search (*regex*, *group=None*)

Search config line for given regex

Parameters

- **regex** (*re.Pattern* or *str*) – Regex to search for
- **group** (*str* or *int*, optional) – Return only specific (named or numbered) group of given regex. If set to “ALL”, return value will be a dictionary with all named groups of the regex.

Examples

Example:

```
# Given the following line stored in `line` variable
# " ip address 10.0.0.1 255.255.255"
pattern = r"^ ip address (?P<ip>\S+) (?P<mask>\S+)"

# Basic search
result = line.re_search(regex=pattern)
print(result)
# Returns: " ip address 10.0.0.1 255.255.255"

# Search for specific group
result = line.re_search(regex=pattern, group="ip")
print(result)
# Returns: "10.0.0.1"

# Get all named groups
result = line.re_search(regex=pattern, group="ALL")
print(result)
# Returns: {"ip": "10.0.0.1", "mask": "255.255.255"}
```

Returns

String that matched given regex, or, if *group* was provided, returns only specific group.

Returns None if regex did not match.

Return type str

re_search_children (*regex*, *group=None*)

Search all children for given regex.

Parameters

- **regex** (*re.Pattern* or *str*) – Regex to search for

- **group** (str or int, optional) – Return only specific (named or numbered) group of given regex. If set to “ALL”, return value will be a dictionary with all named groups of the regex.

Returns

List of all child object which match given regex, or, if *group* was provided, returns list containing matched group across all children.

Example:

```
# Given following config section, interface line stored in
→ `line` variable
config = '''
interface Ethernet0/0
  description Test Interface
  ip address 10.0.0.1 255.255.255.0
  ip address 10.0.1.1 255.255.255.0 secondary
!
'''
pattern = r"^ ip address (?P<ip>\S+) (?P<mask>\S+)"

result = line.re_search_children(regex=pattern)
print(result)
# Returns: [
#   [BaseConfigLine #2 (child): ip address 10.0.0.1 255.255.255.
→0],
#   [BaseConfigLine #3 (child): ip address 10.0.1.1 255.255.255.
→0 secondary]
# ]

result = line.re_search_children(regex=pattern, group="ip")
print(result)
# Returns: [
#   "10.0.0.1",
#   "10.0.1.1"
# ]

result = line.re_search_children(regex=pattern, group="ALL")
print(result)
# Returns: [
#   {"ip": "10.0.0.1", "mask": "255.255.255.0"},
#   {"ip": "10.0.1.1", "mask": "255.255.255.0"}
# ]
```

Return type list

re_search_children_multipattern (regexes: list, group=None, deduplicate: bool = True) → list

Wrapper function for `self.re_search_children()` allowing to use multiple patterns

Parameters

- **regexes** (list) – List of patterns to search
- **group** (str or int, optional) – Return only specific (named or numbered) group of given regex. If set to “ALL”, return value will be a dictionary with all named groups of the regex.
- **deduplicate** (bool, optional) – When set to `True` (default), results will not contain duplicate line objects in cases where multiple patterns match the same line.

Returns List of all child object which match given regex, or, if *group* was provided, returns list containing matched group across all children.

Return type list

`return_obj()`

1.5 BaseInterfaceLine

class BaseInterfaceLine (*number, text, config, verbosity=3, name='BaseInterfaceLine'*)

Bases: `ccutils.ccparser.BaseConfigLine.BaseConfigLine`

Object for retrieving various config options on the interface level.

This class is not meant to be instantiated directly, but only from BaseConfigParser instance.

Parameters

- **number** (*int*) – Index of line in config
- **text** (*str*) – Text of the config line
- **config** (*BaseConfigParser*) – Reference to the parent BaseConfigParser object
- **verbosity** (*int*, optional) – Logging output level, defaults to 3: Warning

access_vlan

Return a number of access VLAN or *None* if the command `switchport access vlan x` is not present.

Caution: This does not mean the interface is necessarily an access port.

Returns

Number of access VLAN or *None*

Returns *None* if absent

Return type int

bandwidth

Return bandwidth of the interface set by command **bandwidth X**.

Returns

Bandwidth

Returns *None* if absent

Return type int

cdp

Checks whether CDP is enabled on the interface. This property takes global CDP configuration into account, meaning if there is no specific configuration on the interface level, it will return state based on the entire config (eg. *no cdp run* in the global config will make this property be *False*)

Returns *True* if CDP is enabled, *False* otherwise

Return type bool

channel_group

Return a dictionary describing Port-channel/Etherchannel related configuration

Returns

Channel-group parameters

Example:

```
{"channel_group_number": "1", "channel_group_mode": "active"}
```

Otherwise returns None

Return type dict

delay

Return delay of the interface set by command **delay X**.

Returns

Delay

Returns None if absent

Return type int

description

Returns description of the interface.

Returns

Interface description

Returns None if absent

Return type str

device_tracking_policy**duplex**

Return duplex of the interface set by command **duplex X**.

Returns

Duplex

Returns None if absent

Return type str

encapsulation

Return encapsulation type and tag for subinterfaces

Returns

Encapsulation parameters

Example:

```
{"type": "dot1q", "tag": 10, "native": False}
```

Returns None if absent

Return type dict

flags

List of flags/tags describing basic properties of the interface. Used for filtering purposes. Currently supported flags are:

12 - Interface is switched port

l3 - Interface is routed port

physical - Interface is a physical interface (Only *Ethernet interfaces)

svi - Interface is SVI (VLAN Interface)

port-channel - Interface is port-channel

pc-member - Interface is a member of Port-channel

tunnel - Interface is a Tunnel

Returns List of flags

Return type list

get_unprocessed

Return a list of config lines under the interface, which did not match any of the existing regex patterns. Mostly for development/testing purposes.

By default returns list of objects.

Parameters **return_type** (*str*) – Set this to “text” to receive list of strings

Returns List of unprocessed config lines

Return type list

helper_address

Return a list of IP addresses specified with **ip helper-address** command (DHCP relay).

Returns

List of helper addresses

Returns None if absent

Return type list

interface_description

interface_name

ip_addresses

Return list of IP addresses present on the interface

Returns

List of dictionaries representing individual IP addresses

Example:

```
[
  {
    "ip_address": "10.0.0.1",
    "mask": "255.255.255.0",
    "secondary": False
  },
  {
    "ip_address": "10.0.1.1",
    "mask": "255.255.255.0",
    "secondary": True
  }
]
```

If there is no IP address present on the interface, an empty list is returned.

Return type list

ip_mtu

Return IP MTU of the interface set by command **ip mtu X**.

Returns

IP MTU

Returns `None` if absent

Return type int

ip_unnumbered_interface

keepalive

load_interval

Return Load Interval of the interface set by command **load-interval X**.

Returns

Load Interval

Returns `None` if absent

Return type int

logging_events

mtu

Return MTU of the interface set by command **mtu X**.

Returns

MTU

Returns `None` if absent

Return type int

name

Return name of the interface, such as *GigabitEthernet0/1*.

Returns Name of the interface

Return type str

native_vlan

Return Native VLAN of L2 Interface

Returns

Native VLAN Number (*None* if absent)

Returns `None` if absent

Return type int

ospf

Return OSPF interface parameters

Returns

OSPF parameters

Example:

```
{ "process_id": 1, "area": 0, "network_type": "point-to-point",  
  ↪ "priority": 200 }
```

Returns None if absent

Return type dict

ospf_priority

Returns OSPF priority of the interface.

Returns OSPF Priority or None

Return type int

port_mode

Checks whether the interface is running in switched (**l2**) or routed (**l3**) mode.

Returns *l2* or *l3*

Return type str

service_instances

service_policy

Return names of applied service policies

Returns

Dictionary containing names of both input and output policies.

Example:

```
{ "input": "TEST_INPUT_POLICY", "output": "TEST_OUTPUT_POLICY" }
```

If there are no policies specified, returns:

```
{ "input": None, "output": None }
```

Return type dict

shutdown

speed

Return speed of the interface set by command **speed X**

Returns

Speed

Returns None if absent

Return type int

standby

HSRP related configuration. Groups, IP addresses, hello/hold timers, priority and authentication.

Returns Dictionary with top level keys being HSRP groups.

storm_control

switchport_mode

Return L2 Mode of interface, either access or trunk

Returns

“access” or “trunk”

Returns `None` if absent

Return type `str`

switchport_nonegotiate

Check whether the port is running DTP or not. Checks for presence of `switchport nonegotiate` command

Returns `True` if command is present, `False` otherwise

Return type `bool`

tcp_mss

Return TCP Max Segment Size of the interface set by command `ip tcp adjust-mss X`.

Returns

TCP MSS

Returns `None` if absent

Return type `int`

trunk_allowed_vlans

Return a expanded list of VLANs allowed with `switchport trunk allowed vlan x,y,z`.

Caution: This does not mean the interface is necessarily a trunk port.

Returns

Expanded list of allowed VLANs

Returns `None` if absent

Returns “none” if `switchport trunk allowed vlan none`

Return type `list`

trunk_encapsulation

Return encapsulation on trunk interfaces

Returns

“dot1q” or “isl”

Returns `None` if absent

Return type `str`

tunnel_properties

Return properties related to Tunnel interfaces

Returns

Dictionary with tunnel properties.

Example:

```
{
  "source": "Loopback0",
  "destination": "10.0.0.1",
  "vrf": None,
  "mode": "ipsec ipv4",
  "ipsec_profile": "TEST_IPSEC_PROFILE"
}
```

Returns `None` if absent

Return type dict

voice_vlan

Return a number of voice VLAN

Returns

Number of voice VLAN or None

Returns None if absent

Return type int

vrf

Return VRF of the interface

Returns

Name of the VRF

Returns None if absent

Return type str

1.6 CiscoIosInterfaceLine

class CiscoIosInterfaceLine (*number, text, config, verbosity=3*)

Bases: `ccutils.ccparser.BaseInterfaceLine`.`BaseInterfaceLine`

access_vlan

Return a number of access VLAN or *None* if the command `switchport access vlan x` is not present.

Caution: This does not mean the interface is necessarily an access port.

Returns

Number of access VLAN or None

Returns None if absent

Return type int

bandwidth

Return bandwidth of the interface set by command **bandwidth X**.

Returns

Bandwidth

Returns None if absent

Return type int

bfd

cdp

Checks whether CDP is enabled on the interface. This property takes global CDP configuration into account, meaning if there is no specific configuration on the interface level, it will return state based on the entire config (eg. *no cdp run* in the global config will make this property be *False*)

Returns True if CDP is enabled, False otherwise

Return type bool

channel_group

Return a dictionary describing Port-channel/Etherchannel related configuration

Returns

Channel-group parameters

Example:

```
{"channel_group_number": "1", "channel_group_mode": "active"}
```

Otherwise returns None

Return type dict

delay

Return delay of the interface set by command **delay X**.

Returns

Delay

Returns None if absent

Return type int

description

Returns description of the interface.

Returns

Interface description

Returns None if absent

Return type str

device_tracking_policy**dhcp_snooping****duplex**

Return duplex of the interface set by command **duplex X**.

Returns

Duplex

Returns None if absent

Return type str

encapsulation

Return encapsulation type and tag for subinterfaces

Returns

Encapsulation parameters

Example:

```
{"type": "dot1q", "tag": 10, "native": False}
```

Returns None if absent

Return type dict

flags

List of flags/tags describing basic properties of the interface. Used for filtering purposes. Currently supported flags are:

12 - Interface is switched port

13 - Interface is routed port

physical - Interface is a physical interface (Only *Ethernet interfaces)

svi - Interface is SVI (VLAN Interface)

port-channel - Interface is port-channel

pc-member - Interface is a member of Port-channel

tunnel - Interface is a Tunnel

Returns List of flags

Return type list

get_unprocessed

Return a list of config lines under the interface, which did not match any of the existing regex patterns. Mostly for development/testing purposes.

By default returns list of objects.

Parameters **return_type** (*str*) – Set this to “text” to receive list of strings

Returns List of unprocessed config lines

Return type list

helper_address

Return a list of IP addresses specified with **ip helper-address** command (DHCP relay).

Returns

List of helper addresses

Returns None if absent

Return type list

interface_description**interface_name****ip_addresses**

Return list of IP addresses present on the interface

Returns

List of dictionaries representing individual IP addresses

Example:

```
[
  {
    "ip_address": "10.0.0.1",
    "mask": "255.255.255.0",
    "secondary": False
  },
  {
    "ip_address": "10.0.1.1",
```

(continues on next page)

(continued from previous page)

```

        "mask": "255.255.255.0",
        "secondary": True
    }
]

```

If there is no IP address present on the interface, an empty list is returned.

Return type list

ip_mtu

Return IP MTU of the interface set by command **ip mtu X**.

Returns

IP MTU

Returns None if absent

Return type int

ip_unnumbered_interface

ipv4_addresses

Return list of IPv4 addresses present on the interface

Returns

List of dictionaries representing individual IPv4 addresses

Example:

```

[
  {
    "address": "10.0.0.1",
    "mask": "255.255.255.0",
    "secondary": False
  },
  {
    "address": "10.0.1.1",
    "mask": "255.255.255.0",
    "secondary": True
  }
]

```

If there is no IP address present on the interface, an empty list is returned.

Return type list

ipv4_unnumbered_interface

ipv6_addresses

Return list of IPv6 addresses present on the interface

Returns

List of dictionaries representing individual IPv6 addresses

If there is no IP address present on the interface, an empty list is returned.

Return type list

isis

Return IS-IS interface parameters

Returns

IS-IS parameters

Example:

```
{ }
```

Returns `None` if absent

Return type dict

keepalive

load_interval

Return Load Interval of the interface set by command **load-interval X**.

Returns

Load Interval

Returns `None` if absent

Return type int

logging_events

mtu

Return MTU of the interface set by command **mtu X**.

Returns

MTU

Returns `None` if absent

Return type int

name

Return name of the interface, such as *GigabitEthernet0/1*.

Returns Name of the interface

Return type str

native_vlan

Return Native VLAN of L2 Interface

Returns

Native VLAN Number (*None* if absent)

Returns `None` if absent

Return type int

negotiation

ospf

Return OSPF interface parameters

Returns

OSPF parameters

Example:

```
{ "process_id": 1, "area": 0, "network_type": "point-to-point",
  ↪ "priority": 200 }
```

Returns None if absent

Return type dict

ospf_priority

Returns OSPF priority of the interface.

Returns OSPF Priority or None

Return type int

port_mode

Checks whether the interface is running in switched (**12**) or routed (**13**) mode.

Returns *12* or *13*

Return type str

service_instances

service_policy

Return names of applied service policies

Returns

Dictionary containing names of both input and output policies.

Example:

```
{ "input": "TEST_INPUT_POLICY", "output": "TEST_OUTPUT_POLICY" }
```

If there are no policies specified, returns:

```
{ "input": None, "output": None }
```

Return type dict

shutdown

speed

Return speed of the interface set by command **speed X**

Returns

Speed

Returns None if absent

Return type int

standby

HSRP related configuration. Groups, IP addresses, hello/hold timers, priority and authentication.

Returns Dictionary with top level keys being HSRP groups.

standby_v1

Use `self.hsrp` or `self.standby` instead HSRP related configuration. Groups, IP addresses, hello/hold timers, priority and authentication.

Returns Dictionary with top level keys being HSRP groups.

Type DEPRECATED

storm_control

switchport_mode

Return L2 Mode of interface, either access or trunk

Returns

“access” or “trunk”

Returns `None` if absent

Return type str

switchport_nonegotiate

Check whether the port is running DTP or not. Checks for presence of `switchport nonegotiate` command

Returns `True` if command is present, `False` otherwise

Return type bool

tcp_mss

Return TCP Max Segment Size of the interface set by command `ip tcp adjust-mss X`.

Returns

TCP MSS

Returns `None` if absent

Return type int

trunk_allowed_vlans

Return an expanded list of VLANs allowed with `switchport trunk allowed vlan x,y,z`.

Caution: This does not mean the interface is necessarily a trunk port.

Returns

Expanded list of allowed VLANs

Returns `None` if absent

Returns “none” if `switchport trunk allowed vlan none`

Return type list

trunk_encapsulation

Return encapsulation on trunk interfaces

Returns

“dot1q” or “isl”

Returns `None` if absent

Return type str

tunnel_properties

Return properties related to Tunnel interfaces

Returns

Dictionary with tunnel properties.

Example:

```
{
  "source": "Loopback0",
  "destination": "10.0.0.1",
  "vrf": None,
  "mode": "ipsec ipv4",
  "ipsec_profile": "TEST_IPSEC_PROFILE"
}
```

Returns None if absent

Return type dict

voice_vlan

Return a number of voice VLAN

Returns

Number of voice VLAN or None

Returns None if absent

Return type int

vrf

Return VRF of the interface

Returns

Name of the VRF

Returns None if absent

Return type str

1.7 ConfigToJson

class ConfigToJson (*config*, *omit_empty=False*, *verbosity=3*)

Bases: object

Parameters

- **config** – Reference to the parent BaseConfigParser object
- **verbosity** (*int*) – Logging output level

get_interface_list (*flags_filter=None*)

get_ordered_interfaces ()

Return interfaces as OrderedDict

Returns Interface section as OrderedDict

Return type (OrderedDict)

static jprint (*data*)

parse_common ()

parse_interfaces ()

Returns

to_json (*indent=2*)

Return JSON formatted structure describing configuration

Parameters `indent` (*int*) – Set JSON indent, defaults to 2

Returns JSON string

Return type str

`to_yaml()`

Return YAML formatted structure describing configuration

Returns YAML string

Return type str

1.8 ConfigMigration

`class ConfigMigration` (*hostname, excel_path, excel_sheet, old_config_folder, verbosity=1*)

Bases: object

`check_standby()`

`get_context_for_new_interface` (*new_interface*)

`get_interface_mapping()`

`get_new_interface` (*old_host, old_interface*)

`get_old_configs()`

`get_old_ctj()`

`get_old_hostnames` (*column='Old Host'*)

`merge_vlans()`

`merge_vrfs()`

`user_selection` (*prompt, options*)

2.1 CC Templater

```
class CCTemplater(template_folder=None)
```

```
    Bases: object
```

```
    render (template_name, context)
```


CHAPTER 3

Templates

3.1 Placeholder

4.1 Common Utils

class UnsortableList

sort (*args, **kwargs)
Stable sort *IN PLACE*.

class UnsortableOrderedDict

items () → a set-like object providing a view on D's items

check_path (path, create=False)

Parameters

- **path** –
- **create** –

Returns

convert_interface_name (interface: str, out: str = 'long')

This function converts interface names between long and short variants. For example Fa0/1 -> FastEthernet0/1 or the other way around.

Parameters

- **interface** –
- **out** –

Return type str

Returns Interface string

get_logger (name, verbosity=4)

match_to_json (*match*, *groups*)

This function converts *re* match object to dict

Parameters

- **match** – re.match object
- **groups** – list

Return type dict

Returns Dictionary with matched groups

split_interface_name (*interface: str*)

This function takes in interface string such as “GigabitEthernet0/10” and returns a list containing name and number, such as [“GigabitEthernet”, “0/10”]

Parameters **interface** (*str*) – Interface to perform split on

Returns List containing name and number of interface, such as ["GigabitEthernet", "0/10"]

Return type list

4.2 CiscoRange

class CiscoRange (*text*, *verbosity=3*)

Bases: collections.abc.MutableSequence

CHANNEL_REGEX = re.compile('\:\:(?P<number>\d+)

PREFIX_REGEX = re.compile('^[A-z\-\-]+(?:\d)', re.MULTILINE)

PREFIX_SLOT_REGEX = re.compile('(P<prefix_slot>^[A-z\-\-]+(?:\d) (?:\d+/*) (P<number>

RANGE_REGEX = re.compile('\d+\s*\-\s*\d+')

SUBINT_REGEX = re.compile('\.\.(?P<number>\d+)\\$')

SUFFIX_REGEX = re.compile('\d.*?\\$')

add (*data*)

check_prefix (*data*)

compress_list (*data*)

has_prefix (*data*)

insert (*index*, *value*)

S.insert(index, value) – insert value before index

static int_or_none (*item*)

remove (*data*)

S.remove(value) – remove first occurrence of value. Raise ValueError if the value is not present.

sort_list (*data*)

split_item (*item*)

split_text (*text*)

split_to_list (*data*)

to_string ()

CHAPTER 5

Indices and tables

- `genindex`
- `modindex`
- `search`

C

`ccutils.utils.common_utils`, 31

Symbols

_check_path() (*BaseConfigParser method*), 3
 _compile_regex() (*BaseConfigParser method*), 3
 _create_cfg_line_objects() (*BaseConfigParser method*), 3
 _device_tracking_attach_policy_regex (*BaseConfigParser attribute*), 3
 _get_clean_config() (*BaseConfigParser method*), 3
 _get_indent() (*BaseConfigParser method*), 3
 _vlan_configuration_regex (*BaseConfigParser attribute*), 3

A

aaa_authorization_exec_methods (*CiscoIosParser attribute*), 5
 aaa_login_methods (*CiscoIosParser attribute*), 5
 access_vlan (*BaseInterfaceLine attribute*), 12
 access_vlan (*CiscoIosInterfaceLine attribute*), 18
 add() (*CiscoRange method*), 32
 all_ipv4_physical_addresses (*CiscoIosParser attribute*), 5
 all_ipv4_standby_addresses (*CiscoIosParser attribute*), 5

B

bandwidth (*BaseInterfaceLine attribute*), 12
 bandwidth (*CiscoIosInterfaceLine attribute*), 18
 BaseConfigLine (*class in ccutils.ccparser*), 9
 BaseConfigParser (*class in ccutils.ccparser*), 1
 BaseInterfaceLine (*class in ccutils.ccparser*), 12
 bfd (*CiscoIosInterfaceLine attribute*), 18

C

CCTemplater (*class in ccutils.cctemplater*), 27
 ccutils.utils.common_utils (*module*), 31
 cdp (*BaseConfigParser attribute*), 3
 cdp (*BaseInterfaceLine attribute*), 12
 cdp (*CiscoIosInterfaceLine attribute*), 18

cdp (*CiscoIosParser attribute*), 5
 channel_group (*BaseInterfaceLine attribute*), 12
 channel_group (*CiscoIosInterfaceLine attribute*), 18
 CHANNEL_REGEX (*CiscoRange attribute*), 32
 check_path() (*in module ccutils.utils.common_utils*), 31
 check_prefix() (*CiscoRange method*), 32
 check_standby() (*ConfigMigration method*), 26
 CiscoIosInterfaceLine (*class in ccutils.ccparser*), 18
 CiscoIosParser (*class in ccutils.ccparser*), 4
 CiscoRange (*class in ccutils.utils*), 32
 comment_regex (*BaseConfigLine attribute*), 9
 compress_list() (*CiscoRange method*), 32
 config_lines_obj (*BaseConfigParser attribute*), 3
 config_lines_str (*BaseConfigParser attribute*), 2
 ConfigMigration (*class in ccutils.ccparser*), 26
 ConfigToJson (*class in ccutils.ccparser*), 25
 convert_interface_name() (*in module ccutils.utils.common_utils*), 31

D

delay (*BaseInterfaceLine attribute*), 13
 delay (*CiscoIosInterfaceLine attribute*), 19
 description (*BaseInterfaceLine attribute*), 13
 description (*CiscoIosInterfaceLine attribute*), 19
 device_tracking_policy (*BaseInterfaceLine attribute*), 13
 device_tracking_policy (*CiscoIosInterfaceLine attribute*), 19
 dhcp_snooping (*CiscoIosInterfaceLine attribute*), 19
 domain_name (*BaseConfigParser attribute*), 3
 domain_name (*CiscoIosParser attribute*), 5
 duplex (*BaseInterfaceLine attribute*), 13
 duplex (*CiscoIosInterfaceLine attribute*), 19

E

encapsulation (*BaseInterfaceLine attribute*), 13
 encapsulation (*CiscoIosInterfaceLine attribute*), 19

F

find_objects() (*BaseConfigParser* method), 3
 fix_indents() (*BaseConfigParser* method), 3
 flags (*BaseInterfaceLine* attribute), 13
 flags (*CiscoIosInterfaceLine* attribute), 19

G

get_children() (*BaseConfigLine* method), 9
 get_context_for_new_interface() (*ConfigMigration* method), 26
 get_interface_list() (*ConfigToJson* method), 25
 get_interface_mapping() (*ConfigMigration* method), 26
 get_logger() (*in module ccutils.utils.common_utils*), 31
 get_new_interface() (*ConfigMigration* method), 26
 get_old_configs() (*ConfigMigration* method), 26
 get_old_ctj() (*ConfigMigration* method), 26
 get_old_hostnames() (*ConfigMigration* method), 26
 get_ordered_interfaces() (*ConfigToJson* method), 25
 get_parent (*BaseConfigLine* attribute), 9
 get_parents (*BaseConfigLine* attribute), 9
 get_section_by_parents() (*BaseConfigParser* method), 4
 get_type (*BaseConfigLine* attribute), 9
 get_unprocessed (*BaseInterfaceLine* attribute), 14
 get_unprocessed (*CiscoIosInterfaceLine* attribute), 20

H

has_prefix() (*CiscoRange* method), 32
 helper_address (*BaseInterfaceLine* attribute), 14
 helper_address (*CiscoIosInterfaceLine* attribute), 20
 hostname (*BaseConfigParser* attribute), 4
 hostname (*CiscoIosParser* attribute), 5

I

insert() (*CiscoRange* method), 32
 int_or_none() (*CiscoRange* static method), 32
 interface_description (*BaseInterfaceLine* attribute), 14
 interface_description (*CiscoIosInterfaceLine* attribute), 20
 INTERFACE_LINE_CLASS (*BaseConfigParser* attribute), 2
 INTERFACE_LINE_CLASS (*CiscoIosParser* attribute), 5
 interface_lines (*BaseConfigParser* attribute), 4
 interface_name (*BaseInterfaceLine* attribute), 14

interface_name (*CiscoIosInterfaceLine* attribute), 20
 ip_addresses (*BaseInterfaceLine* attribute), 14
 ip_addresses (*CiscoIosInterfaceLine* attribute), 20
 ip_mtu (*BaseInterfaceLine* attribute), 15
 ip_mtu (*CiscoIosInterfaceLine* attribute), 21
 ip_unnumbered_interface (*BaseInterfaceLine* attribute), 15
 ip_unnumbered_interface (*CiscoIosInterfaceLine* attribute), 21
 ipv4_addresses (*CiscoIosInterfaceLine* attribute), 21
 ipv4_unnumbered_interface (*CiscoIosInterfaceLine* attribute), 21
 ipv6_addresses (*CiscoIosInterfaceLine* attribute), 21
 is_child (*BaseConfigLine* attribute), 9
 is_interface (*BaseConfigLine* attribute), 10
 is_parent (*BaseConfigLine* attribute), 10
 isis (*CiscoIosInterfaceLine* attribute), 21
 isis (*CiscoIosParser* attribute), 5
 items() (*UnsortableOrderedDict* method), 31

J

jprint() (*ConfigToJson* static method), 25

K

keepalive (*BaseInterfaceLine* attribute), 15
 keepalive (*CiscoIosInterfaceLine* attribute), 22

L

lines (*BaseConfigParser* attribute), 2, 4
 load_interval (*BaseInterfaceLine* attribute), 15
 load_interval (*CiscoIosInterfaceLine* attribute), 22
 logging (*CiscoIosParser* attribute), 5
 logging_events (*BaseInterfaceLine* attribute), 15
 logging_events (*CiscoIosInterfaceLine* attribute), 22
 logging_global_params (*CiscoIosParser* attribute), 5
 logging_servers (*CiscoIosParser* attribute), 5

M

match_to_dict() (*BaseConfigParser* method), 4
 match_to_json() (*in module ccutils.utils.common_utils*), 31
 merge_vlans() (*ConfigMigration* method), 26
 merge_vrfs() (*ConfigMigration* method), 26
 mtu (*BaseInterfaceLine* attribute), 15
 mtu (*CiscoIosInterfaceLine* attribute), 22

N

name (*BaseInterfaceLine* attribute), 15

name (*CiscoIosInterfaceLine attribute*), 22
 name_servers (*BaseConfigParser attribute*), 4
 name_servers (*CiscoIosParser attribute*), 5
 native_vlan (*BaseInterfaceLine attribute*), 15
 native_vlan (*CiscoIosInterfaceLine attribute*), 22
 negotiation (*CiscoIosInterfaceLine attribute*), 22
 ntp (*CiscoIosParser attribute*), 5
 ntp_access_groups (*CiscoIosParser attribute*), 5
 ntp_authentication_keys (*CiscoIosParser attribute*), 5
 ntp_global_params (*CiscoIosParser attribute*), 5
 ntp_peers (*CiscoIosParser attribute*), 6
 ntp_servers (*CiscoIosParser attribute*), 6
 ntp_trusted_keys (*CiscoIosParser attribute*), 6

O

ospf (*BaseInterfaceLine attribute*), 15
 ospf (*CiscoIosInterfaceLine attribute*), 22
 ospf (*CiscoIosParser attribute*), 6
 ospf_priority (*BaseInterfaceLine attribute*), 16
 ospf_priority (*CiscoIosInterfaceLine attribute*), 23

P

parse() (*BaseConfigParser method*), 4
 parse_common() (*ConfigToJson method*), 25
 parse_interfaces() (*ConfigToJson method*), 25
 PATTERN_TYPE (*BaseConfigLine attribute*), 9
 PATTERN_TYPE (*BaseConfigParser attribute*), 2
 port_mode (*BaseInterfaceLine attribute*), 16
 port_mode (*CiscoIosInterfaceLine attribute*), 23
 PREFIX_REGEX (*CiscoRange attribute*), 32
 PREFIX_SLOT_REGEX (*CiscoRange attribute*), 32
 property_autoparse() (*BaseConfigParser method*), 4

R

radius_groups (*CiscoIosParser attribute*), 6
 radius_servers (*CiscoIosParser attribute*), 6
 RANGE_REGEX (*CiscoRange attribute*), 32
 re_match() (*BaseConfigLine method*), 10
 re_search() (*BaseConfigLine method*), 10
 re_search_children() (*BaseConfigLine method*), 10
 re_search_children_multipattern() (*BaseConfigLine method*), 11
 remove() (*CiscoRange method*), 32
 render() (*CCTemplater method*), 27
 return_obj() (*BaseConfigLine method*), 12

S

section_property_autoparse() (*BaseConfigParser method*), 4
 section_unprocessed_lines() (*CiscoIosParser method*), 7

service_instances (*BaseInterfaceLine attribute*), 16
 service_instances (*CiscoIosInterfaceLine attribute*), 23
 service_policy (*BaseInterfaceLine attribute*), 16
 service_policy (*CiscoIosInterfaceLine attribute*), 23
 shutdown (*BaseInterfaceLine attribute*), 16
 shutdown (*CiscoIosInterfaceLine attribute*), 23
 sort() (*UnsortableList method*), 31
 sort_list() (*CiscoRange method*), 32
 speed (*BaseInterfaceLine attribute*), 16
 speed (*CiscoIosInterfaceLine attribute*), 23
 split_interface_name() (*in module ccutils.utils.common_utils*), 32
 split_item() (*CiscoRange method*), 32
 split_text() (*CiscoRange method*), 32
 split_to_list() (*CiscoRange method*), 32
 standby (*BaseInterfaceLine attribute*), 16
 standby (*CiscoIosInterfaceLine attribute*), 23
 standby_v1 (*CiscoIosInterfaceLine attribute*), 23
 static_routes (*BaseConfigParser attribute*), 4
 storm_control (*BaseInterfaceLine attribute*), 16
 storm_control (*CiscoIosInterfaceLine attribute*), 23
 SUBINT_REGEX (*CiscoRange attribute*), 32
 SUFFIX_REGEX (*CiscoRange attribute*), 32
 switchport_mode (*BaseInterfaceLine attribute*), 16
 switchport_mode (*CiscoIosInterfaceLine attribute*), 24
 switchport_nonegotiate (*BaseInterfaceLine attribute*), 17
 switchport_nonegotiate (*CiscoIosInterfaceLine attribute*), 24

T

tacacs_groups (*CiscoIosParser attribute*), 7
 tacacs_servers (*CiscoIosParser attribute*), 7
 tcp_mss (*BaseInterfaceLine attribute*), 17
 tcp_mss (*CiscoIosInterfaceLine attribute*), 24
 to_json() (*ConfigToJson method*), 25
 to_string() (*CiscoRange method*), 32
 to_yaml() (*ConfigToJson method*), 26
 trunk_allowed_vlans (*BaseInterfaceLine attribute*), 17
 trunk_allowed_vlans (*CiscoIosInterfaceLine attribute*), 24
 trunk_encapsulation (*BaseInterfaceLine attribute*), 17
 trunk_encapsulation (*CiscoIosInterfaceLine attribute*), 24
 tunnel_properties (*BaseInterfaceLine attribute*), 17
 tunnel_properties (*CiscoIosInterfaceLine attribute*), 24

U

UnsortableList (class in ccu-
utils.utils.common_utils), 31

UnsortableOrderedDict (class in ccu-
utils.utils.common_utils), 31

user_selection() (ConfigMigration method), 26

V

vlan_groups (BaseConfigParser attribute), 4

vlan_groups (CiscoIosParser attribute), 8

vlans (BaseConfigParser attribute), 4

vlans (CiscoIosParser attribute), 8

voice_vlan (BaseInterfaceLine attribute), 18

voice_vlan (CiscoIosInterfaceLine attribute), 25

vrf (BaseInterfaceLine attribute), 18

vrf (CiscoIosInterfaceLine attribute), 25

vrf_ipv4_physical_addresses (CiscoIosParser
attribute), 8

vrf_ipv4_standby_addresses (CiscoIosParser
attribute), 8

vrfns (BaseConfigParser attribute), 4

vrfns (CiscoIosParser attribute), 9