src docs

Release 0.8.18

Author

September 20, 2018

Contents

Contents:

CHAPTER 1

networkapiclient package

Submodules

networkapiclient.Ambiente module

class networkapiclient.Ambiente.**Ambiente**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: networkapiclient.GenericClient.GenericClient

add_expect_string_healthcheck(*expect_string*)

Inserts a new healthckeck_expect with only expect_string.

Parameters **expect_string** – expect_string.

Returns Dictionary with the following structure:

```
{'healthcheck_expect': {'id': < id >}}
```

Raises

- **InvalidParameterError** – The value of expect_string is invalid.
- **HealthCheckExpectJaCadastradoError** – There is already a healthcheck_expect registered with the same data.
- **HealthCheckExpectNaoExisteError** – Healthcheck_expect not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

add_healthcheck_expect(*id_ambiente*, *expect_string*, *match_list*)

Insere um novo healthckeck_expect e retorna o seu identificador.

Parameters

- **expect_string** – expect_string.
- **id_ambiente** – Identificador do ambiente lógico.
- **match_list** – match list.

Returns Dicionário com a seguinte estrutura: {'healthcheck_expect': {'id': < id >}}

Raises

- **InvalidParameterError** – O identificador do ambiente, match_lis,expect_string, são inválidos ou nulo.
- **HealthCheckExpectJaCadastradoError** – Já existe um healthcheck_expect com os mesmos dados cadastrados.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

add_ip_range(*id_environment*, *id_ip_config*)

Makes relationship of environment with ip config and returns your id.

Parameters

- **id_environment** – Environment ID.
- **id_ip_config** – IP Configuration ID.

Returns Following dictionary:

{'config_do_ambiente': {'id_config_do_ambiente': < id_config_do_ambiente >}}

Raises

- **InvalidParameterError** – Some parameter was invalid.
- **ConfigEnvironmentDuplicateError** – Error saving duplicate Environment Configuration.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

alterar(*id_ambiente*, *id_grupo_l3*, *id_ambiente_logico*, *id_divisao*, *link*, *id_filter=None*, *acl_path=None*, *ipv4_template=None*, *ipv6_template=None*, *min_num_vlan_1=None*, *max_num_vlan_1=None*, *min_num_vlan_2=None*, *max_num_vlan_2=None*, *vrf=None*)

Altera os dados de um ambiente a partir do seu identificador.

Parameters

- **id_ambiente** – Identificador do ambiente.
- **id_grupo_l3** – Identificador do grupo layer 3.
- **id_ambiente_logico** – Identificador do ambiente lógico.
- **id_divisao** – Identificador da divisão data center.
- **id_filter** – Filter identifier.
- **link** – Link
- **acl_path** – Path where the ACL will be stored
- **ipv4_template** – Template that will be used in Ipv6
- **ipv6_template** – Template that will be used in Ipv4
- **min_num_vlan_1** – Min 1 num vlan valid for this environment
- **max_num_vlan_1** – Max 1 num vlan valid for this environment
- **min_num_vlan_2** – Min 2 num vlan valid for this environment
- **max_num_vlan_2** – Max 2 num vlan valid for this environment

Returns None

Raises

- **InvalidParameterError** – O identificador do ambiente, o identificador do grupo l3, o identificador do ambiente lógico, e/ou o identificador da divisão de data center são nulos ou inválidos.
- **GrupoL3NaoExisteError** – Grupo layer 3 não cadastrado.
- **AmbienteLogicoNaoExisteError** – Ambiente lógico não cadastrado.
- **DivisaoDcNaoExisteError** – Divisão data center não cadastrada.
- **AmbienteDuplicadoError** – Ambiente com o mesmo id_grupo_l3, id_ambiente_logico e id_divisao já cadastrado.
- **AmbienteNaoExisteError** – Ambiente não cadastrado.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

associate(*environment_id*, *environment_vip_id*)

Associate a news Environment on Environment VIP and returns its identifier.

Parameters

- **environment_id** – Identifier of the Environment. Integer value and greater than zero.
- **environment_vip_id** – Identifier of the Environment VIP. Integer value and greater than zero.

Returns Following dictionary:

```
{'environment_environment_vip': {'id': < id >}}
```

Raises

- **InvalidParameterError** – The value of environment_id or environment_vip_id is invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

buscar_healthcheck_por_id(*id_healthcheck*)

Get HealthCheck by id.

Parameters **id_healthcheck** – HealthCheck ID.

Returns Following dictionary:

```
{'healthcheck_expect': {'match_list': < match_list >,
 'expect_string': < expect_string >,
 'id': < id >,
 'ambiente': < ambiente >}}
```

Raises

- **HealthCheckNaoExisteError** – HealthCheck not registered.
- **InvalidParameterError** – HealthCheck identifier is null and invalid.
- **DataBaseError** – Can't connect to networkapi database.
- **XMLError** – Failed to generate the XML response.

buscar_por_equipamento(*nome_equipamento*, *ip_equipamento*)
Obtém um ambiente a partir do ip e nome de um equipamento.

Parameters

- **nome_equipamento** – Nome do equipamento.
- **ip_equipamento** – IP do equipamento no formato XXX.XXX.XXX.XXX.

Returns Dicionário com a seguinte estrutura:

```
{'ambiente': {'id': < id_ambiente >,
'link': < link >,
'id_divisao': < id_divisao >,
'nome_divisao': < nome_divisao >,
'id_ambiente_logico': < id_ambiente_logico >,
'nome_ambiente_logico': < nome_ambiente_logico >,
'id_grupo_l3': < id_grupo_l3 >,
'nome_grupo_l3': < nome_grupo_l3 >,
'id_filter': < id_filter >,
'filter_name': < filter_name >,
'ambiente_rede': < ambiente_rede >}}
```

Raises

- **IpError** – IP não cadastrado para o equipamento.
- **InvalidParameterError** – O nome e/ou o IP do equipamento são vazios ou nulos, ou o IP é inválido.
- **EquipamentoNaoExisteError** – Equipamento não cadastrado.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

buscar_por_id(*id_ambiente*)
Obtém um ambiente a partir da chave primária (identificador).

Parameters **id_ambiente** – Identificador do ambiente.

Returns Dicionário com a seguinte estrutura:

```
{'ambiente': {'id': < id_ambiente >,
'link': < link >,
'id_divisao': < id_divisao >,
'nome_divisao': < nome_divisao >,
'id_ambiente_logico': < id_ambiente_logico >,
'nome_ambiente_logico': < nome_ambiente_logico >,
'id_grupo_l3': < id_grupo_l3 >,
'nome_grupo_l3': < nome_grupo_l3 >,
'id_filter': < id_filter >,
'filter_name': < filter_name >,
'acl_path': < acl_path >,
'ipv4_template': < ipv4_template >,
'ipv6_template': < ipv6_template >,
'ambiente_rede': < ambiente_rede >}}
```

Raises

- **AmbienteNaoExisteError** – Ambiente não cadastrado.
- **InvalidParameterError** – Identificador do ambiente é nulo ou inválido.

- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

configuration_list_all(*environment_id*)
List all prefix configurations by environment in DB

Returns Following dictionary:

```
{'lists_configuration': [{
'id': <id_ipconfig>,
'subnet': <subnet>,
'type': <type>,
'new_prefix': <new_prefix>,
}, ... ]}
```

Raises

- **InvalidValueError** – Invalid ID for Environment.
- **AmbienteNotFoundError** – Environment not registered.
- **DataBaseError** – Failed into networkapi access data base.
- **XMLError** – Networkapi failed to generate the XML response.

configuration_remove(*environment_id*, *configuration_id*)
Remove Prefix Configuration

Returns None

Raises

- **InvalidValueError** – Invalid Id for Environment or IpConfig.
- **IPConfigNotFoundError** – Ipconfig not resgistred.
- **AmbienteNotFoundError** – Environment not registered.
- **DataBaseError** – Failed into networkapi access data base.
- **XMLError** – Networkapi failed to generate the XML response.

configuration_save(*id_environment*, *network*, *prefix*, *ip_version*, *network_type*)
Add new prefix configuration

Parameters

- **id_environment** – Identifier of the Environment. Integer value and greater than zero.
- **network** – Network Ipv4 or Ipv6.
- **prefix** – Prefix 0-32 to Ipv4 or 0-128 to Ipv6.
- **ip_version** – v4 to IPv4 or v6 to IPv6
- **network_type** – type network

Returns Following dictionary:

```
{'network':{'id_environment': <id_environment>,
'id_vlan': <id_vlan>,
'network_type': <network_type>,
'network': <network>,
'prefix': <prefix>} }
```

Raises

- **ConfigEnvironmentInvalidError** – Invalid Environment Configuration or not registered.
- **InvalidValueError** – Invalid Id for environment or network or network_type or prefix.
- **AmbienteNotFoundError** – Environment not registered.
- **DataBaseError** – Failed into networkapi access data base.
- **XMLError** – Networkapi failed to generate the XML response.

delete_rule(*id_rule*)

Removes an environment rule

Parameters **id_rule** – Rule id

Returns None

Raises

- **AmbienteNaoExisteError** – Ambiente não cadastrado.
- **InvalidValueError** – Invalid parameter.
- **UserNotAuthorizedError** – Permissão negada.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

disassociate(*environment_id*, *environment_vip_id*)

Remove a relationship of Environment with EnvironmentVip.

Parameters

- **environment_id** – Identifier of the Environment. Integer value and greater than zero.
- **environment_vip_id** – Identifier of the Environment VIP. Integer value and greater than zero.

Returns Nothing

Raises

- **InvalidParameterError** – Environment/Environment VIP identifier is null and/or invalid.
- **EnvironmentNotFoundError** – Environment not registered.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **EnvironmentError** – Option vip is not associated with the environment vip
- **UserNotAuthorizedError** – User does not have authorization to make this association.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

get_all_rules(*id_env*)

Save an environment rule

Parameters **id_env** – Environment id

Returns Estrutura:

```
{ 'rules': [{'id': < id >,
'environment': < Environment Object >,
'content': < content >,
'name': < name >,
'custom': < custom > },... ]}
```

Raises

- **AmbienteNaoExisteError** – Ambiente não cadastrado.
- **UserNotAuthorizedError** – Permissão negada.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

get_blocks(*id_env*)
Get blocks by environment

Parameters **id_env** – Environment id

Returns Following dictionary:

```
{'blocks': [{'id' : <id>, 'content' : <content>},...] }
```

Raises

- **UserNotAuthorizedError** – Permissão negada.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

get_environment_template(*name*, *network*)
Get environments by template name

Parameters

- **name** – Template name.
- **network** – IPv4 or IPv6.

Returns Following dictionary:

```
{'ambiente': [divisao_dc - ambiente_logico - grupo_l3, other envs...] }
```

Raises

- **InvalidParameterError** – Invalid param.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

get_related_environment_list(*environment_vip_id*)
Get all Environment by Environment Vip.

Returns Following dictionary:

```
{'ambiente': [{ 'id': <id_environment>,
'grupo_l3': <id_group_l3>,
'grupo_l3_name': <name_group_l3>,
'ambiente_logico': <id_logical_environment>,
'ambiente_logico_name': <name_ambiente_logico>,
'divisao_dc': <id_dc_division>,
'divisao_dc_name': <name_divisao_dc>,
'filter': <id_filter>,
'filter_name': <filter_name>,
'link': <link> }, ... ]}
```

Raises

- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **DataBaseError** – Can't connect to networkapi database.
- **XMLError** – Failed to generate the XML response.

get_rule_by_pk(*id_rule*)
Get a rule by its identifier

Parameters **id_rule** – Rule identifier.

Returns Seguinte estrutura

```
{ 'rule': {'id': < id >,
  'environment': < Environment Object >,
  'content': < content >,
  'name': < name >,
  'custom': < custom > }}
```

Raises

- **AmbienteNaoExisteError** – Ambiente não cadastrado.
- **InvalidValueError** – Invalid parameter.
- **UserNotAuthorizedError** – Permissão negada.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

inserir(*id_grupo_l3*, *id_ambiente_logico*, *id_divisao*, *link*, *id_filter=None*, *acl_path=None*, *ipv4_template=None*, *ipv6_template=None*, *min_num_vlan_1=None*, *max_num_vlan_1=None*, *min_num_vlan_2=None*, *max_num_vlan_2=None*, *vrf=None*)
Insere um novo ambiente e retorna o seu identificador.

Parameters

- **id_grupo_l3** – Identificador do grupo layer 3.
- **id_ambiente_logico** – Identificador do ambiente lógico.
- **id_divisao** – Identificador da divisão data center.
- **id_filter** – Filter identifier.
- **link** – Link
- **acl_path** – Path where the ACL will be stored

- **ipv4_template** – Template that will be used in Ipv6
- **ipv6_template** – Template that will be used in Ipv4
- **min_num_vlan_1** – Min 1 num vlan valid for this environment
- **max_num_vlan_1** – Max 1 num vlan valid for this environment
- **min_num_vlan_2** – Min 2 num vlan valid for this environment
- **max_num_vlan_2** – Max 2 num vlan valid for this environment

Returns Dicionário com a seguinte estrutura: {'ambiente': {'id': < id >}}

Raises

- **InvalidParameterError** – O identificador do grupo l3, o identificador do ambiente lógico, e/ou o identificador da divisão de data center são nulos ou inválidos.
- **GrupoL3NaoExisteError** – Grupo layer 3 não cadastrado.
- **AmbienteLogicoNaoExisteError** – Ambiente lógico não cadastrado.
- **DivisaoDcNaoExisteError** – Divisão datacenter não cadastrada.
- **AmbienteDuplicadoError** – Ambiente com o mesmo id_grupo_l3, id_ambiente_logico e id_divisao já cadastrado.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

insert_with_ip_range(*id_l3_group*, *id_logical_environment*, *id_division*, *id_ip_config*, *link*, *id_filter=None*)

Insert new environment with ip config and returns your id.

Parameters

- **id_l3_group** – Layer 3 Group ID.
- **id_logical_environment** – Logical Environment ID.
- **id_division** – Data Center Division ID.
- **id_filter** – Filter identifier.
- **id_ip_config** – IP Configuration ID.
- **link** – Link.

Returns Following dictionary: {'ambiente': {'id': < id >}}

Raises

- **ConfigEnvironmentDuplicateError** – Error saving duplicate Environment Configuration.
- **InvalidParameterError** – Some parameter was invalid.
- **GrupoL3NaoExisteError** – Layer 3 Group not found.
- **AmbienteLogicoNaoExisteError** – Logical Environment not found.
- **DivisaoDcNaoExisteError** – Data Center Division not found.
- **AmbienteDuplicadoError** – Environment with this parameters already exists.
- **DataBaseError** – Networkapi failed to access the database.

- **XMLError** – Networkapi failed to generate the XML response.

list_acl_path()
Get all distinct acl paths.

Returns Dictionary with the following structure:

```
{'acl_paths': [
 < acl_path >,
 ... ]}
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

list_all()
List all environments in DB

Returns Following dictionary:

```
{'ambiente': [{ 'id': <id_environment>,
'grupo_l3': <id_group_l3>,
'grupo_l3_name': <name_group_l3>,
'ambiente_logico': <id_logical_environment>,
'ambiente_logico_name': <name_ambiente_logico>,
'divisao_dc': <id_dc_division>,
'divisao_dc_name': <name_divisao_dc>,
'filter': <id_filter>,
'filter_name': <filter_name>,
'link': <link> }, ... ]}
```

Raises DataBaseError Falha na networkapi ao acessar o banco de dados.

list_no_blocks()
List all environments in DB without blocks

Returns Following dictionary:

```
{'ambiente': [{'id': <id_environment>,
  'grupo_l3': <id_group_l3>,
  'grupo_l3_name': <name_group_l3>
  'ambiente_logico': <id_logical_environment>,
  'ambiente_logico_name': <name_ambiente_logico>
  'divisao_dc': <id_dc_division>,
  'divisao_dc_name': <name_divisao_dc>,
  'filter': <id_filter>,
  'filter_name': <filter_name>,
  'link': <link> }, ... ]}
```

Raises DataBaseError Falha na networkapi ao acessar o banco de dados.

listar(*id_divisao=None*, *id_ambiente_logico=None*)
Lista os ambientes filtrados conforme parâmetros informados.

Se os dois parâmetros têm o valor None então retorna todos os ambientes. Se o id_divisao é diferente de None então retorna os ambientes filtrados pelo valor de id_divisao. Se o id_divisao e id_ambiente_logico são diferentes de None então retorna os ambientes filtrados por id_divisao e id_ambiente_logico.

Parameters

- **id_divisao** – Identificador da divisão de data center.
- **id_ambiente_logico** – Identificador do ambiente lógico.

Returns Dicionário com a seguinte estrutura:

```
{'ambiente': [{'id': < id_ambiente >,
'link': < link >,
'id_divisao': < id_divisao >,
'nome_divisao': < nome_divisao >,
'id_ambiente_logico': < id_ambiente_logico >,
'nome_ambiente_logico': < nome_ambiente_logico >,
'id_grupo_l3': < id_grupo_l3 >,
'nome_grupo_l3': < nome_grupo_l3 >,
'id_filter': < id_filter >,
'filter_name': < filter_name >,
'ambiente_rede': < ambiente_rede >},
... demais ambientes ... ]}
```

Raises

- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

listar_healtchcheck_expect_distinct()
Get all expect_string.

Returns Dictionary with the following structure:

```
{'healthcheck_expect': [
 'expect_string': < expect_string >,
 ... demais healthcheck_expects ...]}
```

Raises

- **InvalidParameterError** – Identifier is null and invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

listar_healthcheck_expect(*id_ambiente*)
Lista os healthcheck_expect´s de um ambiente.

Parameters **id_ambiente** – Identificador do ambiente.

Returns Dicionário com a seguinte estrutura:

```
{'healthcheck_expect': [{'id': < id_healthcheck_expect >,
 'expect_string': < expect_string >,
 'match_list': < match_list >,
 'id_ambiente': < id_ambiente >},
 ... demais healthcheck_expects ...]}
```

Raises

- **InvalidParameterError** – O identificador do ambiente é nulo ou inválido.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.

- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

listar_por_equip(*equip_id*)
Lista todos os ambientes por equipamento especifico.

Returns Dicionário com a seguinte estrutura:

```
{'ambiente': {'id': < id_ambiente >,
'link': < link >,
'id_divisao': < id_divisao >,
'nome_divisao': < nome_divisao >,
'id_ambiente_logico': < id_ambiente_logico >,
'nome_ambiente_logico': < nome_ambiente_logico >,
'id_grupo_l3': < id_grupo_l3 >,
'nome_grupo_l3': < nome_grupo_l3 >,
'id_filter': < id_filter >,
'filter_name': < filter_name >,
'ambiente_rede': < ambiente_rede >}}
```

Raises

- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

remover(*id_ambiente*)
Remove um ambiente a partir de seu identificador.

Parameters **id_ambiente** – Identificador do ambiente.

Returns None

Raises

- **AmbienteNaoExisteError** – Ambiente não cadastrado.
- **AmbienteError** – Ambiente está associado a um equipamento e/ou a uma VLAN.
- **InvalidParameterError** – O identificador do ambiente é nulo ou inválido.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

save_blocks(*id_env*, *blocks*)
Save blocks from environment

Parameters

- **id_env** – Environment id
- **blocks** – Lists of blocks in order. Ex: ['content one', 'content two', ...]

Returns None

Raises

- **AmbienteNaoExisteError** – Ambiente não cadastrado.
- **InvalidValueError** – Invalid parameter.
- **UserNotAuthorizedError** – Permissão negada.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.

- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

save_rule(*name*, *id_env*, *contents*, *blocks_id*)
Save an environment rule

Parameters

- **name** – Name of the rule
- **id_env** – Environment id
- **contents** – Lists of contents in order. Ex: ['content one', 'content two', ...]
- **blocks_id** – Lists of blocks id or 0 if is as custom content. Ex: ['0', '5', '0' ...]

Returns None

Raises

- **AmbienteNaoExisteError** – Ambiente não cadastrado.
- **InvalidValueError** – Invalid parameter.
- **UserNotAuthorizedError** – Permissão negada.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

set_template(*id_environment*, *name*, *network*)
Set template value. If id_environment = 0, set '' to all environments related with the template name.

Parameters

- **id_environment** – Environment Identifier.
- **name** – Template Name.
- **network** – IPv4 or IPv6.

Returns None

Raises

- **InvalidParameterError** – Invalid param.
- **AmbienteNaoExisteError** – Ambiente não cadastrado.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

update_blocks(*id_env*, *blocks*)
Update blocks from environment

Parameters

- **id_env** – Environment id
- **blocks** – Lists of blocks in order. Ex: ['content one', 'content two', ...]

Returns None

Raises

- **AmbienteNaoExisteError** – Ambiente não cadastrado.

- **InvalidValueError** – Invalid parameter.
- **UserNotAuthorizedError** – Permissão negada.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

update_rule(*name*, *id_env*, *contents*, *blocks_id*, *id_rule*)
Save an environment rule

Parameters

- **name** – Name of the rule
- **id_env** – Environment id
- **contents** – Lists of contents in order. Ex: ['content one', 'content two', ...]
- **blocks_id** – Lists of blocks id or 0 if is as custom content. Ex: ['0', '5', '0' ...]
- **id_rule** – Rule id

Returns None

Raises

- **AmbienteNaoExisteError** – Ambiente não cadastrado.
- **InvalidValueError** – Invalid parameter.
- **UserNotAuthorizedError** – Permissão negada.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

networkapiclient.AmbienteLogico module

class networkapiclient.AmbienteLogico.**AmbienteLogico**(*networkapi_url*, *user*, *password*, *user_ldap=None*)
Bases: networkapiclient.GenericClient.GenericClient

alterar(*id_logicalenvironment*, *name*)
Change Logical Environment from by the identifier.

Parameters

- **id_logicalenvironment** – Identifier of the Logical Environment. Integer value and greater than zero.
- **name** – Logical Environment name. String with a minimum 2 and maximum of 80 characters

Returns None

Raises

- **InvalidParameterError** – The identifier of Logical Environment or name is null and invalid.
- **NomeAmbienteLogicoDuplicadoError** – There is already a registered Logical Environment with the value of name.

- **AmbienteLogicoNaoExisteError** – Logical Environment not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

inserir(*name*)

Inserts a new Logical Environment and returns its identifier.

Parameters **name** – Logical Environment name. String with a minimum 2 and maximum of 80 characters

Returns Dictionary with the following structure:

```
{'logical_environment': {'id': < id_logical_environment >}}
```

Raises

- **InvalidParameterError** – Name is null and invalid.
- **NomeAmbienteLogicoDuplicadoError** – There is already a registered Logical Environment with the value of name.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

listar()

List all Logical Environment.

Returns Dictionary with the following structure:

```
{'logical_environment':
[{'id': < id >,
'nome': < nome >}, ...more Logical Environment...]}
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

remover(*id_logicalenvironment*)

Remove Logical Environment from by the identifier.

Parameters **id_logicalenvironment** – Identifier of the Logical Environment. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – The identifier of Logical Environment is null and invalid.
- **AmbienteLogicoNaoExisteError** – Logical Environment not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.ApiGenericClient module

class networkapiclient.ApiGenericClient.**ApiGenericClient**(*networkapi_url*, *user*, *password*, *user_ldap=None*, *log_level='INFO'*)

Bases: object

Class inherited by all NetworkAPI-Client classes who implements access methods to new pattern rest networkAPI.

delete(*uri*, *data=None*)
Sends a DELETE request.

@param uri: Uri of Service API.

@raise NetworkAPIClientError: Client failed to access the API.

get(*uri*)
Sends a GET request.

@param uri: Uri of Service API. @param data: Requesting Data. Default: None

@raise NetworkAPIClientError: Client failed to access the API.

post(*uri*, *data=None*, *files=None*)
Sends a POST request.

@param uri: Uri of Service API. @param data: Requesting Data. Default: None

@raise NetworkAPIClientError: Client failed to access the API.

prepare_url(*uri*, *kwargs*)
Convert dict for URL params

put(*uri*, *data=None*)
Sends a PUT request.

@param uri: Uri of Service API. @param data: Requesting Data. Default: None

@raise NetworkAPIClientError: Client failed to access the API.

networkapiclient.ApiVipRequest module

class networkapiclient.ApiVipRequest.**ApiVipRequest**(*networkapi_url*, *user*, *password*, *user_ldap=None*, *log_level='INFO'*)

Bases: networkapiclient.ApiGenericClient.ApiGenericClient

create(*vips*)
Method to create vip's

Parameters **vips** – List containing vip's desired to be created on database

Returns None

create_vip(*vip_request_ids*)
Method to create vip request

param vip_request_ids: vip_request ids

delete(*ids*)
Method to delete vip's by their id's

Parameters **ids** – Identifiers of vip's

Returns None

delete_vip_request(*vip_request_ids*)
Method to delete vip request

param vip_request_ids: vip_request ids

deploy(*ids*)
Method to deploy vip's

Parameters **vips** – List containing vip's desired to be deployed on equipment

Returns None

get(*ids*, ***kwargs*)
Method to get vips by their id's

Parameters

- **ids** – List containing identifiers of vip's
- **include** – Array containing fields to include on response.
- **exclude** – Array containing fields to exclude on response.
- **fields** – Array containing fields to override default fields.
- **kind** – Determine if result will be detailed ('detail') or basic ('basic').

Returns Dict containing vip's

get_vip_request(*vip_request_id*)
Method to get vip request

param vip_request_id: vip_request id

get_vip_request_details(*vip_request_id*)
Method to get details of vip request

param vip_request_id: vip_request id

list_environment_by_environmet_vip(*environment_vip_id*)

option_vip_by_environmentvip(*environment_vip_id*)
List Option Vip by Environment Vip

param environment_vip_id: Id of Environment Vip

redeploy(*vips*)
Method to redeploy vip's

Parameters **vips** – List containing vip's desired to updated on equipment

Returns None

remove_vip(*vip_request_ids*)
Method to delete vip request

param vip_request_ids: vip_request ids

save_vip_request(*vip_request*)
Method to save vip request

param vip_request: vip_request object

search(***kwargs*)
Method to search vip's based on extends search.

Parameters

- **search** – Dict containing QuerySets to find vip's.
- **include** – Array containing fields to include on response.
- **exclude** – Array containing fields to exclude on response.
- **fields** – Array containing fields to override default fields.
- **kind** – Determine if result will be detailed ('detail') or basic ('basic').

Returns Dict containing vip's

search_vip_request(*search*)
Method to list vip request

param search: search

search_vip_request_details(*search*)
Method to list vip request

param search: search

undeploy(*ids*, *clean_up=0*)
Method to undeploy vip's

Parameters **vips** – List containing vip's desired to be undeployed on equipment

Returns None

update(*vips*)
Method to update vip's

Parameters **vips** – List containing vip's desired to updated

Returns None

update_vip(*vip_request*, *vip_request_id*)
Method to update vip request

param vip_request: vip_request object param vip_request_id: vip_request id

update_vip_request(*vip_request*, *vip_request_id*)
Method to update vip request

param vip_request: vip_request object param vip_request_id: vip_request id

networkapiclient.BlockRule module

class networkapiclient.BlockRule.**BlockRule**(*networkapi_url*, *user*, *password*, *user_ldap=None*)
Bases: networkapiclient.GenericClient.GenericClient

get_rule_by_id(*rule_id*)
Get rule by indentifier.

Parameters **rule_id** – Rule identifier

Returns Dictionary with the following structure:

```
{'rule': {'environment': < environment_id >,
'content': < content >,
'custom': < custom >,
'id': < id >,
'name': < name >}}
```

Raises

- **UserNotAuthorizedError** – User dont have permition.
- **InvalidParameterError** – RULE identifier is null or invalid.
- **DataBaseError** – Can't connect to networkapi database.

networkapiclient.ClientFactory module

class networkapiclient.ClientFactory.**ClientFactory**(*networkapi_url*, *user*, *password*, *user_ldap=None*, *log_level='INFO'*)

Bases: object

Factory to create entities for NetworkAPI-Client.

create_ambiente()
Get an instance of ambiente services facade.

create_ambiente_logico()
Get an instance of ambiente_logico services facade.

create_api_environment()
Get an instance of Api Environment services facade.

create_api_environment_vip()
Get an instance of Api Environment Vip services facade.

create_api_equipment()
Get an instance of Api Equipment services facade.

create_api_interface_request()
Get an instance of Api Vip Requests services facade.

create_api_ipv4()
Get an instance of Api IPv4 services facade.

create_api_ipv6()
Get an instance of Api IPv6 services facade.

create_api_network_ipv4()
Get an instance of Api Networkv4 services facade.

create_api_network_ipv6()
Get an instance of Api Networkv6 services facade.

create_api_object_group_permission()
Get an instance of Api Vip Requests services facade.

create_api_object_group_permission_general()
Get an instance of Api Vip Requests services facade.

create_api_object_type()
Get an instance of Api Vip Requests services facade.

create_api_option_vip()
Get an instance of Api Option Vip services facade.

create_api_pool()
Get an instance of Api Pool services facade.

create_api_pool_deploy()
Get an instance of Api Pool Deploy services facade.

create_api_v4_as()
Get an instance of Api As services facade.

create_api_v4_equipment()
Get an instance of Api Equipment services facade.

create_api_v4_ipv4()
Get an instance of Api V4 IPv4 services facade.

create_api_v4_ipv6()
Get an instance of Api V4 IPv6 services facade.

create_api_v4_neighbor()
Get an instance of Api Neighbor services facade.

create_api_v4_virtual_interface()
Get an instance of Api Virtual Interface services facade.

create_api_vip_request()
Get an instance of Api Vip Requests services facade.

create_api_vlan()
Get an instance of Api Vlan services facade.

create_api_vrf()
Get an instance of Api Vrf services facade.

create_apirack()
Get an instance of Api Rack Variables services facade.

create_dhcprelay_ipv4()
Get an instance of DHCPRelayIPv4 services facade.

create_dhcprelay_ipv6()
Get an instance of DHCPRelayIPv6 services facade.

create_direito_grupo_equipamento()
Get an instance of direito_grupo_equipamento services facade.

create_divisao_dc()
Get an instance of divisao_dc services facade.

create_environment_vip()
Get an instance of environment_vip services facade.

create_equipamento()
Get an instance of equipamento services facade.

create_equipamento_acesso()
Get an instance of equipamento_acesso services facade.

create_equipamento_ambiente()
Get an instance of equipamento_ambiente services facade.

create_equipamento_roteiro()
Get an instance of equipamento_roteiro services facade.

create_filter()
Get an instance of filter services facade.

create_grupo_equipamento()
Get an instance of grupo_equipamento services facade.

create_grupo_l3()
Get an instance of grupo_l3 services facade.

create_grupo_usuario()
Get an instance of grupo_usuario services facade.

create_grupo_virtual()
Get an instance of grupo_virtual services facade.

create_healthcheck()
Get an instance of Poll services facade.

create_interface()
Get an instance of interface services facade.

create_ip()
Get an instance of ip services facade.

create_log()
Get an instance of log services facade.

create_marca()
Get an instance of marca services facade.

create_modelo()
Get an instance of modelo services facade.

create_network()
Get an instance of vlan services facade.

create_option_pool()
Get an instance of option_pool services facade.

create_option_vip()
Get an instance of option_vip services facade.

create_permissao_administrativa()
Get an instance of permissao_administrativa services facade.

create_permission()
Get an instance of permission services facade.

create_pool()
Get an instance of Poll services facade.

create_rack()
Get an instance of rack services facade.

create_rackservers()
Get an instance of rackservers services facade.

create_roteiro()
Get an instance of roteiro services facade.

create_rule()
Get an instance of block rule services facade.

create_system()
Get an instance of Api System Variables services facade.

create_tipo_acesso()
Get an instance of tipo_acesso services facade.

create_tipo_equipamento()
Get an instance of tipo_equipamento services facade.

create_tipo_rede()
Get an instance of tipo_rede services facade.

create_tipo_roteiro()
Get an instance of tipo_roteiro services facade.

create_usuario()
Get an instance of usuario services facade.

create_usuario_grupo()
Get an instance of usuario_grupo services facade.

create_vip()
Get an instance of vip services facade.

create_vlan()
Get an instance of vlan services facade.

networkapiclient.Config module

class networkapiclient.Config.**IP_VERSION**

IPv4 = ('v4', 'IPv4')

IPv6 = ('v6', 'IPv6')

List = (('v4', 'IPv4'), ('v6', 'IPv6'))

networkapiclient.DireitoGrupoEquipamento module

class networkapiclient.DireitoGrupoEquipamento.**DireitoGrupoEquipamento**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: networkapiclient.GenericClient.GenericClient

alterar(*id_direito*, *leitura*, *escrita*, *alterar_config*, *exclusao*)
Altera os direitos de um grupo de usuário em um grupo de equipamento a partir do seu identificador.

Parameters

- **id_direito** – Identificador do direito grupo equipamento.
- **leitura** – Indicação de permissão de leitura ('0' ou '1').

- **escrita** – Indicação de permissão de escrita (‘0’ ou ‘1’).
- **alterar_config** – Indicação de permissão de alterar_config (‘0’ ou ‘1’).
- **exclusao** – Indicação de permissão de exclusão (‘0’ ou ‘1’).

Returns None

Raises

- **InvalidParameterError** – Pelo menos um dos parâmetros é nulo ou inválido.
- **ValorIndicacaoDireitoInvalidoError** – Valor de leitura, escrita, alterar_config e/ou exclusão inválido.
- **DireitoGrupoEquipamentoNaoExisteError** – Direito Grupo Equipamento não cadastrado.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

buscar_por_id(*id_direito*)

Obtém os direitos de um grupo de usuário e um grupo de equipamento.

Parameters **id_direito** – Identificador do direito grupo equipamento.

Returns Dicionário com a seguinte estrutura:

```
{'direito_grupo_equipamento':
{'id_grupo_equipamento': < id_grupo_equipamento >,
'exclusao': < exclusao >,
'alterar_config': < alterar_config >,
'nome_grupo_equipamento': < nome_grupo_equipamento >,
'id_grupo_usuario': < id_grupo_usuario >,
'escrita': < escrita >,
'nome_grupo_usuario': < nome_grupo_usuario >,
'id': < id >,
'leitura': < leitura >}}
```

Raises

- **InvalidParameterError** – O identificador do direito grupo equipamento é nulo ou inválido.
- **DireitoGrupoEquipamentoNaoExisteError** – Direito Grupo Equipamento não cadastrado.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

inserir(*id_grupo_usuario*, *id_grupo_equipamento*, *leitura*, *escrita*, *alterar_config*, *exclusao*)

Cria um novo direito de um grupo de usuário em um grupo de equipamento e retorna o seu identificador.

Parameters

- **id_grupo_usuario** – Identificador do grupo de usuário.
- **id_grupo_equipamento** – Identificador do grupo de equipamento.
- **leitura** – Indicação de permissão de leitura (‘0’ ou ‘1’).
- **escrita** – Indicação de permissão de escrita (‘0’ ou ‘1’).

- **alterar_config** – Indicação de permissão de alterar_config (‘0’ ou ‘1’).
- **exclusao** – Indicação de permissão de exclusão (‘0’ ou ‘1’).

Returns Dicionário com a seguinte estrutura: {‘direito_grupo_equipamento’: {‘id’: < id>}}

Raises

- **InvalidParameterError** – Pelo menos um dos parâmetros é nulo ou inválido.
- **GrupoEquipamentoNaoExisteError** – Grupo de Equipamento não cadastrado.
- **GrupoUsuarioNaoExisteError** – Grupo de Usuário não cadastrado.
- **ValorIndicacaoDireitoInvalidoError** – Valor de leitura, escrita, alterar_config e/ou exclusão inválido.
- **DireitoGrupoEquipamentoDuplicadoError** – Já existe direitos cadastrados para o grupo de usuário e grupo de equipamento informados.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

listar()

Lista todos os direitos de grupos de usuário em grupos de equipamento.

Returns Dicionário com a seguinte estrutura:

```
{'direito_grupo_equipamento':
[{'id_grupo_equipamento': < id_grupo_equipamento >,
'exclusao': < exclusao >,
'alterar_config': < alterar_config >,
'nome_grupo_equipamento': < nome_grupo_equipamento >,
'id_grupo_usuario': < id_grupo_usuario >,
'escrita': < escrita >,
'nome_grupo_usuario': < nome_grupo_usuario >,
'id': < id >,
'leitura': < leitura >}, ... demais direitos ...]}
```

Raises

- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

listar_por_grupo_equipamento(*id_grupo_equipamento*)

Lista todos os direitos de grupos de usuário em um grupo de equipamento.

Parameters **id_grupo_equipamento** – Identificador do grupo de equipamento para filtrar a pesquisa.

Returns Dicionário com a seguinte estrutura:

```
{'direito_grupo_equipamento':
[{'id_grupo_equipamento': < id_grupo_equipamento >,
'exclusao': < exclusao >,
'alterar_config': < alterar_config >,
'nome_grupo_equipamento': < nome_grupo_equipamento >,
'id_grupo_usuario': < id_grupo_usuario >,
'escrita': < escrita >,
'nome_grupo_usuario': < nome_grupo_usuario >,
```

```
'id': < id >,
'leitura': < leitura >}, ... demais direitos ...]}
```

Raises

- **InvalidParameterError** – O identificador do grupo de equipamento é nulo ou inválido.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

listar_por_grupo_usuario(*id_grupo_usuario*)

Lista todos os direitos de um grupo de usuário em grupos de equipamento.

Parameters **id_grupo_usuario** – Identificador do grupo de usuário para filtrar a pesquisa.

Returns Dicionário com a seguinte estrutura:

```
{'direito_grupo_equipamento':
[{'id_grupo_equipamento': < id_grupo_equipamento >,
'exclusao': < exclusao >,
'alterar_config': < alterar_config >,
'nome_grupo_equipamento': < nome_grupo_equipamento >,
'id_grupo_usuario': < id_grupo_usuario >,
'escrita': < escrita >,
'nome_grupo_usuario': < nome_grupo_usuario >,
'id': < id >,
'leitura': < leitura >}, ... demais direitos ...]}
```

Raises

- **InvalidParameterError** – O identificador do grupo de usuário é nulo ou inválido.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

remover(*id_direito*)

Remove os direitos de um grupo de usuário em um grupo de equipamento a partir do seu identificador.

Parameters **id_direito** – Identificador do direito grupo equipamento

Returns None

Raises

- **DireitoGrupoEquipamentoNaoExisteError** – Direito Grupo Equipamento não cadastrado.
- **InvalidParameterError** – O identificador do direito grupo equipamento é nulo ou inválido.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

networkapiclient.DivisaoDc module

class networkapiclient.DivisaoDc.**DivisaoDc**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: networkapiclient.GenericClient.GenericClient

alterar(*id_divisiondc*, *name*)

Change Division Dc from by the identifier.

Parameters

- **id_divisiondc** – Identifier of the Division Dc. Integer value and greater than zero.
- **name** – Division Dc name. String with a minimum 2 and maximum of 80 characters

Returns None

Raises

- **InvalidParameterError** – The identifier of Division Dc or name is null and invalid.
- **NomeDivisaoDcDuplicadoError** – There is already a registered Division Dc with the value of name.
- **DivisaoDcNaoExisteError** – Division Dc not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

inserir(*name*)

Inserts a new Division Dc and returns its identifier.

Parameters **name** – Division Dc name. String with a minimum 2 and maximum of 80 characters

Returns Dictionary with the following structure:

```
{'division_dc': {'id': < id_division_dc >}}
```

Raises

- **InvalidParameterError** – Name is null and invalid.
- **NomeDivisaoDcDuplicadoError** – There is already a registered Division Dc with the value of name.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

listar()

List all Division Dc.

Returns Dictionary with the following structure:

```
{'division_dc':
[{'id': < id >,
'name': < name >}, ...more Division Dc...]}
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

remover(*id_divisiondc*)
Remove Division Dc from by the identifier.

Parameters **id_divisiondc** – Identifier of the Division Dc. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – The identifier of Division Dc is null and invalid.
- **DivisaoDcNaoExisteError** – Division Dc not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.EnvironmentVIP module

class networkapiclient.EnvironmentVIP.**EnvironmentVIP**(*networkapi_url*, *user*, *password*, *user_ldap=None*)
Bases: networkapiclient.GenericClient.GenericClient

add(*finalidade_txt*, *cliente_txt*, *ambiente_p44_txt*, *description*)
Inserts a new Environment VIP and returns its identifier.

Parameters

- **finalidade_txt** – Finality. String with a maximum of 50 characters and respect [a-zA-Z_-]
- **cliente_txt** – ID Client. String with a maximum of 50 characters and respect [a-zA-Z_-]
- **ambiente_p44_txt** – Environment P44. String with a maximum of 50 characters and respect [a-zA-Z_-]

Returns Following dictionary:

```
{'environment_vip': {'id': < id >}}
```

Raises

- **InvalidParameterError** – The value of finalidade_txt, cliente_txt or ambiente_p44_txt is invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

alter(*id_environment_vip*, *finalidade_txt*, *cliente_txt*, *ambiente_p44_txt*, *description*)
Change Environment VIP from by the identifier.

Parameters

- **id_environment_vip** – Identifier of the Environment VIP. Integer value and greater than zero.
- **finalidade_txt** – Finality. String with a maximum of 50 characters and respect [a-zA-Z_-]
- **cliente_txt** – ID Client. String with a maximum of 50 characters and respect [a-zA-Z_-]
- **ambiente_p44_txt** – Environment P44. String with a maximum of 50 characters and respect [a-zA-Z_-]

Returns None

Raises

- **InvalidParameterError** – Environment VIP identifier is null and invalid.
- **InvalidParameterError** – The value of finalidade_txt, cliente_txt or ambiente_p44_txt is invalid.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

buscar_ambientep44_por_finalidade_cliente (*finalidade_txt*, *cliente_txt*)
Search ambiente_p44_txt environment vip

Returns Dictionary with the following structure:

```
{'ambiente_p44_txt':
'id':<'id_ambientevip'>,
'finalidade': <'finalidade_txt'>,
'cliente_txt: <'cliente_txt'>',
'ambiente_p44: <'ambiente_p44'>',}
```

Raises

- **InvalidParameterError** – finalidade_txt and cliente_txt is null and invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

buscar_cliente_por_finalidade (*finalidade_txt*)
Search cliente_txt environment vip

Returns Dictionary with the following structure:

```
{'cliente_txt':
'finalidade': <'finalidade_txt'>,
'cliente_txt: <'cliente_txt'>'}
```

Raises

- **InvalidParameterError** – finalidade_txt is null and invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

buscar_finalidade ()
Search finalidade_txt environment vip

Returns Dictionary with the following structure:

:: {'finalidade': 'finalidade': <'finalidade_txt'>}

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

get_vips(*id_environment_vip*)
Get to list all the VIPs related to Environment VIP from by the identifier.

Parameters **id_environment_vip** – Identifier of the Environment VIP. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{'vip_< id >':
{'id': < id >,
'validado': < validado >,
'finalidade': < finalidade >,
'cliente': < cliente >,
'ambiente': < ambiente >,
'cache': < cache >,
'metodo_bal': < método_bal >,
'persistencia': < persistencia >,
'healthcheck_type': <  healthcheck_type >,
'healthcheck': < healthcheck >,
'timeout': < timeout >,
'host': < host >,
'maxcon': < maxcon >,
'dsr': < dsr >,
'bal_ativo': < bal_ativo >,
'transbordos':{'transbordo':[< transbordo >]},
'reals':{'real':[{'real_name':< real_name >, 'real_ip':< real_ip >}]},
'portas_servicos':{'porta':[< porta >]},
'vip_criado': < vip_criado >,
'id_ip': < id_ip >,
'id_ipv6': < id_ipv6 >,
'id_healthcheck_expect': < id_healthcheck_expect >}}
```

Raises

- **InvalidParameterError** – Environment VIP identifier is null and invalid.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

list_all()
List all environment vips

Returns Following dictionary:

```
{'environment_vip': [{'id': <id>,
'finalidade_txt': <finalidade_txt>,
'cliente_txt': <cliente_txt>,
'ambiente_p44_txt': <ambiente_p44_txt> } {... other environments vip ...}]}
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

list_all_available(*id_vlan*)
List all environment vips availables

Returns Following dictionary:

```
{'environment_vip': [{'id': <id>,
'finalidade_txt': <finalidade_txt>,
'cliente_txt': <cliente_txt>,
'ambiente_p44_txt': <ambiente_p44_txt> }
{... other environments vip ...}]}
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

remove(*id_environment_vip*)
Remove Environment VIP from by the identifier.

Parameters **id_environment_vip** – Identifier of the Environment VIP. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – Environment VIP identifier is null and invalid.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **EnvironmentVipError** – There networkIPv4 or networkIPv6 associated with environment vip.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

search(*id_environment_vip=None*, *finalidade_txt=None*, *cliente_txt=None*, *ambiente_p44_txt=None*)
Search Environment VIP from by parameters.

Case the id parameter has been passed, the same it has priority over the other parameters.

Parameters

- **id_environment_vip** – Identifier of the Environment VIP. Integer value and greater than zero.
- **finalidade_txt** – Finality. String with a maximum of 50 characters and respect [a-zA-Z_-]
- **cliente_txt** – ID Client. String with a maximum of 50 characters and respect [a-zA-Z_-]
- **ambiente_p44_txt** – Environment P44. String with a maximum of 50 characters and respect [a-zA-Z_-]

Returns Following dictionary:

```
{'environment_vip':
{'id': < id >,
'finalidade_txt': < finalidade_txt >,
'finalidade': < finalidade >,
'cliente_txt': < cliente_txt >,
'ambiente_p44_txt': < ambiente_p44_txt >}}
```

Raises

- **InvalidParameterError** – The value of id_environment_vip, finalidade_txt, cliente_txt or ambiente_p44_txt is invalid.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.Equipamento module

class networkapiclient.Equipamento.**Equipamento**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: networkapiclient.GenericClient.GenericClient

add_ipv4(*id_network_ipv4*, *id_equipamento*, *descricao*)

Allocate an IP on a network to an equipment. Insert new IP for network and associate to the equipment

Parameters

- **id_network_ipv4** – ID for NetworkIPv4.
- **id_equipamento** – ID for Equipment.
- **descricao** – Description for IP.

Returns Following dictionary:

```
{'ip': {'id': < id_ip >,
'id_network_ipv4': < id_network_ipv4 >,
'oct1': < oct1 >,
'oct2': < oct2 >,
'oct3': < oct3 >,
'oct4': < oct4 >,
'descricao': < descricao >}}
```

Raises

- **InvalidParameterError** – Invalid ID for NetworkIPv4 or Equipment.
- **InvalidParameterError** – The value of description is invalid.
- **EquipamentoNaoExisteError** – Equipment not found.
- **RedeIPv4NaoExisteError** – NetworkIPv4 not found.
- **IPNaoDisponivelError** – There is no network address is available to create the VLAN.
- **ConfigEnvironmentInvalidError** – Invalid Environment Configuration or not registered
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

add_ipv6(*id_network_ipv6*, *id_equip*, *description*)

Allocate an IP on a network to an equipment. Insert new IP for network and associate to the equipment

Parameters

- **id_network_ipv6** – ID for NetworkIPv6.
- **id_equip** – ID for Equipment.
- **description** – Description for IP.

Returns Following dictionary:

```
{'ip': {'id': < id_ip >,
'id_network_ipv6': < id_network_ipv6 >,
'bloco1': < bloco1 >,
'bloco2': < bloco2 >,
'bloco3': < bloco3 >,
'bloco4': < bloco4 >,
'bloco5': < bloco5 >,
'bloco6': < bloco6 >,
'bloco7': < bloco7 >,
'bloco8': < bloco8 >,
'descricao': < descricao >}}
```

Raises

- **InvalidParameterError** – NetworkIPv6 identifier or Equipament identifier is null and invalid,
- **InvalidParameterError** – The value of description is invalid.
- **EquipamentoNaoExisteError** – Equipment not found.
- **RedeIPv6NaoExisteError** – NetworkIPv6 not found.
- **IPNaoDisponivelError** – There is no network address is available to create the VLAN.
- **ConfigEnvironmentInvalidError** – Invalid Environment Configuration or not registered
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

associar_grupo(*id_equipamento*, *id_grupo_equipamento*)

Associa um equipamento a um grupo.

Parameters

- **id_equipamento** – Identificador do equipamento.
- **id_grupo_equipamento** – Identificador do grupo de equipamento.

Returns Dicionário com a seguinte estrutura: {'equipamento_grupo':{'id': < id_equip_do_grupo >}}

Raises

- **GrupoEquipamentoNaoExisteError** – Grupo de equipamento não cadastrado.
- **InvalidParameterError** – O identificador do equipamento e/ou do grupo são nulos ou inválidos.
- **EquipamentoNaoExisteError** – Equipamento não cadastrado.
- **EquipamentoError** – Equipamento já está associado ao grupo.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

associar_ip(*id_equipamento*, *id_ip*)

Associa um IP a um equipamento.

Parameters

- **id_equipamento** – Identificador do equipamento.
- **id_ip** – Identificador do IP.

Returns Dicionário com a seguinte estrutura: {'ip_equipamento': {'id': < id_ip_do_equipamento >}}

Raises

- **EquipamentoNaoExisteError** – Equipamento não cadastrado.
- **IpNaoExisteError** – IP não cadastrado.
- **IpError** – IP já está associado ao equipamento.
- **InvalidParameterError** – O identificador do equipamento e/ou do IP são nulos ou inválidos.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

associate_ipv6(*id_equip*, *id_ipv6*)

Associates an IPv6 to a equipament.

Parameters

- **id_equip** – Identifier of the equipment. Integer value and greater than zero.
- **id_ipv6** – Identifier of the ip. Integer value and greater than zero.

Returns Dictionary with the following structure: {'ip_equipamento': {'id': < id_ip_do_equipamento >}}

Raises

- **EquipamentoNaoExisteError** – Equipment is not registered.
- **IpNaoExisteError** – IP not registered.
- **IpError** – IP is already associated with the equipment.
- **InvalidParameterError** – Identifier of the equipment and/or IP is null or invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

criar_ip(*id_vlan*, *id_equipamento*, *descricao*)

Aloca um IP em uma VLAN para um equipamento.

Insere um novo IP para a VLAN e o associa ao equipamento.

Parameters

- **id_vlan** – Identificador da vlan.
- **id_equipamento** – Identificador do equipamento.
- **descricao** – Descriçao do IP.

Returns Dicionário com a seguinte estrutura:

```
{'ip': {'id': < id_ip >,
'id_network_ipv4': < id_network_ipv4 >,
'oct1': < oct1 >,
'oct2': < oct2 >,
'oct3': < oct3 >,
```

```
'oct4': < oct4 >,
'descricao': < descricao >}}
```

Raises

- **InvalidParameterError** – O identificador da VLAN e/ou do equipamento são nulos ou inválidos.
- **EquipamentoNaoExisteError** – Equipamento não cadastrado.
- **VlanNaoExisteError** – VLAN não cadastrada.
- **IPNaoDisponivelError** – Não existe IP disponível para a VLAN informada.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

edit(*id_equip*, *nome*, *id_tipo_equipamento*, *id_modelo*, *maintenance=None*)

Change Equipment from by the identifier.

Parameters

- **id_equip** – Identifier of the Equipment. Integer value and greater than zero.
- **nome** – Equipment name. String with a minimum 3 and maximum of 30 characters
- **id_tipo_equipamento** – Identifier of the Equipment Type. Integer value and greater than zero.
- **id_modelo** – Identifier of the Model. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – The identifier of Equipment, model, equipment type or name is null and invalid.
- **EquipamentoNaoExisteError** – Equipment not registered.
- **TipoEquipamentoNaoExisteError** – Equipment Type not registered.
- **ModeloEquipamentoNaoExisteError** – Model not registered.
- **GrupoEquipamentoNaoExisteError** – Group not registered.
- **EquipamentoError** – Equipamento com o nome duplicado ou Equipamento do grupo "Equipamentos Orquestração" somente poderá ser criado com tipo igual a "Servidor Virtual".
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response

find_equips(*name*, *iexact*, *environment*, *equip_type*, *group*, *ip*, *pagination*)

Find vlans by all search parameters

Parameters

- **name** – Filter by vlan name column
- **iexact** – Filter by name will be exact?
- **environment** – Filter by environment ID related

- **equip_type** – Filter by equipment_type ID related
- **group** – Filter by equipment group ID related
- **ip** – Filter by each octs in ips related
- **pagination** – Class with all data needed to paginate

Returns Following dictionary:

```
{'equipamento': {'id': < id_vlan >,
'nome': < nome_vlan >,
'num_vlan': < num_vlan >,
'id_ambiente': < id_ambiente >,
'descricao': < descricao >,
'acl_file_name': < acl_file_name >,
'acl_valida': < acl_valida >,
'ativada': < ativada >,
'ambiente_name': < divisao_dc-ambiente_logico-grupo_l3 >
'redeipv4': [ { all networkipv4 related  } ],
'redeipv6': [ { all networkipv6 related } ] },
'total': {< total_registros >} }
```

Raises

- **InvalidParameterError** – Some parameter was invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

get_all()

Return all equipments in database

Returns Dictionary with the following structure:

:: {'equipaments': {'name' :< name_equipament >}, {... demais equipamentos ...} }

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

get_ips_by_equipment_and_environment(*equip_nome*, *id_ambiente*)

Search Group Equipment from by the identifier.

Parameters **id_egroup** – Identifier of the Group Equipment. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{'equipaments':
[{'nome': < name_equipament >, 'grupos': < id_group >,
'mark': {'id': < id_mark >, 'nome': < name_mark >},'modelo': < id_model >,
'tipo_equipamento': < id_type >,
'model': {'nome': , 'id': < id_model >, 'marca': < id_mark >},
'type': {id': < id_type >, 'tipo_equipamento': < name_type >},
'id': < id_equipment >}, ... ]}
```

Raises

- **InvalidParameterError** – Group Equipment is null and invalid.

- **GrupoEquipamentoNaoExisteError** – Group Equipment not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

get_real_related(*id_equip*)
Find reals related with equipment

Parameters **id_equip** – Identifier of equipment

Returns Following dictionary:

```
{'vips': [{'port_real': < port_real >,
'server_pool_member_id': < server_pool_member_id >,
'ip': < ip >,
'port_vip': < port_vip >,
'host_name': < host_name >,
'id_vip': < id_vip >, ...],
'equip_name': < equip_name > }}
```

Raises

- **EquipamentoNaoExisteError** – Equipment not registered.
- **InvalidParameterError** – Some parameter was invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

inserir(*name*, *id_equipment_type*, *id_model*, *id_group*, *maintenance=False*)
Inserts a new Equipment and returns its identifier

Além de inserir o equipamento, a networkAPI também associa o equipamento ao grupo informado.

Parameters

- **name** – Equipment name. String with a minimum 3 and maximum of 30 characters
- **id_equipment_type** – Identifier of the Equipment Type. Integer value and greater than zero.
- **id_model** – Identifier of the Model. Integer value and greater than zero.
- **id_group** – Identifier of the Group. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{'equipamento': {'id': < id_equipamento >},
'equipamento_grupo': {'id': < id_grupo_equipamento >}}
```

Raises

- **InvalidParameterError** – The identifier of Equipment type, model, group or name is null and invalid.
- **TipoEquipamentoNaoExisteError** – Equipment Type not registered.
- **ModeloEquipamentoNaoExisteError** – Model not registered.
- **GrupoEquipamentoNaoExisteError** – Group not registered.

- **EquipamentoError** – Equipamento com o nome duplicado ou Equipamento do grupo "Equipamentos Orquestração" somente poderá ser criado com tipo igual a "Servidor Virtual".
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response

list_all()
Return all equipments in database

Returns Dictionary with the following structure:

```
{'equipaments': {'name' :< name_equipament >}, {... demais equipamentos ...} }
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

list_by_group(*id_egroup*)
Search Group Equipment from by the identifier.

Parameters **id_egroup** – Identifier of the Group Equipment. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{'equipaments':
[{'nome': < name_equipament >, 'grupos': < id_group >,
'mark': {'id': < id_mark >, 'nome': < name_mark >},'modelo': < id_model >,
'tipo_equipamento': < id_type >,
'model': {'nome': , 'id': < id_model >, 'marca': < id_mark >},
'type': {id': < id_type >, 'tipo_equipamento': < name_type >},
'id': < id_equipment >}, ... ]}
```

Raises

- **InvalidParameterError** – Group Equipment is null and invalid.
- **GrupoEquipamentoNaoExisteError** – Group Equipment not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

listar_por_id(*id*)
Obtém um equipamento a partir do seu identificador.

Parameters **id** – ID do equipamento.

Returns Dicionário com a seguinte estrutura:

```
{'equipamento': {'id': < id_equipamento >,
'nome': < nome_equipamento >,
'id_tipo_equipamento': < id_tipo_equipamento >,
'nome_tipo_equipamento': < nome_tipo_equipamento >,
'id_modelo': < id_modelo >,
'nome_modelo': < nome_modelo >,
'id_marca': < id_marca >,
'nome_marca': < nome_marca >}}
```

Raises

- **EquipamentoNaoExisteError** – Equipamento com o id informado não cadastrado.
- **InvalidParameterError** – O nome do equipamento é nulo ou vazio.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

listar_por_nome(*nome*)

Obtém um equipamento a partir do seu nome.

Parameters **nome** – Nome do equipamento.

Returns Dicionário com a seguinte estrutura:

```
{'equipamento': {'id': < id_equipamento >,
'nome': < nome_equipamento >,
'id_tipo_equipamento': < id_tipo_equipamento >,
'nome_tipo_equipamento': < nome_tipo_equipamento >,
'id_modelo': < id_modelo >,
'nome_modelo': < nome_modelo >,
'id_marca': < id_marca >,
'nome_marca': < nome_marca >}}
```

Raises

- **EquipamentoNaoExisteError** – Equipamento com o nome informado não cadastrado.
- **InvalidParameterError** – O nome do equipamento é nulo ou vazio.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

listar_por_tipo_ambiente(*id_tipo_equipamento*, *id_ambiente*)

Lista os equipamentos de um tipo e que estão associados a um ambiente.

Parameters

- **id_tipo_equipamento** – Identificador do tipo do equipamento.
- **id_ambiente** – Identificador do ambiente.

Returns Dicionário com a seguinte estrutura:

```
{'equipamento': [{'id': < id_equipamento >,
'nome': < nome_equipamento >,
'id_tipo_equipamento': < id_tipo_equipamento >,
'nome_tipo_equipamento': < nome_tipo_equipamento >,
'id_modelo': < id_modelo >,
'nome_modelo': < nome_modelo >,
'id_marca': < id_marca >,
'nome_marca': < nome_marca >
}, ... demais equipamentos ...]}
```

Raises

- **InvalidParameterError** – O identificador do tipo de equipamento e/ou do ambiente são nulos ou inválidos.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.

- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

remove_ipv6(*id_equip*, *id_ipv6*)
Remove an IPv6 to a equipament.

Parameters

- **id_equip** – Identifier of the equipment. Integer value and greater than zero.
- **id_ipv6** – Identifier of the ip. Integer value and greater than zero.

Returns None

Raises

- **EquipamentoNaoExisteError** – Equipment is not registered.
- **IpNaoExisteError** – IP not registered.
- **IpError** – Dont IP is already associated with the equipment.
- **InvalidParameterError** – Identifier of the equipment and/or IP is null or invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

remover(*id_equipamento*)
Remove um equipamento a partir do seu identificador.

Além de remover o equipamento, a API também remove:

•O relacionamento do equipamento com os tipos de acessos.

•O relacionamento do equipamento com os roteiros.

•O relacionamento do equipamento com os IPs.

•As interfaces do equipamento.

•O relacionamento do equipamento com os ambientes.

•O relacionamento do equipamento com os grupos.

Parameters **id_equipamento** – Identificador do equipamento.

Returns None

Raises

- **EquipamentoNaoExisteError** – Equipamento não cadastrado.
- **InvalidParameterError** – O identificador do equipamento é nulo ou inválido.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

remover_grupo(*id_equipamento*, *id_grupo*)
Remove a associação de um equipamento com um grupo de equipamento.

Parameters

- **id_equipamento** – Identificador do equipamento.
- **id_grupo** – Identificador do grupo de equipamento.

Returns None

Raises

- **EquipamentoGrupoNaoExisteError** – Associação entre grupo e equipamento não cadastrada.
- **EquipamentoNaoExisteError** – Equipamento não cadastrado.
- **EquipmentDontRemoveError** – Failure to remove an association between an equipment and a group because the group is related only to a group.
- **InvalidParameterError** – O identificador do equipamento e/ou do grupo são nulos ou inválidos.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

remover_ip(*id_equipamento*, *id_ip*)

Removes association of IP and Equipment. If IP has no other association with equipments, IP is also removed.

Parameters

- **id_equipamento** – Equipment identifier
- **id_ip** – IP identifier.

Returns None

Raises

- **VipIpError** – Ip can't be removed because there is a created Vip Request.
- **IpEquipCantDissociateFromVip** – Equipment is the last balancer for created Vip Request.
- **IpError** – IP not associated with equipment.
- **InvalidParameterError** – Equipment or IP identifier is none or invalid.
- **EquipamentoNaoExisteError** – Equipment doesn't exist.
- **DataBaseError** – Networkapi failed to access database.
- **XMLError** – Networkapi failed to build response XML.

networkapiclient.EquipamentoAcesso module

class networkapiclient.EquipamentoAcesso.**EquipamentoAcesso**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: networkapiclient.GenericClient.GenericClient

edit_by_id(*id_equip_acesso*, *id_tipo_acesso*, *fqdn*, *user*, *password*, *enable_pass*)

Edit access type, fqdn, user, password and enable_pass of the relationship of equipment and access type.

Parameters

- **id_tipo_acesso** – Access type identifier.
- **id_equip_acesso** – Equipment identifier.
- **fqdn** – Equipment FQDN.
- **user** – User.

- **password** – Password.
- **enable_pass** – Enable access.

Returns None

Raises

- **InvalidParameterError** – The parameters fqdn, user, password or access type id are invalid or none.
- **EquipamentoAcessoNaoExisteError** – Equipment access type relationship doesn't exist.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

get_access(*id_access*)

Get Equipment Access by id.

Returns Dictionary with following:

```
{'equipamento_acesso':
{'id_equipamento': < id_equipamento >,
'fqdn': < fqdn >,
'user': < user >,
'pass': < pass >,
'id_tipo_acesso': < id_tipo_acesso >,
'enable_pass': < enable_pass >}}
```

inserir(*id_equipamento*, *fqdn*, *user*, *password*, *id_tipo_acesso*, *enable_pass*)

Add new relationship between equipment and access type and returns its id.

Parameters

- **id_equipamento** – Equipment identifier.
- **fqdn** – Equipment FQDN.
- **user** – User.
- **password** – Password.
- **id_tipo_acesso** – Access Type identifier.
- **enable_pass** – Enable access.

Returns Dictionary with the following: {'equipamento_acesso': {'id': < id >}}

Raises

- **EquipamentoNaoExisteError** – Equipment doesn't exist.
- **TipoAcessoNaoExisteError** – Access Type doesn't exist.
- **EquipamentoAcessoError** – Equipment and access type already associated.
- **InvalidParameterError** – The parameters equipment id, fqdn, user, password or access type id are invalid or none.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

list_by_equip(*name*)

List all equipment access by equipment name

Returns Dictionary with the following structure:

```
{'equipamento_acesso':[ {'id': <id_equiptos_access>,
'equipamento': <id_equip>,
'fqdn': <fqdn>,
'user': <user>,
'password': <pass>
'tipo_acesso': <id_tipo_acesso>,
'enable_pass': <enable_pass> }]}
```

Raises

- **InvalidValueError** – Invalid parameter.
- **EquipamentoNotFoundError** – Equipment name not found in database.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

listar()

List Equipment Access relationships.

Return only the relationships from equipments the user have write permissions in one of the equipment groups

Returns Dictionary with the following:

```
{'equipamento_acesso':
[{'id_equipamento': < id_equipamento >,
'fqdn': < fqdn >,
'user': < user >,
'pass': < pass >,
'id_tipo_acesso': < id_tipo_acesso >,
'enable_pass': < enable_pass >,
'protocolo_tipo_acesso': < protocol_tipo_acesso >},
... other equipment_access ....]}
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

remover(*id_tipo_acesso*, *id_equipamento*)

Removes relationship between equipment and access type.

Parameters

- **id_equipamento** – Equipment identifier.
- **id_tipo_acesso** – Access type identifier.

Returns None

Raises

- **EquipamentoNaoExisteError** – Equipment doesn't exist.
- **EquipamentoAcessoNaoExisteError** – Relationship between equipment and access type doesn't exist.
- **InvalidParameterError** – Equipment and/or access type id is/are invalid.
- **DataBaseError** – Networkapi failed to access the database.

- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.EquipamentoAmbiente module

class networkapiclient.EquipamentoAmbiente.**EquipamentoAmbiente**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: networkapiclient.GenericClient.GenericClient

inserir(*id_equipment*, *id_environment*, *is_router=0*)

Inserts a new Related Equipment with Environment and returns its identifier

Parameters

- **id_equipment** – Identifier of the Equipment. Integer value and greater than zero.
- **id_environment** – Identifier of the Environment. Integer value and greater than zero.
- **is_router** – Identifier of the Environment. Boolean value.

Returns Dictionary with the following structure:

```
{'equipamento_ambiente': {'id': < id_equipment_environment >}}
```

Raises

- **InvalidParameterError** – The identifier of Equipment or Environment is null and invalid.
- **AmbienteNaoExisteError** – Environment not registered.
- **EquipamentoNaoExisteError** – Equipment not registered.
- **EquipamentoAmbienteError** – Equipment is already associated with the Environment.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

remover(*id_equipment*, *id_environment*)

Remove Related Equipment with Environment from by the identifier.

Parameters

- **id_equipment** – Identifier of the Equipment. Integer value and greater than zero.
- **id_environment** – Identifier of the Environment. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – The identifier of Environment, Equipament is null and invalid.
- **EquipamentoNotFoundError** – Equipment not registered.
- **EquipamentoAmbienteNaoExisteError** – Environment not registered.
- **VipIpError** – IP-related equipment is being used for a request VIP.
- **XMLError** – Networkapi failed to generate the XML response.
- **DataBaseError** – Networkapi failed to access the database.

update(*id_equipment*, *id_environment*, *is_router*)

Remove Related Equipment with Environment from by the identifier.

Parameters

- **id_equipment** – Identifier of the Equipment. Integer value and greater than zero.
- **id_environment** – Identifier of the Environment. Integer value and greater than zero.
- **is_router** – Identifier of the Environment. Boolean value.

Returns None

Raises

- **InvalidParameterError** – The identifier of Environment, Equipament is null and invalid.
- **EquipamentoNotFoundError** – Equipment not registered.
- **EquipamentoAmbienteNaoExisteError** – Environment not registered.
- **VipIpError** – IP-related equipment is being used for a request VIP.
- **XMLError** – Networkapi failed to generate the XML response.
- **DataBaseError** – Networkapi failed to access the database.

networkapiclient.EquipamentoRoteiro module

class networkapiclient.EquipamentoRoteiro.**EquipamentoRoteiro**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: networkapiclient.GenericClient.GenericClient

inserir(*id_equipment*, *id_script*)

Inserts a new Related Equipment with Script and returns its identifier

Parameters

- **id_equipment** – Identifier of the Equipment. Integer value and greater than zero.
- **id_script** – Identifier of the Script. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{'equipamento_roteiro': {'id': < id_equipment_script >}}
```

Raises

- **InvalidParameterError** – The identifier of Equipment or Script is null and invalid.
- **RoteiroNaoExisteError** – Script not registered.
- **EquipamentoNaoExisteError** – Equipment not registered.
- **EquipamentoRoteiroError** – Equipment is already associated with the script.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response

list_by_equip(*name*)

List all equipment script by equipment name

Returns Dictionary with the following structure:

```
{'equipamento_roteiro':[ {'id': <id_equipment_script>,
'roteiro_id': <id_script>,
'roteiro_name': <name_script>,
'roteiro_desc': <desc_script>,
'tipo_roteiro_id': <id_script_type>,
'tipo_roteiro_name': <name_script_type>,
'tipo_roteiro_desc': <desc_script_type>, }],
'equipamento':
{'id': <id_equipment>,
'name': <name_equipment>,}}
```

Raises

- **InvalidParameterError** – Name is null and invalid.
- **EquipamentoNotFoundError** – Equipment name not found in database.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response

listar()

List all Related Equipment with Script.

Somente retorna os relacionamentos dos equipamentos que o usuário autenticado tem permissão de leitura em pelo menos um grupo do equipamento.

Returns Dictionary with the following structure:

```
{'equipamento_roteiro': [{'roteiro': {'nome_tipo_roteiro': < nome_tipo_roteiro >,
'descricao': < descricao >,
'nome': < nome >,
'id': < id >,
'id_tipo_roteiro': < id_tipo_roteiro >,
'descricao_tipo_roteiro': < descrição_tipo_roteiro >},
'equipamento': {'id_modelo': < id_modelo >,
'nome': < nome >,
'nome_marca': < nome_marca >,
'nome_modelo': < nome_modelo >,
'id_marca': < id_marca >,
'nome_tipo_equipamento': < nome_tipo_equipamento >,
'id_tipo_equipamento': < id_tipo_equipamento >,
'id': < id >}},
... demais equipamento_roteiro´s ...]}
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response

remover(*id_equipment*, *id_script*)

Remove Related Equipment with Script from by the identifier.

Parameters

- **id_equipment** – Identifier of the Equipment. Integer value and greater than zero.
- **id_script** – Identifier of the Script. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – The identifier of Equipment or Script is null and invalid.
- **RoteiroNaoExisteError** – Script not registered.
- **EquipamentoNaoExisteError** – Equipment not registered.
- **EquipamentoRoteiroNaoExisteError** – Equipment is not associated with the script.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response

networkapiclient.EspecificacaoGrupoVirtual module

class networkapiclient.EspecificacaoGrupoVirtual.**EspecificacaoGrupoVirtual**
Bases: object

Classe auxiliar para construção dos parâmetros das chamadas para provisionar ou remover grupo virtual.

add_equipamento(*id_tipo_equipamento*, *id_modelo*, *prefixo*, *id_grupo*, *id_vlan*, *descricao_vlan*)
Adiciona um equipamento na lista de equipamentos para operação de inserir/alterar um grupo virtual.

Parameters

- **id_tipo_equipamento** – Identificador do tipo de equipamento.
- **id_modelo** – Identificador do modelo do equipamento.
- **prefixo** – Prefixo do nome do equipamento.
- **id_grupo** – Identificador do grupo do equipamento.
- **id_vlan** – Identificador da VLAN para criar um IP para o equipamento.
- **descricao_vlan** – Descrição do IP que será criado.

Returns None

add_equipamento_remove(*id*, *id_ip*, *ids_ips_vips*)
Adiciona um equipamento na lista de equipamentos para operação de remover um grupo virtual.

Parameters

- **id** – Identificador do equipamento.
- **id_ip** – Identificador do IP do equipamento.
- **ids_ips_vips** – Lista com os identificadores de IPs criados para cada VIP e associados ao equipamento.

Returns None

add_vip(*id*, *real_name_sufixo*, *id_vlan*, *descricao_vlan*, *id_vlan_real*, *descricao_vlan_real*, *balanceadores*, *id_healthcheck_expect*, *finalidade*, *cliente*, *ambiente*, *cache*, *metodo_bal*, *persistencia*, *healthcheck_type*, *healthcheck*, *timeout*, *host*, *maxcon*, *dsr*, *bal_ativo*, *transbordos*, *portas*, *real_maps*, *id_requisicao_vip*, *areanegocio='Orquestra'*, *nome_servico='Orquestra'*, *l7_filter=None*, *reals_prioritys=None*, *reals_weights=None*)
Adiciona um VIP na lista de VIPs para operação de inserir/alterar um grupo virtual.

Os parâmetros abaixo somente são necessários para a operação de alteração:

•'real_maps': Deverá conter os reals atualmente criados para a requisição de VIP.

•'id_requisicao_vip': O identificador da requisição que deverá ser alterada.

Os parâmetros abaixo somente são necessários para a operação de inserção:

•'id_vlan': Identificador da VLAN para criar o IP do VIP.

•'descricao_vlan': Descrição do IP do VIP.

•balanceadores: Lista com os identificadores dos balanceadores que serão associados ao IP do VIP.

Parameters

- **id** – Identificador do VIP utilizado pelo sistema de orquestração.
- **real_name_sufixo** – Sufixo utilizado para criar os reals_names dos equipamentos na requisição de VIP.
- **id_vlan** – Identificador da VLAN para criar um IP para o VIP.
- **descricao_vlan** – Descrição do IP que será criado para o VIP.
- **id_vlan_real** – Identificador da VLAN para criar os IPs dos equipamentos no VIP.
- **descricao_vlan_real** – Descrição dos IPs que serão criados para os equipamentos no VIP.
- **balanceadores** – Lista com os identificadores dos balanceadores que serão associados ao IP do VIP.
- **id_healthcheck_expect** – Identificador do healthcheck_expect para criar a requisição de VIP.
- **finalidade** – Finalidade da requisição de VIP.
- **cliente** – Cliente da requisição de VIP.
- **ambiente** – Ambiente da requisição de VIP.
- **cache** – Cache da requisição de VIP.
- **metodo_bal** – Método de balanceamento da requisição de VIP.
- **persistencia** – Persistência da requisição de VIP.
- **healthcheck_type** – Healthcheck_type da requisição de VIP.
- **healthcheck** – Healthcheck da requisição de VIP.
- **timeout** – Timeout da requisição de VIP.
- **host** – Host da requisição de VIP.
- **maxcon** – Máximo número de conexão da requisição de VIP.
- **dsr** – DSR da requisição de VIP.
- **bal_ativo** – Balanceador ativo da requisição de VIP.
- **transbordos** – Lista com os IPs dos transbordos da requisição de VIP.
- **portas** – Lista com as portas da requisição de VIP.
- **real_maps** – Lista dos mapas com os dados dos reals da requisição de VIP. Cada mapa deverá ter a estrutura: {'real_name':< real_name>, 'real_ip':< real_ip>}
- **id_requisicao_vip** – Identificador da requisição de VIP para operação de alterar um grupo virtual.
- **areanegocio** – Área de negócio para a requisição de VIP (é utilizado 'Orquestra' caso seja None).

- **nome_servico** – Nome do serviço para a requisição de VIP (é utilizado 'Orquestra' caso seja None).
- **l7_filter** – Filtro L7 para a requisição de VIP.
- **reals_prioritys** – Lista dos dados de prioridade dos reals da requisição de VIP (lista de zeros, caso seja None).
- **reals_weights** – Lista dos dados de pesos dos reals da requisição de VIP (lista de zeros, caso seja None).

Returns None

add_vip_incremento(*id*)
Adiciona um vip à especificação do grupo virtual.

Parameters **id** – Identificador de referencia do VIP.

add_vip_remove(*id_ip*, *id_equipamentos*)
Adiciona um VIP na lista de VIPs para operação de remover um grupo virtual.

Parameters

- **id_ip** – Identificador do IP criado para o VIP.
- **id_equipamentos** – Lista com os identificadores dos balanceadores associados ao IP do VIP.

Returns None

get_equipamentos()
Obtem a lista de Equipamentos.

get_equipamentos_remove()
Obtem a lista de equipamentos a serem removidos.

get_vips()
Obtem a lista de VIPs.

get_vips_remove()
Obtem a lista de VIPs a serem removidos.

networkapiclient.EventLog module

class networkapiclient.EventLog.**EventLog**(*networkapi_url*, *user*, *password*, *user_ldap=None*)
Bases: networkapiclient.GenericClient.GenericClient

find_logs(*user_name*, *first_date*, *start_time*, *last_date*, *end_time*, *action*, *functionality*, *parameter*, *pagination*)
Search all logs, filtering by the given parameters. :param user_name: Filter by user_name :param first_date: Sets initial date for begin of the filter :param start_time: Sets initial time :param last_date: Sets final date :param end_time: Sets final time and ends the filter. That defines the searching gap :param action: Filter by action (Create, Update or Delete) :param functionality: Filter by class :param parameter: Filter by parameter :param pagination: Class with all data needed to paginate

Returns Following dictionary:

```
{'eventlog': {'id_usuario' : < id_user >,
'hora_evento': < hora_evento >,
'acao': < acao >,
'funcionalidade': < funcionalidade >,
```

```
'parametro_anterior': < parametro_anterior >,
'parametro_atual': < parametro_atual > }
'total' : {< total_registros >} }
```

Raises

- **InvalidParameterError** – Some parameter was invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

get_choices()

Returns a dictionary with the values used to construct the select box of actions, functionalities and users.

Returns the following dictionary:

```
{'choices_map': {'usuario' : [{ 'usuario' : < user_id >
'usuario__nome' : < nome >
'usuario__user' : < user > }]
'acao' : ['action1', 'action2', 'action3' .. 'actionN']
'funcionalidade' : ['functionality1', 'functionality2', .. 'functionalityN'] }}
```

get_version()

Returns the API's version

Returns

```
{'version_api': <version_api> }
```

networkapiclient.Filter module

class networkapiclient.Filter.**Filter**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: networkapiclient.GenericClient.GenericClient

add(*name*, *description*)

Inserts a new Filter and returns its identifier.

Parameters

- **name** – Name. String with a maximum of 100 characters and respect [a-zA-Z_-]
- **description** – Description. String with a maximum of 200 characters and respect [a-zA-Z_-]

Returns Following dictionary:

```
{'filter': {'id': < id >}}
```

Raises

- **InvalidParameterError** – The value of name or description is invalid.
- **FilterDuplicateError** – A filter named by name already exists.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

alter(*id_filter*, *name*, *description*)
Change Filter by the identifier.

Parameters

- **id_filter** – Identifier of the Filter. Integer value and greater than zero.
- **name** – Name. String with a maximum of 50 characters and respect [a-zA-Z_-]
- **description** – Description. String with a maximum of 50 characters and respect [a-zA-Z_-]

Returns None

Raises

- **InvalidParameterError** – Filter identifier is null and invalid.
- **InvalidParameterError** – The value of name or description is invalid.
- **FilterNotFoundError** – Filter not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

associate(*et_id*, *id_filter*)
Create a relationship between Filter and TipoEquipamento.

Parameters

- **et_id** – Identifier of TipoEquipamento. Integer value and greater than zero.
- **id_filter** – Identifier of Filter. Integer value and greater than zero.

Returns Following dictionary:

```
{'equiptype_filter_xref': {'id': < id_equiptype_filter_xref >} }
```

Raises

- **InvalidParameterError** – TipoEquipamento/Filter identifier is null and/or invalid.
- **TipoEquipamentoNaoExisteError** – TipoEquipamento not registered.
- **FilterNotFoundError** – Filter not registered.
- **FilterEqTypeAssociationError** – TipoEquipamento and Filter already associated.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

dissociate(*id_filter*, *id_eq_type*)
Removes relationship between Filter and TipoEquipamento.

Parameters

- **id_filter** – Identifier of Filter. Integer value and greater than zero.
- **id_eq_type** – Identifier of TipoEquipamento. Integer value, greater than zero.

Returns None

Raises

- **FilterNotFoundError** – Filter not registered.

- **TipoEquipamentoNotFoundError** – TipoEquipamento not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

get(*id_filter*)
Get filter by id.

Parameters **id_filter** – Identifier of the Filter. Integer value and greater than zero.

Returns Following dictionary:

```
{'filter': {'id': < id >,
'name': < name >,
'description': < description >}}
```

Raises

- **InvalidParameterError** – The value of id_filter is invalid.
- **FilterNotFoundError** – Filter not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

list_all()
List all filters

Returns Following dictionary:

```
{'filter': [{'id': <id>,
'name': <name>,
'description': <description>,
'equip_types': [<TipoEquipamento>,
{...demais TipoEquipamento's...}]}
{... demais filters ...}] }
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

remove(*id_filter*)
Remove Filter by the identifier.

Parameters **id_filter** – Identifier of the Filter. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – Filter identifier is null and invalid.
- **FilterNotFoundError** – Filter not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.GenericClient module

class networkapiclient.GenericClient.**GenericClient**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: object

Class inherited by all NetworkAPI-Client classes who implements access methods to networkAPI.

get_error(*xml*)

Obtem do XML de resposta, o código e a descrição do erro.

O XML corresponde ao corpo da resposta HTTP de código 500.

Parameters **xml** – XML contido na resposta da requisição HTTP.

Returns Tupla com o código e a descrição do erro contido no XML: (< codigo_erro>, < descricao_erro>)

get_url(*postfix*)

Constroe e retorna a URL completa para acesso à networkAPI.

Parameters **postfix** – Posfixo a ser colocado na URL básica. Ex: /ambiente

Returns URL completa.

response(*code*, *xml*, *force_list=None*)

Cria um dicionário com os dados de retorno da requisição HTTP ou lança uma exceção correspondente ao erro ocorrido.

Se a requisição HTTP retornar o código 200 então este método retorna o dicionário com os dados da resposta.

Se a requisição HTTP retornar um código diferente de 200 então este método lança uma exceção correspondente ao erro.

Todas as exceções lançadas por este método deverão herdar de NetworkAPIClientError.

Parameters

- **code** – Código de retorno da requisição HTTP.
- **xml** – XML ou descrição (corpo) da resposta HTTP.
- **force_list** – Lista com as tags do XML de resposta que deverão ser transformadas obrigatoriamente em uma lista no dicionário de resposta.

Returns Dicionário com os dados da resposta HTTP retornada pela networkAPI.

submit(*map*, *method*, *postfix*)

Realiza um requisição HTTP para a networkAPI.

Parameters

- **map** – Dicionário com os dados para gerar o XML enviado no corpo da requisição HTTP.
- **method** – Método da requisição HTTP ('GET', 'POST', 'PUT' ou 'DELETE').
- **postfix** – Posfixo a ser colocado na URL básica de acesso à networkAPI. Ex: /ambiente

Returns Tupla com o código e o corpo da resposta HTTP: (< codigo>, < descricao>)

Raises NetworkAPIClientError Erro durante a chamada HTTP para acesso à networkAPI.

networkapiclient.GrupoEquipamento module

class networkapiclient.GrupoEquipamento.**GrupoEquipamento**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: networkapiclient.GenericClient.GenericClient

alterar(*id_egrupo*, *nome*)

Altera os dados de um grupo de equipamento a partir do seu identificador.

Parameters

- **id_egrupo** – Identificador do grupo de equipamento.
- **nome** – Nome do grupo de equipamento.

Returns None

Raises

- **InvalidParameterError** – O identificador e/ou o nome do grupo são nulos ou inválidos.
- **GrupoEquipamentoNaoExisteError** – Grupo de equipamento não cadastrado.
- **NomeGrupoEquipamentoDuplicadoError** – Nome do grupo de equipamento duplicado.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

associa_equipamento(*id_equip*, *id_grupo_equipamento*)

Associa um equipamento a um grupo.

Parameters

- **id_equip** – Identificador do equipamento.
- **id_grupo_equipamento** – Identificador do grupo de equipamento.

Returns Dicionário com a seguinte estrutura: {'equipamento_grupo':{'id': < id_equip_do_grupo >}}

Raises

- **GrupoEquipamentoNaoExisteError** – Grupo de equipamento não cadastrado.
- **InvalidParameterError** – O identificador do equipamento e/ou do grupo são nulos ou inválidos.
- **EquipamentoNaoExisteError** – Equipamento não cadastrado.
- **EquipamentoError** – Equipamento já está associado ao grupo.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

inserir(*nome*)

Insere um novo grupo de equipamento e retorna o seu identificador.

Parameters **nome** – Nome do grupo de equipamento.

Returns Dicionário com a seguinte estrutura: {'grupo': {'id': < id >}}

Raises

- **InvalidParameterError** – Nome do grupo é nulo ou vazio.
- **NomeGrupoEquipamentoDuplicadoError** – Nome do grupo de equipmaneto duplicado.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

listar()

Lista todos os grupos de equipamentos.

Returns Dicionário com a seguinte estrutura:

```
{'grupo':
[{'id': < id >,
'nome': < nome >},
... demais grupos ...]}
```

Raises

- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

listar_por_equip(*equip_id*)

Lista todos os grupos de equipamentos por equipamento especifico.

Returns Dicionário com a seguinte estrutura:

```
{'grupo':  [{'id': < id >,
'nome': < nome >},
... demais grupos ...]}
```

Raises

- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

remove(*id_equipamento*, *id_egrupo*)

Remove a associacao de um grupo de equipamento com um equipamento a partir do seu identificador.

Parameters

- **id_egrupo** – Identificador do grupo de equipamento.
- **id_equipamento** – Identificador do equipamento.

Returns None

Raises

- **EquipamentoNaoExisteError** – Equipamento não cadastrado.
- **GrupoEquipamentoNaoExisteError** – Grupo de equipamento não cadastrado.
- **InvalidParameterError** – O identificador do grupo é nulo ou inválido.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

remover(*id_egrupo*)
Remove um grupo de equipamento a partir do seu identificador.

Parameters **id_egrupo** – Identificador do grupo de equipamento.

Returns None

Raises

- **GrupoEquipamentoNaoExisteError** – Grupo de equipamento não cadastrado.
- **InvalidParameterError** – O identificador do grupo é nulo ou inválido.
- **GroupDontRemoveError** –
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

search(*id_egroup*)
Search Group Equipament from by the identifier.

Parameters **id_egroup** – Identifier of the Group Equipament. Integer value and greater than zero.

Returns Following dictionary:

```
{'group_equipament': {'id': < id_egrupo >,
'nome': < nome >} }
```

Raises

- **InvalidParameterError** – Group Equipament identifier is null and invalid.
- **GrupoEquipamentoNaoExisteError** – Group Equipament not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.GrupoL3 module

class networkapiclient.GrupoL3.**GrupoL3**(*networkapi_url*, *user*, *password*, *user_ldap=None*)
Bases: networkapiclient.GenericClient.GenericClient

alterar(*id_groupl3*, *name*)
Change Group L3 from by the identifier.

Parameters

- **id_groupl3** – Identifier of the Group L3. Integer value and greater than zero.
- **name** – Group L3 name. String with a minimum 2 and maximum of 80 characters

Returns None

Raises

- **InvalidParameterError** – The identifier of Group L3 or name is null and invalid.
- **NomeGrupoL3DuplicadoError** – There is already a registered Group L3 with the value of name.
- **GrupoL3NaoExisteError** – Group L3 not registered.

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

inserir(*name*)

Inserts a new Group L3 and returns its identifier.

Parameters **name** – Group L3 name. String with a minimum 2 and maximum of 80 characters

Returns Dictionary with the following structure:

```
{'group_l3': {'id': < id_group_l3 >}}
```

Raises

- **InvalidParameterError** – Name is null and invalid.
- **NomeGrupoL3DuplicadoError** – There is already a registered Group L3 with the value of name.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

listar()

List all Group L3.

Returns Dictionary with the following structure:

```
{'group_l3': [{'id': < id >,
'name': < name >},
...more Group L3...]}
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

remover(*id_groupl3*)

Remove Group L3 from by the identifier.

Parameters **id_groupl3** – Identifier of the Group L3. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – The identifier of Group L3 is null and invalid.
- **GrupoL3NaoExisteError** – Group L3 not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.GrupoUsuario module

class networkapiclient.GrupoUsuario.**GrupoUsuario**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: networkapiclient.GenericClient.GenericClient

alterar(*id_user_group*, *name*, *read*, *write*, *edit*, *remove*)
Edit user group data from its identifier.

Parameters

- **id_user_group** – User group id.
- **name** – User group name.
- **read** – If user group has read permission ('S' ou 'N').
- **write** – If user group has write permission ('S' ou 'N').
- **edit** – If user group has edit permission ('S' ou 'N').
- **remove** – If user group has remove permission ('S' ou 'N').

Returns None

Raises

- **NomeGrupoUsuarioDuplicadoError** – User group name already exists.
- **ValorIndicacaoPermissaoInvalidoError** – Read, write, edit or remove value is invalid.
- **GrupoUsuarioNaoExisteError** – User Group not found.
- **InvalidParameterError** – At least one of the parameters is invalid or none.
- **DataBaseError** – Networkapi failed to access database.
- **XMLError** – Networkapi fails generating response XML.

inserir(*name*, *read*, *write*, *edit*, *remove*)
Insert new user group and returns its identifier.

Parameters

- **name** – User group's name.
- **read** – If user group has read permission ('S' ou 'N').
- **write** – If user group has write permission ('S' ou 'N').
- **edit** – If user group has edit permission ('S' ou 'N').
- **remove** – If user group has remove permission ('S' ou 'N').

Returns Dictionary with structure: {'user_group': {'id': < id >}}

Raises

- **InvalidParameterError** – At least one of the parameters is invalid or none..
- **NomeGrupoUsuarioDuplicadoError** – User group name already exists.
- **ValorIndicacaoPermissaoInvalidoError** – Read, write, edit or remove value is invalid.
- **DataBaseError** – Networkapi failed to access database.
- **XMLError** – Networkapi fails generating response XML.

listar()
List all user groups.

Returns Dictionary with structure:

```
{'user_group':  [{'escrita': < escrita >,
'nome': < nome >,
'exclusao': < exclusao >,
'edicao': < edicao >,
'id': < id >,
'leitura': < leitura >},
... other groups ...]}
```

Raises

- **DataBaseError** – Networkapi failed to access database.
- **XMLError** – Networkapi fails generating response XML.

remover(*id_user_group*)

Removes a user group by its id.

Parameters **id_user_group** – User Group's identifier. Valid integer greater than zero.

Returns None

Raises

- **GrupoUsuarioNaoExisteError** – User Group not found.
- **InvalidParameterError** – User Group id is invalid or none.
- **DataBaseError** – Networkapi failed to access database.
- **XMLError** – Networkapi fails generating response XML.

search(*id_ugroup*)

Search Group User by its identifier.

Parameters **id_ugroup** – Identifier of the Group User. Integer value and greater than zero.

Returns Following dictionary:

```
{'user_group': {'escrita': < escrita >,
'nome': < nome >,
'exclusao': < exclusao >,
'edicao': < edicao >,
'id': < id >,
'leitura': < leitura >}}
```

Raises

- **InvalidParameterError** – Group User identifier is none or invalid.
- **GrupoUsuarioNaoExisteError** – Group User not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.GrupoVirtual module

Title: Infrastructure NetworkAPI Author: globo.com / TQI Copyright: (c) 2009 globo.com todos os direitos reservados.

class networkapiclient.GrupoVirtual.**GrupoVirtual**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: networkapiclient.GenericClient.GenericClient

provisionar(*equipamentos*, *vips*)

Realiza a inserção ou alteração de um grupo virtual para o sistema de Orquestração de VM.

Parameters

- **equipamentos** – Lista de equipamentos gerada pelo método "add_equipamento" da classe "EspecificacaoGrupoVirtual".
- **vips** – Lista de VIPs gerada pelo método "add_vip" da classe "EspecificacaoGrupoVirtual".

Returns Retorno da operação de inserir ou alterar um grupo virtual:

```
{'equipamentos': {'equipamento': [{'id': < id>,
'nome': < nome>,
'ip':  {'id': < id>,
'id_vlan': < id_vlan>,
'oct4': < oct4>,
'oct3': < oct3>,
'oct2': < oct2>,
'oct1': < oct1>,
'descricao': < descricao>},
'vips': {'vip': [{'id': < id>,
'ip': {'id': < id>,
'id_vlan': < id_vlan>,
'oct4': < oct4>,
'oct3': < oct3>,
'oct2': < oct2>,
'oct1': < oct1>,
'descricao': < descricao>}}, ... demais vips ...]}}, ... demais equipamentos...]},
'vips': {'vip': [{'id': < id>,
'ip': {'id': < id>,
'id_vlan': < id_vlan>,
'oct4': < oct4>,
'oct3': < oct3>,
'oct2': < oct2>,
'oct1': < oct1>,
'descricao': < descricao>},
'requisicao_vip': {'id': < id>}}, ... demais vips...]}}

{'equipamentos': {'equipamento': [{'id': < id>,
'nome': < nome>,
'ip': {'id': < id>,
'id_vlan': < id_vlan>,
'oct4': < oct4>,
'oct3': < oct3>,
'oct2': < oct2>,
'oct1': < oct1>,
'descricao': < descricao>},
'vips': {'vip': [{'id': < id>,
'ip': {'id': < id>,
'id_vlan': < id_vlan>,
'oct4': < oct4>,
'oct3': < oct3>,
'oct2': < oct2>,
'oct1': < oct1>,
```

```
'descricao': < descricao>}}, ... demais vips ...]}}, ... demais equipamentos...]},
'vips': {'vip': [{'id': < id>,
'requisicao_vip': {'id': < id>}}, ... demais vips...]}}
```

Raises

- **InvalidParameterError** – Algum dado obrigatório não foi informado nas listas ou possui um valor inválido.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **VlanNaoExisteError** – VLAN não cadastrada.
- **EquipamentoNaoExisteError** – Equipamento não cadastrado.
- **IPNaoDisponivelError** – Não existe IP disponível para a VLAN informada.
- **TipoEquipamentoNaoExisteError** – Tipo de equipamento não cadastrado.
- **ModeloEquipamentoNaoExisteError** – Modelo do equipamento não cadastrado.
- **GrupoEquipamentoNaoExisteError** – Grupo de equipamentos não cadastrado.
- **EquipamentoError** – Equipamento com o nome duplicado ou Equipamento do grupo "Equipamentos Orquestração" somente poderá ser criado com tipo igual a "Servidor Virtual".
- **IpNaoExisteError** – IP não cadastrado.
- **IpError** – IP já está associado ao equipamento.
- **VipNaoExisteError** – Requisição de VIP não cadastrada.
- **HealthCheckExpectNaoExisteError** – Healthcheck_expect não cadastrado.

remover_provisionamento(*equipamentos*, *vips*)

Remove o provisionamento de um grupo virtual para o sistema de Orquestração VM.

Parameters

- **equipamentos** – Lista de equipamentos gerada pelo método "add_equipamento_remove" da classe "EspecificacaoGrupoVirtual".
- **vips** – Lista de VIPs gerada pelo método "add_vip_remove" da classe "EspecificacaoGrupoVirtual".

Returns None

Raises

- **InvalidParameterError** – Algum dado obrigatório não foi informado nas listas ou possui um valor inválido.
- **IpNaoExisteError** – IP não cadastrado.
- **EquipamentoNaoExisteError** – Equipamento não cadastrado.
- **IpError** – IP não está associado ao equipamento.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao ler o XML de requisição ou gerar o XML de resposta.

networkapiclient.Interface module

class networkapiclient.Interface.**Interface**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: networkapiclient.GenericClient.GenericClient

alterar(*id_interface*, *nome*, *protegida*, *descricao*, *id_ligacao_front*, *id_ligacao_back*, *tipo=None*, *vlan=None*)

Edit an interface by its identifier.

Equipment identifier is not changed.

Parameters

- **nome** – Interface name.
- **protegida** – Indication of protected ('0' or '1').
- **descricao** – Interface description.
- **id_ligacao_front** – Front end link interface identifier.
- **id_ligacao_back** – Back end link interface identifier.
- **id_interface** – Interface identifier.

Returns None

Raises

- **InvalidParameterError** – The parameters interface id, nome and protegida are none or invalid.
- **NomeInterfaceDuplicadoParaEquipamentoError** – There is already an interface with this name for this equipment.
- **InterfaceNaoExisteError** – Front link interface and/or back link interface doesn't exist.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

associar_ambiente(*ambiente*, *interface*)

delete_channel(*channel_name*)

dissociar(*interface*)

editar_channel(*id_channel*, *nome*, *lacp*, *int_type*, *vlan*, *envs*, *ids_interface*)

get_by_id(*id_interface*)

Get an interface by id.

Parameters **id_interface** – Interface identifier.

Returns Following dictionary:

```
{'interface': {'id': < id >,
'interface': < interface >,
'descricao': < descricao >,
'protegida': < protegida >,
'tipo_equip': < id_tipo_equipamento >,
'equipamento': < id_equipamento >,
'equipamento_nome': < nome_equipamento >
'ligacao_front': < id_ligacao_front >,
'nome_ligacao_front': < interface_name >,
```

```
'nome_equip_l_front': < equipment_name >,
'ligacao_back': < id_ligacao_back >,
'nome_ligacao_back': < interface_name >,
'nome_equip_l_back': < equipment_name > }}
```

Raises

- **InvalidParameterError** – Interface identifier is invalid or none.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

get_env_by_id(*id_interface*)

get_interface_by_channel(*channel_name*, *equip_name*)

inserir(*nome*, *protegida*, *descricao*, *id_ligacao_front*, *id_ligacao_back*, *id_equipamento*, *tipo=None*, *vlan=None*)

Insert new interface for an equipment.

Parameters

- **nome** – Interface name.
- **protegida** – Indication of protected ('0' or '1').
- **descricao** – Interface description.
- **id_ligacao_front** – Front end link interface identifier.
- **id_ligacao_back** – Back end link interface identifier.
- **id_equipamento** – Equipment identifier.

Returns Dictionary with the following: {'interface': {'id': < id >}}

Raises

- **EquipamentoNaoExisteError** – Equipment does not exist.
- **InvalidParameterError** – The parameters nome, protegida and/or equipment id are none or invalid.
- **NomeInterfaceDuplicadoParaEquipamentoError** – There is already an interface with this name for this equipment.
- **InterfaceNaoExisteError** – Front link interface and/or back link interface doesn't exist.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

inserir_channel(*interfaces*, *nome*, *lacp*, *int_type*, *vlan*, *envs*)

list_all_by_equip(*id_equipamento*)

List all interfaces of an equipment.

Parameters **id_equipamento** – Equipment identifier.

Returns Following dictionary:

```
{'interfaces': [ {'id': < id >,
'interface': < interface >,
'descricao': < descricao >,
'protegida': < protegida >,
```

```
'tipo_equip': < id_tipo_equipamento >,
'equipamento': < id_equipamento >,
'equipamento_nome': < nome_equipamento >
'ligacao_front': < id_ligacao_front >,
'nome_ligacao_front': < interface_name >,
'nome_equip_l_front': < equipment_name >,
'ligacao_back': < id_ligacao_back >,
'nome_ligacao_back': < interface_name >,
'nome_equip_l_back': < equipment_name > }, ... other interfaces ...]}
```

Raises

- **InvalidParameterError** – Equipment identifier is invalid or none.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

list_all_interface_types()

list_available_interfaces(*channel_name*, *id_equip*)

list_connections(*nome_interface*, *id_equipamento*)
List interfaces linked to back and front of specified interface.

Parameters

- **nome_interface** – Interface name.
- **id_equipamento** – Equipment identifier.

Returns Dictionary with the following:

```
{'interfaces':[ {'id': < id >,
'interface': < nome >,
'descricao': < descricao >,
'protegida': < protegida >,
'equipamento': < id_equipamento >,
'tipo_equip': < id_tipo_equipamento >,
'ligacao_front': < id_ligacao_front >,
'nome_ligacao_front': < interface_name >,
'nome_equip_l_front': < equipment_name >,
'ligacao_back': < id_ligacao_back >,
'nome_ligacao_back': < interface_name >,
'nome_equip_l_back': < equipment_name > }, ... other interfaces ...]}
```

Raises

- **InterfaceNaoExisteError** – Interface doesn't exist or is not associated with this equipment.
- **EquipamentoNaoExisteError** – Equipment doesn't exist.
- **InvalidParameterError** – Interface name and/or equipment identifier are none or invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

listar_ligacoes(*nome_interface*, *id_equipamento*)
List interfaces linked to back and front of specified interface.

Parameters

- **nome_interface** – Interface name.
- **id_equipamento** – Equipment identifier.

Returns Dictionary with the following:

```
{'interface': [{'protegida': < protegida >,
'nome': < nome >,
'id_ligacao_front': < id_ligacao_front >,
'id_equipamento': < id_equipamento >,
'id': < id >,
'descricao': < descricao >,
'id_ligacao_back': < id_ligacao_back >}, ... other interfaces ...]}
```

Raises

- **InterfaceNaoExisteError** – Interface doesn't exist or is not associated with this equipment.
- **EquipamentoNaoExisteError** – Equipment doesn't exist.
- **InvalidParameterError** – Interface name and/or equipment identifier are none or invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

listar_por_equipamento(*id_equipamento*)

List all interfaces of an equipment.

Parameters **id_equipamento** – Equipment identifier.

Returns Dictionary with the following:

```
{'interface':
[{'protegida': < protegida >,
'nome': < nome >,
'id_ligacao_front': < id_ligacao_front >,
'id_equipamento': < id_equipamento >,
'id': < id >,
'descricao': < descricao >,
'id_ligacao_back': < id_ligacao_back >}, ... other interfaces ...]}
```

Raises

- **InvalidParameterError** – Equipment identifier is invalid or none.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

listar_switch_router(*id_equipamento*)

remove_connection(*id_interface*, *back_or_front*)

Remove a connection between two interfaces

Parameters

- **id_interface** – One side of relation
- **back_or_front** – This side of relation is back(0) or front(1)

Returns None

Raises

- **InterfaceInvalidBackFrontError** – Front or Back of interfaces not match to remove connection
- **InvalidParameterError** – Interface id or back or front indicator is none or invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

remover(*id_interface*)
Remove an interface by its identifier.

Parameters **id_interface** – Interface identifier.

Returns None

Raises

- **InterfaceNaoExisteError** – Interface doesn't exist.
- **InterfaceError** – Interface is linked to another interface.
- **InvalidParameterError** – The interface identifier is invalid or none.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.Ip module

class networkapiclient.Ip.**Ip**(*networkapi_url*, *user*, *password*, *user_ldap=None*)
Bases: networkapiclient.GenericClient.GenericClient

assoc_ipv4(*id_ip*, *id_equip*, *id_net*)
Associate an IP4 with equipment.

Parameters

- **id_ip** – IPv4 identifier.
- **id_equip** – Equipment identifier. Integer value and greater than zero.
- **id_net** – Network identifier. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – IPv4, Equipment or Network identifier is none or invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

assoc_ipv6(*id_ip*, *id_equip*, *id_net*)
Associate an IP6 with equipment.

Parameters

- **id_ip** – IPv6 identifier.
- **id_equip** – Equipment identifier. Integer value and greater than zero.

- **id_net** – Network identifier. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – IPv6, Equipment or Network identifier is none or invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

buscar_por_ip_ambiente(*ip*, *id_environment*)

Get IP with an associated environment.

Parameters

- **ip** – IP address in the format x1.x2.x3.x4.
- **id_environment** – Identifier of the environment. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{'ip': {'id': < id >,
'id_vlan': < id_vlan >,
'oct4': < oct4 >,
'oct3': < oct3 >,
'oct2': < oct2 >,
'oct1': < oct1 >,
'descricao': < descricao > }}
```

Raises

- **IpNaoExisteError** – IP is not registered or not associated with environment.
- **InvalidParameterError** – The environment identifier and/or IP is/are null or invalid.
- **DataBaseError** – Networkapi failed to access the database.

check_vip_ip(*ip*, *id_evip*)

Get a Ipv4 or Ipv6 for Vip request

Parameters **ip** – IPv4 or Ipv6. 'xxx.xxx.xxx.xxx or xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx'

Returns Dictionary with the following structure:

```
{'ip': {'ip': < ip - octs for ipv4, blocks for ipv6 - >,
'id': <id>,
'network4 or network6'}}.
```

Raises

- **IpNaoExisteError** – Ipv4 or Ipv6 not found.
- **EnvironemntVipNotFoundError** – Vip environment not found.
- **IPNaoDisponivelError** – Ip not available for Vip Environment.
- **UserNotAuthorizedError** – User dont have permission to perform operation.
- **InvalidParameterError** – Ip string or vip environment is none or invalid.
- **XMLError** – Networkapi failed to generate the XML response.
- **DataBaseError** – Networkapi failed to access the database.

delete_ip4(*id_ip*)
Delete an IP4

Parameters **id_ip** – Ipv4 identifier. Integer value and greater than zero.

Returns None

Raises

- **IpNotFoundError** – IP is not registered.
- **DataBaseError** – Networkapi failed to access the database.

delete_ip6(*id_ip*)
Delete an IP6

Parameters **id_ip** – Ipv6 identifier. Integer value and greater than zero.

Returns None

Raises

- **IpNotFoundError** – IP is not registered.
- **DataBaseError** – Networkapi failed to access the database.

edit_ipv4(*ip4*, *descricao*, *id_ip*)
Edit a IP4

Parameters

- **ip4** – An IP4 available to save in format x.x.x.x.
- **id_ip** – IP identifier. Integer value and greater than zero.
- **descricao** – IP description.

Returns None

edit_ipv6(*ip6*, *descricao*, *id_ip*)
Edit a IP6

Parameters

- **ip6** – An IP6 available to save in format xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx.
- **descricao** – IP description.
- **id_ip** – Ipv6 identifier. Integer value and greater than zero.

Returns None

find_ip4_by_id(*id_ip*)
Get an IP by ID

Parameters **id_ip** – IP identifier. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{ ips { id: <id_ip4>,
oct1: <oct1>,
oct2: <oct2>,
oct3: <oct3>,
oct4: <oct4>,
equipamento: [ {all equipamentos related} ] ,
descricao: <descricao>} }
```

Raises

- **IpNotAvailableError** – Network dont have available IP.
- **NetworkIPv4NotFoundError** – Network was not found.
- **UserNotAuthorizedError** – User dont have permission to perform operation.
- **InvalidParameterError** – Ip identifier is none or invalid.
- **XMLError** – Networkapi failed to generate the XML response.
- **DataBaseError** – Networkapi failed to access the database.

find_ip4_by_network(*id_network*)
List IPv4 from network.

Parameters **id_network** – Networkv ipv4 identifier. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{'ip': {'id': < id >,
'id_vlan': < id_vlan >,
'oct4': < oct4 >,
'oct3': < oct3 >,
'oct2': < oct2 >,
'oct1': < oct1 >,
'descricao': < descricao >
'equipamento': [ { all name equipamentos related } ], }}
```

Raises

- **IpNaoExisteError** – Network does not have any ips.
- **InvalidParameterError** – Network identifier is none or invalid.
- **DataBaseError** – Networkapi failed to access the database.

find_ip6_by_id(*id_ip*)
Get an IP6 by ID

Parameters **id_ip** – IP6 identifier. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{'ip': {'id': < id >,
'block1': <block1>,
'block2': <block2>,
'block3': <block3>,
'block4': <block4>,
'block5': <block5>,
'block6': <block6>,
'block7': <block7>,
'block8': <block8>,
'descricao': < description >,
'equipamento': [ { all name equipamentos related} ], }}
```

Raises

- **IpNotAvailableError** – Network dont have available IPv6.
- **NetworkIPv4NotFoundError** – Network was not found.
- **UserNotAuthorizedError** – User dont have permission to perform operation.
- **InvalidParameterError** – IPv6 identifier is none or invalid.

- **XMLError** – Networkapi failed to generate the XML response.
- **DataBaseError** – Networkapi failed to access the database.

find_ip6_by_network(*id_network*)
List IPv6 from network.

Parameters **id_network** – Network ipv6 identifier. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{'ip': {'id': < id >,
'id_vlan': < id_vlan >,
'block1': <block1>,
'block2': <block2>,
'block3': <block3>,
'block4': <block4>,
'block5': <block5>,
'block6': <block6>,
'block7': <block7>,
'block8': <block8>,
'descricao': < description >
'equipamento': [ { all name equipamentos related } ], }}
```

Raises

- **IpNaoExisteError** – Network does not have any ips.
- **InvalidParameterError** – Network identifier is none or invalid.
- **DataBaseError** – Networkapi failed to access the database.

find_ips_by_equip(*id_equip*)
Get Ips related to equipment by its identifier

Parameters **id_equip** – Equipment identifier. Integer value and greater than zero.

Returns

Dictionary with the following structure:

{ ips: { ipv4:[id: <id_ip4>, oct1: <oct1>, oct2: <oct2>, oct3: <oct3>, oct4: <oct4>, descricao: <descricao>] ipv6:[id: <id_ip6>, block1: <block1>, block2: <block2>, block3: <block3>, block4: <block4>, block5: <block5>, block6: <block6>, block7: <block7>, block8: <block8>, descricao: <descricao>] } }

Raises

- **UserNotAuthorizedError** – User dont have permission to list ips.
- **InvalidParameterError** – Equipment identifier is none or invalid.
- **XMLError** – Networkapi failed to generate the XML response.
- **DataBaseError** – Networkapi failed to access the database.

get_available_ip4(*id_network*)
Get a available IP in the network ipv4

Parameters **id_network** – Network identifier. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{'ip': {'ip': < available_ip >}}
```

Raises

- **IpNotAvailableError** – Network dont have available IP for insert a new IP
- **NetworkIPv4NotFoundError** – Network is not found
- **UserNotAuthorizedError** – User dont have permission to get a available IP
- **InvalidParameterError** – Network identifier is null or invalid.
- **XMLError** – Networkapi failed to generate the XML response.
- **DataBaseError** – Networkapi failed to access the database.

get_available_ip4_for_vip(*id_evip*, *name*)

Get and save a available IP in the network ipv6 for vip request

Parameters

- **id_evip** – Vip environment identifier. Integer value and greater than zero.
- **name** – Ip description

Returns Dictionary with the following structure:

```
{'ip': {'oct1': < oct1 >,
'oct2': < oct2 >,
'oct3': < oct3 >,
'oct4': < oct4 >,
'networkipv4': <networkipv4>,
'id': <id>,
'descricao': <descricao>}}
```

Raises

- **IpNotAvailableError** – Network dont have available IP for vip environment.
- **EnvironmentVipNotFoundError** – Vip environment not registered.
- **UserNotAuthorizedError** – User dont have permission to perform operation.
- **InvalidParameterError** – Vip environment identifier is none or invalid.
- **XMLError** – Networkapi failed to generate the XML response.
- **DataBaseError** – Networkapi failed to access the database.

get_available_ip6(*id_network6*)

Get a available IP in Network ipv6

Parameters **id_network6** – Network ipv6 identifier. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{'ip6': {'ip6': < available_ip6 >}}
```

Raises

- **IpNotAvailableError** – Network dont have available IP.
- **NetworkIPv4NotFoundError** – Network was not found.

- **UserNotAuthorizedError** – User dont have permission to get a available IP.
- **InvalidParameterError** – Network ipv6 identifier is none or invalid.
- **XMLError** – Networkapi failed to generate the XML response.
- **DataBaseError** – Networkapi failed to access the database.

get_available_ip6_for_vip(*id_evip*, *name*)
Get and save a available IP in the network ipv6 for vip request

Parameters

- **id_evip** – Vip environment identifier. Integer value and greater than zero.
- **name** – Ip description

Returns Dictionary with the following structure:

```
{'ip': {'bloco1':<bloco1>,
'bloco2':<bloco2>,
'bloco3':<bloco3>,
'bloco4':<bloco4>,
'bloco5':<bloco5>,
'bloco6':<bloco6>,
'bloco7':<bloco7>,
'bloco8':<bloco8>,
'id':<id>,
'networkipv6':<networkipv6>,
'description':<description>}}
```

Raises

- **IpNotAvailableError** – Network dont have available IP for vip environment.
- **EnvironmentVipNotFoundError** – Vip environment not registered.
- **UserNotAuthorizedError** – User dont have permission to perform operation.
- **InvalidParameterError** – Vip environment identifier is none or invalid.
- **XMLError** – Networkapi failed to generate the XML response.
- **DataBaseError** – Networkapi failed to access the database.

get_ip_by_equip_and_vip(*equip_name*, *id_evip*)
Get a available IP in the Equipment related Environment VIP

Parameters

- **equip_name** – Equipment Name.
- **id_evip** – Vip environment identifier. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{ 'ipv4': [ {'id': < id >, 'ip': < ip >, 'network': { 'id': < id >, 'network': < network >,
'ipv6': [ {'id': < id >, 'ip': < ip >, 'network': { 'id': < id >, 'network': < network >, 'm
```

Raises

- **InvalidParameterError** – Vip environment identifier or equipment name is none or invalid.
- **EquipamentoNotFoundError** – Equipment not registered.

- **EnvironmentVipNotFoundError** – Vip environment not registered.
- **UserNotAuthorizedError** – User dont have permission to perform operation.
- **XMLError** – Networkapi failed to generate the XML response.
- **DataBaseError** – Networkapi failed to access the database.

get_ipv4(*id_ip*)
Get IPv4 by id.

Parameters **id_ip** – ID of IPv4.

Returns Dictionary with the following structure:

```
{'ip': {'id': < id >,
'networkipv4': < networkipv4 >,
'oct4': < oct4 >,
'oct3': < oct3 >,
'oct2': < oct2 >,
'oct1': < oct1 >,
'descricao': < descricao >,
'equipamentos': [ { all name of equipments related } ] , }}
```

Raises

- **IpNaoExisteError** – IP is not registered.
- **InvalidParameterError** – IP identifier is null or invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

get_ipv4_or_ipv6(*ip*)
Get a Ipv4 or Ipv6 by IP

Parameters **ip** – IPv4 or Ipv6. 'xxx.xxx.xxx.xxx or xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx'

Returns Dictionary with the following structure:

```
{'ips': [{'oct4': < oct4 >, 'oct2': < oct2 >, 'oct3': < oct3 >,
'oct1': < oct1 >, 'version': < version >,
'networkipv4': < networkipv4 >, 'id': < id >, 'descricao': < descricao >}, ... ] }.
or
{'ips': [ {'block1': < block1 >, 'block2': < block2 >, 'block3': < block3 >, 'block4': < blo
'version': < version >, 'networkipv6': < networkipv6 >, 'id': < id >, 'descricao': < descric
```

Raises

- **IpNaoExisteError** – Ipv4 or Ipv6 not found.
- **UserNotAuthorizedError** – User dont have permission to perform operation.
- **InvalidParameterError** – Ip string is none or invalid.
- **XMLError** – Networkapi failed to generate the XML response.
- **DataBaseError** – Networkapi failed to access the database.

get_ipv6(*id_ip*)
Get IPv6 by id.

Parameters **id_ip** – ID of IPv6.

Returns Dictionary with the following structure:

```
{'ip': {'id': < id >,
'networkipv6': < networkipv6 >,
'block1': < block1 >,
'block2': < block2 >,
'block3': < block3 >,
'block4': < block4 >,
'block5': < block5 >,
'block6': < block6 >,
'block7': < block7 >,
'block8': < block8 >,
'description': < description >,
'equipamentos': [ { all name of equipments related } ] , }}
```

Raises

- **IpNaoExisteError** – IP is not registered.
- **InvalidParameterError** – IP identifier is null or invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

save_ipv4 (*ip4*, *id_equip*, *descricao*, *id_net*)
Save a IP4 and associate with equipment

Parameters

- **ip4** – An IP4 available to save in format x.x.x.x.
- **id_equip** – Equipment identifier. Integer value and greater than zero.
- **descricao** – IP description.
- **id_net** – Network identifier. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{ ip: { id: <id_ip4>,
oct1: <oct1>,
oct2: <oct2>,
oct3: <oct3>,
oct4: <oct4>,
equipamento: [ { all equipamentos related } ] ,
descricao: <descricao> } }
```

save_ipv6 (*ip6*, *id_equip*, *descricao*, *id_net*)
Save an IP6 and associate with equipment

Parameters

- **ip6** – An IP6 available to save in format xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx.
- **id_equip** – Equipment identifier. Integer value and greater than zero.
- **descricao** – IPv6 description.
- **id_net** – Network identifier. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{'ipv6': {'id': < id >,
'block1': <block1>,
'block2': <block2>,
'block3': <block3>,
'block4': <block4>,
'block5': <block5>,
'block6': <block6>,
'block7': <block7>,
'block8': <block8>,
'descricao': < description >,
'equipamento': [ { all name equipamentos related } ], }}
```

search_ipv6_environment(*ipv6*, *id_environment*)

Get IPv6 with an associated environment.

Parameters

- **ipv6** – IPv6 address in the format x1:x2:x3:x4:x5:x6:x7:x8.
- **id_environment** – Environment identifier. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{'ipv6': {'id': < id >,
'id_vlan': < id_vlan >,
'bloco1': < bloco1 >,
'bloco2': < bloco2 >,
'bloco3': < bloco3 >,
'bloco4': < bloco4 >,
'bloco5': < bloco5 >,
'bloco6': < bloco6 >,
'bloco7': < bloco7 >,
'bloco8': < bloco8 >,
'descricao': < descricao > }}
```

Raises

- **IpNaoExisteError** – IPv6 is not registered or is not associated to the environment.
- **AmbienteNaoExisteError** – Environment not found.
- **InvalidParameterError** – Environment identifier and/or IPv6 string is/are none or invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.Marca module

class networkapiclient.Marca.**Marca**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: networkapiclient.GenericClient.GenericClient

alterar(*id_brand*, *name*)

Change Brand from by the identifier.

Parameters

- **id_brand** – Identifier of the Brand. Integer value and greater than zero.

- **name** – Brand name. String with a minimum 3 and maximum of 100 characters

Returns None

Raises

- **InvalidParameterError** – The identifier of Brand or name is null and invalid.
- **NomeMarcaDuplicadoError** – There is already a registered Brand with the value of name.
- **MarcaNaoExisteError** – Brand not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

inserir(*name*)

Inserts a new Brand and returns its identifier

Parameters **name** – Brand name. String with a minimum 3 and maximum of 100 characters

Returns Dictionary with the following structure:

```
{'marca': {'id': < id_brand >}}
```

Raises

- **InvalidParameterError** – Name is null and invalid.
- **NomeMarcaDuplicadoError** – There is already a registered Brand with the value of name.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

listar()

List all Brand.

Returns Dictionary with the following structure:

```
{'brand': [{'id': < id >,
'nome': < nome >}, ... too Brands ...]}
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

remover(*id_brand*)

Remove Brand from by the identifier.

Parameters **id_brand** – Identifier of the Brand. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – The identifier of Brand is null and invalid.
- **MarcaNaoExisteError** – Brand not registered.
- **MarcaError** – The brand is associated with a model.

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.Modelo module

class networkapiclient.Modelo.**Modelo**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: networkapiclient.GenericClient.GenericClient

alterar(*id_model*, *id_brand*, *name*)

Change Model from by the identifier.

Parameters

- **id_model** – Identifier of the Model. Integer value and greater than zero.
- **id_brand** – Identifier of the Brand. Integer value and greater than zero.
- **name** – Model name. String with a minimum 3 and maximum of 100 characters

Returns None

Raises

- **InvalidParameterError** – The identifier of Model, Brand or name is null and invalid.
- **MarcaNaoExisteError** – Brand not registered.
- **ModeloEquipamentoNaoExisteError** – Model not registered.
- **NomeMarcaModeloDuplicadoError** – There is already a registered Model with the value of name and brand.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response

get_by_script_id(*script_id*)

inserir(*id_brand*, *name*)

Inserts a new Model and returns its identifier

Parameters

- **id_brand** – Identifier of the Brand. Integer value and greater than zero.
- **name** – Model name. String with a minimum 3 and maximum of 100 characters

Returns Dictionary with the following structure:

```
{'model': {'id': < id_model >}}
```

Raises

- **InvalidParameterError** – The identifier of Brand or name is null and invalid.
- **NomeMarcaModeloDuplicadoError** – There is already a registered Model with the value of name and brand.
- **MarcaNaoExisteError** – Brand not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response

listar()
List all Model.

Returns Dictionary with the following structure:

```
{'model': [{'id': < id >,
'nome':< nome >,
'id_marca':< id_marca >,
'nome_marca':< nome_marca >}, ... too Model ... ]}
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response

listar_por_marca(*id_brand*)
List all Model by Brand.

Parameters **id_brand** – Identifier of the Brand. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{'model': [{'id': < id >,
'nome': < nome >,
'id_marca': < id_marca >}, ... too Model ...]}
```

Raises

- **InvalidParameterError** – The identifier of Brand is null and invalid.
- **MarcaNaoExisteError** – Brand not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response

remover(*id_model*)
Remove Model from by the identifier.

Parameters **id_model** – Identifier of the Model. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – The identifier of Model is null and invalid.
- **ModeloEquipamentoNaoExisteError** – Model not registered.
- **ModeloEquipamentoError** – The Model is associated with a equipment.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response

networkapiclient.Network module

class networkapiclient.Network.**DHCPRelayIPv4**(*networkapi_url*, *user*, *password*, *user_ldap=None*)
Bases: networkapiclient.ApiGenericClient.ApiGenericClient

add(*networkipv4_id*, *ipv4_id*)
List all DHCPRelayIPv4.

Param networkipv4_id, ipv4_id

Returns

Following dictionary: { "networkipv4": <networkipv4_id>, "id": <id>, "ipv4": {

"oct4": <oct4>, "oct2": <oct2>, "oct3": <oct3>, "oct1": <oct1>, "ip_formated": "<string IPv4>", "networkipv4": <networkipv4_id>, "id": <ipv4_id>, "descricao": "<string description>"

}

Raises NetworkAPIException Falha ao acessar fonte de dados

get_by_pk(*dhcprelayv4_id*)
List DHCPRelayIPv4 by ID

Param dhcprelayv4_id

Returns

Following dictionary: { "networkipv4": <networkipv4_id>, "id": <id>, "ipv4": {

"oct4": <oct4>, "oct2": <oct2>, "oct3": <oct3>, "oct1": <oct1>, "ip_formated": "<string IPv4>", "networkipv4": <networkipv4_id>, "id": <ipv4_id>, "descricao": "<string description>"

}

Raises NetworkAPIException Falha ao acessar fonte de dados

list(*networkipv4=None*, *ipv4=None*)
List all DHCPRelayIPv4.

Param networkipv4: networkipv4 id - list all dhcprelay filtering by networkipv4 id ipv4: ipv4 id - list all dhcprelay filtering by ipv4 id

Returns

Following dictionary: [

{ "networkipv4": <networkipv4_id>, "id": <id>, "ipv4": {

"oct4": <oct4>, "oct2": <oct2>, "oct3": <oct3>, "oct1": <oct1>, "ip_formated": "<string IPv4>", "networkipv4": <networkipv4_id>, "id": <ipv4_id>, "descricao": "<string description>"

}, {...}

]

Raises NetworkAPIException Falha ao acessar fonte de dados

remove(*dhcprelayv4_id*)
Remove DHCPRelayIPv4 by ID

Param dhcprelayv4_id

Returns Nothing

Raises NetworkAPIException Falha ao acessar fonte de dados

class networkapiclient.Network.**DHCPRelayIPv6**(*networkapi_url*, *user*, *password*, *user_ldap=None*)
Bases: networkapiclient.ApiGenericClient.ApiGenericClient

add(*networkipv6_id*, *ipv6_id*)
List all DHCPRelayIPv4.

Param Object DHCPRelayIPv4

Returns

Following dictionary: { "networkipv6": <networkipv4_id>, "id": <id>, "ipv6": {

"block1": <block1>, "block2": <block2>, "block3": <block3>, "block4": <block4>, "block5": <block5>, "block6": <block6>, "block7": <block7>, "block8": <block8>, "ip_formated": "<string IPv6>", "networkipv6": <networkipv6_id>, "id": <ipv6_id>, "description": "<string description>"

}

Raises NetworkAPIException Falha ao acessar fonte de dados

get_by_pk(*dhcprelayv6_id*)
List DHCPRelayIPv6 by ID

Param dhcprelayv4_id

Returns

Following dictionary: { "networkipv6": <networkipv4_id>, "id": <id>, "ipv6": {

"block1": <block1>, "block2": <block2>, "block3": <block3>, "block4": <block4>, "block5": <block5>, "block6": <block6>, "block7": <block7>, "block8": <block8>, "ip_formated": "<string IPv6>", "networkipv6": <networkipv6_id>, "id": <ipv6_id>, "description": "<string description>"

}

Raises NetworkAPIException Falha ao acessar fonte de dados

list(*networkipv6=None*, *ipv6=None*)
List all DHCPRelayIPv6.

Param networkipv6: networkipv6 id - list all dhcprelay filtering by networkipv6 id ipv6: ipv6 id - list all dhcprelay filtering by ipv6 id

Returns

Following dictionary: [

{ "networkipv6": <networkipv4_id>, "id": <id>, "ipv6": {

"block1": <block1>, "block2": <block2>, "block3": <block3>, "block4": <block4>, "block5": <block5>, "block6": <block6>, "block7": <block7>, "block8": <block8>, "ip_formated": "<string IPv6>", "networkipv6": <networkipv6_id>, "id": <ipv6_id>, "description": "<string description>"

}, {...}

]

Raises NetworkAPIException Falha ao acessar fonte de dados

remove(*dhcprelayv6_id*)
Remove DHCPRelayIPv6 by ID

Param dhcprelayv6_id

Returns Nothing

Raises NetworkAPIException Falha ao acessar fonte de dados

class networkapiclient.Network.**Network**(*networkapi_url*, *user*, *password*, *user_ldap=None*)
Bases: networkapiclient.GenericClient.GenericClient

add_network(*network*, *id_vlan*, *id_network_type*, *id_environment_vip=None*, *cluster_unit=None*)
Add new network

Parameters

- **network** – IPV4 or IPV6 (ex.: "10.0.0.3/24")
- **id_vlan** – Identifier of the Vlan. Integer value and greater than zero.
- **id_network_type** – Identifier of the NetworkType. Integer value and greater than zero.
- **id_environment_vip** – Identifier of the Environment Vip. Integer value and greater than zero.

Returns Following dictionary:

```
{'network':{'id': <id_network>,
'rede': <network>,
'broadcast': <broadcast> (if is IPv4),
'mask': <net_mask>,
'id_vlan': <id_vlan>,
'id_tipo_rede': <id_network_type>,
'id_ambiente_vip': <id_ambiente_vip>,
'active': <active>} }
```

Raises

- **TipoRedeNaoExisteError** – NetworkType not found.
- **InvalidParameterError** – Invalid ID for Vlan or NetworkType.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **IPNaoDisponivelError** – Network address unavailable to create a NetworkIPv4.
- **NetworkIPRangeEnvError** – Other environment already have this ip range.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

add_network_ipv4(*id_vlan*, *id_tipo_rede*, *id_ambiente_vip=None*, *prefix=None*)
Add new networkipv4

Parameters

- **id_vlan** – Identifier of the Vlan. Integer value and greater than zero.
- **id_tipo_rede** – Identifier of the NetworkType. Integer value and greater than zero.
- **id_ambiente_vip** – Identifier of the Environment Vip. Integer value and greater than zero.
- **prefix** – Prefix.

Returns Following dictionary:

```
{'vlan': {'id': < id_vlan >,
'nome': < nome_vlan >,
'num_vlan': < num_vlan >,
'id_tipo_rede': < id_tipo_rede >,
'id_ambiente': < id_ambiente >,
'rede_oct1': < rede_oct1 >,
```

```
'rede_oct2': < rede_oct2 >,
'rede_oct3': < rede_oct3 >,
'rede_oct4': < rede_oct4 >,
'bloco': < bloco >,
'mascara_oct1': < mascara_oct1 >,
'mascara_oct2': < mascara_oct2 >,
'mascara_oct3': < mascara_oct3 >,
'mascara_oct4': < mascara_oct4 >,
'broadcast': < broadcast >,
'descricao': < descricao >,
'acl_file_name': < acl_file_name >,
'acl_valida': < acl_valida >,
'ativada': < ativada >}}
```

Raises

- **TipoRedeNaoExisteError** – NetworkType not found.
- **InvalidParameterError** – Invalid ID for Vlan or NetworkType.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **IPNaoDisponivelError** – Network address unavailable to create a NetworkIPv4.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

add_network_ipv4_hosts(*id_vlan*, *id_tipo_rede*, *num_hosts*, *id_ambiente_vip=None*)

Add new networkipv4

Parameters

- **id_vlan** – Identifier of the Vlan. Integer value and greater than zero.
- **id_tipo_rede** – Identifier of the NetworkType. Integer value and greater than zero.
- **num_hosts** – Number of hosts expected. Integer value and greater than zero.
- **id_ambiente_vip** – Identifier of the Environment Vip. Integer value and greater than zero.

Returns Following dictionary:

```
{'vlan': {'id': < id_vlan >,
'nome': < nome_vlan >,
'num_vlan': < num_vlan >,
'id_tipo_rede': < id_tipo_rede >,
'id_ambiente': < id_ambiente >,
'rede_oct1': < rede_oct1 >,
'rede_oct2': < rede_oct2 >,
'rede_oct3': < rede_oct3 >,
'rede_oct4': < rede_oct4 >,
'bloco': < bloco >,
'mascara_oct1': < mascara_oct1 >,
'mascara_oct2': < mascara_oct2 >,
'mascara_oct3': < mascara_oct3 >,
'mascara_oct4': < mascara_oct4 >,
'broadcast': < broadcast >,
'descricao': < descricao >,
'acl_file_name': < acl_file_name >,
'acl_valida': < acl_valida >,
'ativada': < ativada >}}
```

Raises

- **TipoRedeNaoExisteError** – NetworkType not found.
- **InvalidParameterError** – Invalid ID for Vlan or NetworkType.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **IPNaoDisponivelError** – Network address unavailable to create a NetworkIPv4.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

add_network_ipv6(*id_vlan*, *id_tipo_rede*, *id_ambiente_vip=None*, *prefix=None*)
Add new networkipv6

Parameters

- **id_vlan** – Identifier of the Vlan. Integer value and greater than zero.
- **id_tipo_rede** – Identifier of the NetworkType. Integer value and greater than zero.
- **id_ambiente_vip** – Identifier of the Environment Vip. Integer value and greater than zero.
- **prefix** – Prefix.

Returns Following dictionary:

```
{'vlan': {'id': < id_vlan >,
'nome': < nome_vlan >,
'num_vlan': < num_vlan >,
'id_tipo_rede': < id_tipo_rede >,
'id_ambiente': < id_ambiente >,
'rede_oct1': < rede_oct1 >,
'rede_oct2': < rede_oct2 >,
'rede_oct3': < rede_oct3 >,
'rede_oct4': < rede_oct4 >,
'rede_oct5': < rede_oct4 >,
'rede_oct6': < rede_oct4 >,
'rede_oct7': < rede_oct4 >,
'rede_oct8': < rede_oct4 >,
'bloco': < bloco >,
'mascara_oct1': < mascara_oct1 >,
'mascara_oct2': < mascara_oct2 >,
'mascara_oct3': < mascara_oct3 >,
'mascara_oct4': < mascara_oct4 >,
'mascara_oct5': < mascara_oct4 >,
'mascara_oct6': < mascara_oct4 >,
'mascara_oct7': < mascara_oct4 >,
'mascara_oct8': < mascara_oct4 >,
'broadcast': < broadcast >,
'descricao': < descricao >,
'acl_file_name': < acl_file_name >,
'acl_valida': < acl_valida >,
'ativada': < ativada >}}
```

Raises

- **TipoRedeNaoExisteError** – NetworkType not found.
- **InvalidParameterError** – Invalid ID for Vlan or NetworkType.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.

- **IPNaoDisponivelError** – Network address unavailable to create a NetworkIPv6.
- **ConfigEnvironmentInvalidError** – Invalid Environment Configuration or not registered
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

add_network_ipv6_hosts(*id_vlan*, *id_tipo_rede*, *num_hosts*, *id_ambiente_vip=None*)
Add new networkipv6

Parameters

- **id_vlan** – Identifier of the Vlan. Integer value and greater than zero.
- **id_tipo_rede** – Identifier of the NetworkType. Integer value and greater than zero.
- **num_hosts** – Number of hosts expected. Integer value and greater than zero.
- **id_ambiente_vip** – Identifier of the Environment Vip. Integer value and greater than zero.

Returns Following dictionary:

```
{'vlan': {'id': < id_vlan >,
'nome': < nome_vlan >,
'num_vlan': < num_vlan >,
'id_tipo_rede': < id_tipo_rede >,
'id_ambiente': < id_ambiente >,
'rede_oct1': < rede_oct1 >,
'rede_oct2': < rede_oct2 >,
'rede_oct3': < rede_oct3 >,
'rede_oct4': < rede_oct4 >,
'rede_oct5': < rede_oct4 >,
'rede_oct6': < rede_oct4 >,
'rede_oct7': < rede_oct4 >,
'rede_oct8': < rede_oct4 >,
'bloco': < bloco >,
'mascara_oct1': < mascara_oct1 >,
'mascara_oct2': < mascara_oct2 >,
'mascara_oct3': < mascara_oct3 >,
'mascara_oct4': < mascara_oct4 >,
'mascara_oct5': < mascara_oct4 >,
'mascara_oct6': < mascara_oct4 >,
'mascara_oct7': < mascara_oct4 >,
'mascara_oct8': < mascara_oct4 >,
'broadcast': < broadcast >,
'descricao': < descricao >,
'acl_file_name': < acl_file_name >,
'acl_valida': < acl_valida >,
'ativada': < ativada >}}
```

Raises

- **TipoRedeNaoExisteError** – NetworkType not found.
- **InvalidParameterError** – Invalid ID for Vlan or NetworkType.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **IPNaoDisponivelError** – Network address unavailable to create a NetworkIPv6.
- **ConfigEnvironmentInvalidError** – Invalid Environment Configuration or not registered
- **DataBaseError** – Networkapi failed to access the database.

- **XMLError** – Networkapi failed to generate the XML response.

create_networks(*ids*, *id_vlan*)
Set column 'active = 1' in tables redeipv4 and redeipv6]

Parameters **ids** – ID for NetworkIPv4 and/or NetworkIPv6

Returns Nothing

deallocate_network_ipv4(*id_network_ipv4*)
Deallocate all relationships between NetworkIPv4.

Parameters **id_network_ipv4** – ID for NetworkIPv4

Returns Nothing

Raises

- **InvalidParameterError** – Invalid ID for NetworkIPv4.
- **NetworkIPv4NotFoundError** – NetworkIPv4 not found.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

deallocate_network_ipv6(*id_network_ipv6*)
Deallocate all relationships between NetworkIPv6.

Parameters **id_network_ipv6** – ID for NetworkIPv6

Returns Nothing

Raises

- **InvalidParameterError** – Invalid ID for NetworkIPv6.
- **NetworkIPv6NotFoundError** – NetworkIPv6 not found.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

edit_network(*id_network*, *ip_type*, *id_net_type*, *id_env_vip=None*, *cluster_unit=None*)
Edit a network 4 or 6

Parameters

- **id_network** – Identifier of the Network. Integer value and greater than zero.
- **id_net_type** – Identifier of the NetworkType. Integer value and greater than zero.
- **id_env_vip** – Identifier of the Environment Vip. Integer value and greater than zero.
- **ip_type** – Identifier of the Network IP Type: 0 = IP4 Network; 1 = IP6 Network;

Returns None

Raises

- **TipoRedeNaoExisteError** – NetworkType not found.
- **InvalidParameterError** – Invalid ID for Vlan or NetworkType.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **IPNaoDisponivelError** – Network address unavailable to create a NetworkIPv4.
- **DataBaseError** – Networkapi failed to access the database.

- **XMLError** – Networkapi failed to generate the XML response.

get_network_ipv4(*id_network*)
Get networkipv4

Parameters **id_network** – Identifier of the Network. Integer value and greater than zero.

Returns Following dictionary:

```
{'network': {'id': < id_networkIpv6 >,
'network_type': < id_tipo_rede >,
'ambiente_vip': < id_ambiente_vip >,
'vlan': <id_vlan>
'oct1': < rede_oct1 >,
'oct2': < rede_oct2 >,
'oct3': < rede_oct3 >,
'oct4': < rede_oct4 >
'blocK': < bloco >,
'mask_oct1': < mascara_oct1 >,
'mask_oct2': < mascara_oct2 >,
'mask_oct3': < mascara_oct3 >,
'mask_oct4': < mascara_oct4 >,
'active': < ativada >,
'broadcast':<'broadcast>, }}
```

Raises

- **NetworkIPv4NotFoundError** – NetworkIPV4 not found.
- **InvalidValueError** – Invalid ID for NetworkIpv4
- **NetworkIPv4Error** – Error in NetworkIpv4
- **XMLError** – Networkapi failed to generate the XML response.

get_network_ipv6(*id_network*)
Get networkipv6

Parameters **id_network** – Identifier of the Network. Integer value and greater than zero.

Returns Following dictionary:

```
{'network': {'id': < id_networkIpv6 >,
'network_type': < id_tipo_rede >,
'ambiente_vip': < id_ambiente_vi >,
'vlan': <id_vlan>
'block1': < rede_oct1 >,
'block2': < rede_oct2 >,
'block3': < rede_oct3 >,
'block4': < rede_oct4 >,
'block5': < rede_oct4 >,
'block6': < rede_oct4 >,
'block7': < rede_oct4 >,
'block8': < rede_oct4 >,
'blocK': < bloco >,
'mask1': < mascara_oct1 >,
'mask2': < mascara_oct2 >,
'mask3': < mascara_oct3 >,
'mask4': < mascara_oct4 >,
'mask5': < mascara_oct4 >,
'mask6': < mascara_oct4 >,
```

```
'mask7': < mascara_oct4 >,
'mask8': < mascara_oct4 >,
'active': < ativada >, }}
```

Raises

- **NetworkIPv6NotFoundError** – NetworkIPV6 not found.
- **InvalidValueError** – Invalid ID for NetworkIpv6
- **NetworkIPv6Error** – Error in NetworkIpv6
- **XMLError** – Networkapi failed to generate the XML response.

remove_networks(*ids*)

Set column 'active = 0' in tables redeipv4 and redeipv6]

Parameters **ids** – ID for NetworkIPv4 and/or NetworkIPv6

Returns Nothing

Raises

- **NetworkInactiveError** – Unable to remove the network because it is inactive.
- **InvalidParameterError** – Invalid ID for Network or NetworkType.
- **NetworkIPv4NotFoundError** – NetworkIPv4 not found.
- **NetworkIPv6NotFoundError** – NetworkIPv6 not found.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.OptionVIP module

class networkapiclient.OptionVIP.**OptionVIP**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: networkapiclient.GenericClient.GenericClient

add(*tipo_opcao*, *nome_opcao_txt*)

Inserts a new Option VIP and returns its identifier.

Parameters

- **tipo_opcao** – Type. String with a maximum of 50 characters and respect [a-zA-Z_-]
- **nome_opcao_txt** – Name Option. String with a maximum of 50 characters and respect [a-zA-Z_-]

Returns Following dictionary:

```
{'option_vip': {'id': < id >}}
```

Raises

- **InvalidParameterError** – The value of tipo_opcao or nome_opcao_txt is invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

alter(*id_option_vip*, *tipo_opcao*, *nome_opcao_txt*)
Change Option VIP from by the identifier.

Parameters

- **id_option_vip** – Identifier of the Option VIP. Integer value and greater than zero.
- **tipo_opcao** – Type. String with a maximum of 50 characters and respect [a-zA-Z_-]
- **nome_opcao_txt** – Name Option. String with a maximum of 50 characters and respect [a-zA-Z_-]

Returns None

Raises

- **InvalidParameterError** – Option VIP identifier is null and invalid.
- **InvalidParameterError** – The value of tipo_opcao or nome_opcao_txt is invalid.
- **OptionVipNotFoundError** – Option VIP not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

associate(*id_option_vip*, *id_environment_vip*)
Create a relationship of OptionVip with EnvironmentVip.

Parameters

- **id_option_vip** – Identifier of the Option VIP. Integer value and greater than zero.
- **id_environment_vip** – Identifier of the Environment VIP. Integer value and greater than zero.

Returns Following dictionary

```
{'opcoesvip_ambiente_xref': {'id': < id_opcoesvip_ambiente_xref >} }
```

Raises

- **InvalidParameterError** – Option VIP/Environment VIP identifier is null and/or invalid.
- **OptionVipNotFoundError** – Option VIP not registered.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **OptionVipError** – Option vip is already associated with the environment vip.
- **UserNotAuthorizedError** – User does not have authorization to make this association.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

buscar_balanceamento_opcvip(*id_ambiente_vip*)
Search nome_opcao_txt of Option VIP when tipo_opcao = 'Balanceamento' by environmentvip_id

Returns Dictionary with the following structure:

```
{'balanceamento_opt': 'balanceamento_opt': <'nome_opcao_txt'>}
```

Raises

- **InvalidParameterError** – Environment VIP identifier is null and invalid.

- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **InvalidParameterError** – finalidade_txt and cliente_txt is null and invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

buscar_grupo_cache_opcvip(*id_ambiente_vip*)
Search nome_opcao_txt of Option VIP when tipo_opcao = 'Grupo de Caches' by environmentvip_id

Returns Dictionary with the following structure:

```
{'grupocache_opt': 'grupocache_opt': <'nome_opcao_txt'>}
```

Raises

- **EnvironmentVipNotFoundError** – Ambiente VIP não encontrado
- **OptionVipError** – Falha na requisição.
- **EnvironmentVipError** – Falha na requisição.
- **InvalidParameterError** – O identificador da requisição de VIP é inválido ou nulo.
- **ScriptError** – Falha ao executar o script.
- **DataBaseError** – Falha na networkapi ao acessar o banco de dados.
- **XMLError** – Falha na networkapi ao gerar o XML de resposta.

buscar_healthchecks(*id_ambiente_vip*)
Search healthcheck by environmentvip_id

Returns Dictionary with the following structure:

```
{'healthcheck_opt': [{'name': <name>, 'id': <id>},...]}
```

Raises

- **InvalidParameterError** – Environment VIP identifier is null and invalid.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **InvalidParameterError** – id_ambiente_vip is null and invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

buscar_idtrafficreturn_opcvip(*nome_opcao_txt*)
Search id of Option VIP when tipo_opcao = 'Retorno de trafego'

Returns Dictionary with the following structure:

```
{'trafficreturn_opt': 'trafficreturn_opt': <'id'>}
```

Raises

- **InvalidParameterError** – Environment VIP identifier is null and invalid.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **InvalidParameterError** – finalidade_txt and cliente_txt is null and invalid.

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

buscar_persistencia_opcvip(*id_ambiente_vip*)
Search nome_opcao_txt of Option VIP when tipo_opcao = 'Persistencia' by environmentvip_id

Returns Dictionary with the following structure:

```
{'persistencia_opt': 'persistencia_opt': <'nome_opcao_txt'>}
```

Raises

- **InvalidParameterError** – Environment VIP identifier is null and invalid.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **InvalidParameterError** – finalidade_txt and cliente_txt is null and invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

buscar_rules(*id_ambiente_vip*, *id_vip=''*)
Search rules by environmentvip_id

Returns Dictionary with the following structure:

```
{'name_rule_opt': [{'name_rule_opt': <name>, 'id': <id>}, ...]}
```

Raises

- **InvalidParameterError** – Environment VIP identifier is null and invalid.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **InvalidParameterError** – finalidade_txt and cliente_txt is null and invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

buscar_timeout_opcvip(*id_ambiente_vip*)
Buscar nome_opcao_txt das Opcoes VIp quando tipo_opcao = 'Timeout' pelo environmentvip_id

Returns Dictionary with the following structure:

:: {'timeout_opt': 'timeout_opt': <'nome_opcao_txt'>}

Raises

- **InvalidParameterError** – Environment VIP identifier is null and invalid.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **InvalidParameterError** – finalidade_txt and cliente_txt is null and invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

buscar_trafficreturn_opcvip(*id_ambiente_vip*)
Search nome_opcao_txt of Option VIP when tipo_opcao = 'Persistencia' by environmentvip_id

Returns Dictionary with the following structure:

```
{'persistencia_opt': 'persistencia_opt': <'nome_opcao_txt'>}
```

Raises

- **InvalidParameterError** – Environment VIP identifier is null and invalid.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **InvalidParameterError** – finalidade_txt and cliente_txt is null and invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

disassociate(*id_option_vip*, *id_environment_vip*)

Remove a relationship of OptionVip with EnvironmentVip.

Parameters

- **id_option_vip** – Identifier of the Option VIP. Integer value and greater than zero.
- **id_environment_vip** – Identifier of the Environment VIP. Integer value and greater than zero.

Returns Nothing

Raises

- **InvalidParameterError** – Option VIP/Environment VIP identifier is null and/or invalid.
- **OptionVipNotFoundError** – Option VIP not registered.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **OptionVipError** – Option vip is not associated with the environment vip
- **UserNotAuthorizedError** – User does not have authorization to make this association.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

get_all()

Get all Option VIP.

Returns Dictionary with the following structure:

```
{'option_vip': [{'id': < id >,
'tipo_opcao': < tipo_opcao >,
'nome_opcao_txt': < nome_opcao_txt >}, ... other option vips ...] }
```

Raises

- **OptionVipNotFoundError** – Option VIP not registered.
- **DataBaseError** – Can't connect to networkapi database.
- **XMLError** – Failed to generate the XML response.

get_option_vip(*id_environment_vip*)

Get all Option VIP by Environment Vip.

Returns Dictionary with the following structure:

```
{'option_vip': [{'id': < id >,
'tipo_opcao': < tipo_opcao >,
'nome_opcao_txt': < nome_opcao_txt >}, ... too option vips ...] }
```

Raises

- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **DataBaseError** – Can't connect to networkapi database.
- **XMLError** – Failed to generate the XML response.

remove(*id_option_vip*)

Remove Option VIP from by the identifier.

Parameters **id_option_vip** – Identifier of the Option VIP. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – Option VIP identifier is null and invalid.
- **OptionVipNotFoundError** – Option VIP not registered.
- **OptionVipError** – Option VIP associated with environment vip.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

search(*id_option_vip*)

Search Option VIP from by the identifier.

Parameters **id_option_vip** – Identifier of the Option VIP. Integer value and greater than zero.

Returns Following dictionary:

```
{'option_vip': {'id': < id_option_vip >,
'tipo_opcao': < tipo_opcao >,
'nome_opcao_txt': < nome_opcao_txt >} }
```

Raises

- **InvalidParameterError** – Option VIP identifier is null and invalid.
- **OptionVipNotFoundError** – Option VIP not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.Pagination module

class networkapiclient.Pagination.**Pagination**(*start_record*, *end_record*, *asorting_cols*, *searchable_columns*, *custom_search*)

asorting_cols

custom_search

end_record

searchable_columns

start_record

networkapiclient.PermissaoAdministrativa module

class networkapiclient.PermissaoAdministrativa.**PermissaoAdministrativa**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: networkapiclient.GenericClient.GenericClient

alterar(*id_perm*, *id_permission*, *read*, *write*, *id_group*)

Change Administrative Permission from by the identifier.

Parameters

- **id_perm** – Identifier of the Administrative Permission. Integer value and greater than zero.
- **id_permission** – Identifier of the Permission. Integer value and greater than zero.
- **read** – Read. 0 or 1
- **write** – Write. 0 or 1
- **id_group** – Identifier of the Group of User. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – The identifier of Administrative Permission, identifier of Permission, identifier of Group of User, read or write is null and invalid.
- **ValorIndicacaoPermissaoInvalidoError** – The value of read or write is null and invalid.
- **PermissaoAdministrativaNaoExisteError** – Administrative Permission not registered.
- **GrupoUsuarioNaoExisteError** – Group of User not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

inserir(*id_permission*, *read*, *write*, *id_group*)

Inserts a new Administrative Permission and returns its identifier.

Parameters

- **id_permission** – Identifier of the Permission. Integer value and greater than zero.
- **read** – Read. 0 or 1
- **write** – Write. 0 or 1
- **id_group** – Identifier of the Group of User. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{'perm': {'id': < id_perm >}}
```

Raises

- **InvalidParameterError** – The identifier of Administrative Permission, identifier of Group of User, read or write is null and invalid.
- **ValorIndicacaoPermissaoInvalidoError** – The value of read or write is null and invalid.
- **PermissaoAdministrativaDuplicadaError** – Function already registered for the user group.
- **GrupoUsuarioNaoExisteError** – Group of User not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

list_by_group(*id_ugroup*)

Search Administrative Permission by Group User by identifier.

Parameters **id_ugroup** – Identifier of the Group User. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{'perms': [{'ugrupo': < ugrupo_id >, 'permission':  { 'function' < function >, 'id': < id >
'id': < id >, 'escrita': < escrita >,
'leitura': < leitura >}, ... ] }
```

Raises

- **InvalidParameterError** – Group User is null and invalid.
- **UGrupoNotFoundError** – Group User not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

listar()

List all Administrative Permission.

Returns Dictionary with the following structure:

```
{'perms': [{'ugrupo': < ugrupo_id >,
'permission': < permission_id >,
'id': < id >,
'escrita': < escrita >,
'leitura': < leitura >}, ... demais permissões ...]}
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

remover(*id_perms*)

Remove Administrative Permission from by the identifier.

Parameters **id_perms** – Identifier of the Administrative Permission. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – The identifier of Administrative Permission is null and invalid.

- **PermissaoAdministrativaNaoExisteError** – Administrative Permission not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

search(*id_perm*)
Search Administrative Permission from by the identifier.

Parameters **id_perm** – Identifier of the Administrative Permission. Integer value and greater than zero.

Returns Following dictionary:

```
{'perm': {'ugrupo': < ugrupo_id >,
'permission': < permission_id >, 'id': < id >,
'escrita': < escrita >, 'leitura': < leitura >}}
```

Raises

- **InvalidParameterError** – Group User identifier is null and invalid.
- **PermissaoAdministrativaNaoExisteError** – Administrative Permission not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.Permission module

class networkapiclient.Permission.**Permission**(*networkapi_url*, *user*, *password*, *user_ldap=None*)
Bases: networkapiclient.GenericClient.GenericClient

list_all()
List all Permission.

Returns Dictionary with the following structure:

```
{'perms': [{ 'function' < function >, 'id': < id > }, ... more permissions ...]}
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.Pool module

class networkapiclient.Pool.**Pool**(*networkapi_url*, *user*, *password*, *user_ldap=None*)
Bases: networkapiclient.ApiGenericClient.ApiGenericClient

create(*pools*)
Create Pools Running Script And Update to Created

Parameters **pools** – List of pools

Returns None on success

:raise PoolDoesNotExistException :raise ScriptCreatePoolException :raise InvalidIdPoolException :raise NetworkAPIException

delete_pool(*pool_ids*)

deploy_create_pool(*pool_ids*)

deploy_remove_pool(*pool_ids*)

deploy_update_pool(*pool*, *pool_ids*)

deploy_update_pool_members(*pool_id*, *pool*)

disable(*ids*)
Disable Pool Members Running Script

Parameters **ids** – List of ids

Returns None on success

:raise PoolMemberDoesNotExistException :raise InvalidIdPoolMemberException :raise ScriptDisablePoolException :raise NetworkAPIException

enable(*ids*)
Enable Pool Members Running Script

Parameters **ids** – List of ids

Returns None on success

:raise PoolMemberDoesNotExistException :raise InvalidIdPoolMemberException :raise ScriptEnablePoolException :raise NetworkAPIException

get_available_ips_to_add_server_pool(*equip_name*, *id_ambiente*)

get_by_pk(*pk*)

get_equip_by_ip(*id_ip*)
Get equipment by IP id

Parameters **id_ip** – IP id

Returns

Following dictionary:{ “equipamento” :[{

“id”: < id > “tipo_equipamento”: < tipo_equipamento >, “modelo”: < modelo >, “nome”: < nome >, “grupos”: < grupos >

}]

Raises NetworkAPIException Falha ao acessar fonte de dados

get_opcoes_pool_by_ambiente(*id_ambiente*)

get_opcoes_pool_by_environment(*env_id*)

get_pool(*pool_id*)

get_pool_members(*pool_id*, *checkstatus='0'*)

get_poolmember_state(*id_pools*, *checkstatus=0*)
Enable/Disable pool member by list

get_requisicoes_vip_by_pool(*id_server_pool*, *pagination*)

get_vip_by_pool(*pool_id*)

inserir(*identifier*, *default_port*, *environment*, *balancing*, *healthcheck_type*, *healthcheck_expect*, *healthcheck_request*, *old_healthcheck_id*, *maxcon*, *ip_list_full*, *nome_equips*, *id_equips*, *priorities*, *weight*, *ports_reals*, *servicedownaction=None*)

list_all(*environment_id*, *pagination*)
List All Pools To Populate Datatable

Parameters **pagination** – Object Pagination

Returns

Following dictionary:{ "total" : < total >, "pools" :[{

"id": < id > "default_port": < default_port >, "identifier": < identifier >, "healthcheck": < healthcheck >,

}, ... too ...]}

Raises NetworkAPIException Falha ao acessar fonte de dados

list_all_by_reqvip(*id_vip*, *pagination*)
List All Pools To Populate Datatable

Parameters **pagination** – Object Pagination

Returns

Following dictionary:{ "total" : < total >, "pools" :[{

"id": < id > "default_port": < default_port >, "identifier": < identifier >, "healthcheck": < healthcheck >,

}, ... too ...]}

Raises NetworkAPIException Falha ao acessar fonte de dados

list_all_environment_related_environment_vip()

list_all_members_by_pool(*id_server_pool*, *checkstatus=False*, *pagination=None*)

list_all_members_by_poolll_members(*id_pools*)

list_by_environment(*environment_id*)
Disable Pool Members Running Script

Parameters **ids** – List of ids

Returns

Following dictionary:{ "pools" :[{

"id": < id > "default_port": < default_port >, "identifier": < identifier >, "healthcheck": < healthcheck >,

}, ... too ...]}

:raise ObjectDoesNotExistException :raise ValidationException :raise NetworkAPIException

list_by_environmet_vip(*environment_vip_id*)

list_environments_with_pools()

list_healthchecks()
List all healthchecks

Returns Following dictionary:

```
{'ambiente': [{ 'id': <id_environment>,
'grupo_l3': <id_group_l3>,
'grupo_l3_name': <name_group_l3>,
'ambiente_logico': <id_logical_environment>,
'ambiente_logico_name': <name_ambiente_logico>,
'divisao_dc': <id_dc_division>,
'divisao_dc_name': <name_divisao_dc>,
'filter': <id_filter>,
'filter_name': <filter_name>,
'link': <link> }, ... ]}
```

Raises DataBaseError Falha na networkapi ao acessar o banco de dados.

list_pool(*search*)

list_pool_members(*pool_id*)
Disable Pool Members Running Script

Parameters ids – List of ids

Returns

:raise ObjectDoesNotExistException :raise ValidationException :raise NetworkAPIException

remove(*pools*)
Remove Pools Running Script And Update to Not Created

Parameters ids – List of ids

Returns None on success

:raise ScriptRemovePoolException :raise InvalidIdPoolException :raise NetworkAPIException

save(*id*, *identifier*, *default_port*, *environment*, *balancing*, *healthcheck_type*, *healthcheck_expect*, *healthcheck_request*, *maxcon*, *ip_list_full*, *nome_equips*, *id_equips*, *priorities*, *weight*, *ports_reals*, *id_pool_member*, *servicedownaction=None*)

save_pool(*pool*)

save_reals(*id_server_pool*, *ip_list_full*, *nome_equips*, *id_equips*, *priorities*, *weight*, *ports_reals*, *id_pool_member*)

set_poolmember_state(*id_pools*, *pools*)
Enable/Disable pool member by list

update(*id_server_pool*, *default_port*, *balancing*, *healthcheck_type*, *healthcheck_expect*, *healthcheck_request*, *old_healthcheck_id*, *maxcon*, *ip_list_full*, *nome_equips*, *id_equips*, *priorities*, *weight*, *ports_reals*, *servicedownaction=None*)

update_pool(*pool*, *pool_id*)

networkapiclient.Roteiro module

class networkapiclient.Roteiro.**Roteiro**(*networkapi_url*, *user*, *password*, *user_ldap=None*)
Bases: networkapiclient.GenericClient.GenericClient

alterar(*id_script*, *id_script_type*, *script*, *description*, *model=None*)
Change Script from by the identifier.

Parameters

- **id_script** – Identifier of the Script. Integer value and greater than zero.
- **id_script_type** – Identifier of the Script Type. Integer value and greater than zero.
- **script** – Script name. String with a minimum 3 and maximum of 40 characters
- **description** – Script description. String with a minimum 3 and maximum of 100 characters

Returns None

Raises

- **InvalidParameterError** – The identifier of Script, script Type, script or description is null and invalid.
- **RoteiroNaoExisteError** – Script not registered.
- **TipoRoteiroNaoExisteError** – Script Type not registered.
- **NomeRoteiroDuplicadoError** – Script already registered with informed.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

get_by_id(*id_script*)

inserir(*id_script_type*, *script*, *model*, *description*)

Inserts a new Script and returns its identifier.

Parameters

- **id_script_type** – Identifier of the Script Type. Integer value and greater than zero.
- **script** – Script name. String with a minimum 3 and maximum of 40 characters
- **description** – Script description. String with a minimum 3 and maximum of 100 characters

Returns Dictionary with the following structure:

```
{'script': {'id': < id_script >}}
```

Raises

- **InvalidParameterError** – The identifier of Script Type, script or description is null and invalid.
- **TipoRoteiroNaoExisteError** – Script Type not registered.
- **NomeRoteiroDuplicadoError** – Script already registered with informed.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

listar()

List all Script.

Returns Dictionary with the following structure:

```
{'script': [{'id': < id >,
'tipo_roteiro': < tipo_roteiro >,
'nome': < nome >,
'descricao': < descricao >}, ...more Script...]}
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

listar_por_equipamento(*id_equipment*)
List all Script related Equipment.

Parameters **id_equipment** – Identifier of the Equipment. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{'script': [{'id': < id >,
'nome': < nome >,
'descricao': < descricao >,
'id_tipo_roteiro': < id_tipo_roteiro >,
'nome_tipo_roteiro': < nome_tipo_roteiro >,
'descricao_tipo_roteiro': < descricao_tipo_roteiro >}, ...more Script...]}
```

Raises

- **InvalidParameterError** – The identifier of Equipment is null and invalid.
- **EquipamentoNaoExisteError** – Equipment not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

listar_por_tipo(*id_script_type*)
List all Script by Script Type.

Parameters **id_script_type** – Identifier of the Script Type. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{'script': [{'id': < id >,
'tipo_roteiro': < id_tipo_roteiro >,
'nome': < nome >,
'descricao': < descricao >}, ...more Script...]}
```

Raises

- **InvalidParameterError** – The identifier of Script Type is null and invalid.
- **TipoRoteiroNaoExisteError** – Script Type not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

remover(*id_script*)
Remove Script from by the identifier.

Parameters **id_script** – Identifier of the Script. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – The identifier of Script is null and invalid.
- **RoteiroNaoExisteError** – Script not registered.

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.RoteiroEquipamento module

class networkapiclient.RoteiroEquipamento.**RoteiroEquipamento**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: networkapiclient.GenericClient.GenericClient

criar(*id_equipamento*, *id_roteiro*)

listar(*id_equipamento*)

listar_todos()

remover(*id_equipamento*, *id_roteiro*)

networkapiclient.TipoAcesso module

class networkapiclient.TipoAcesso.**TipoAcesso**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: networkapiclient.GenericClient.GenericClient

alterar(*id_tipo_acesso*, *protocolo*)

Edit access type by its identifier.

Parameters

- **id_tipo_acesso** – Access type identifier.
- **protocolo** – Protocol.

Returns None

Raises

- **ProtocoloTipoAcessoDuplicadoError** – Protocol already exists.
- **InvalidParameterError** – Protocol value is invalid or none.
- **TipoAcessoNaoExisteError** – Access type doesn't exist.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

inserir(*protocolo*)

Insert new access type and returns its identifier.

Parameters **protocolo** – Protocol.

Returns Dictionary with structure: {'tipo_acesso': {'id': < id >}}

Raises

- **ProtocoloTipoAcessoDuplicadoError** – Protocol already exists.
- **InvalidParameterError** – Protocol value is invalid or none.
- **DataBaseError** – Networkapi failed to access the database.

- **XMLError** – Networkapi failed to generate the XML response.

listar()
List all access types.

Returns Dictionary with structure:

```
{'tipo_acesso': [{'id': < id >,
'protocolo': < protocolo >}, ... other access types ...]}
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

remover(*id_tipo_acesso*)
Removes access type by its identifier.

Parameters **id_tipo_acesso** – Access type identifier.

Returns None

Raises

- **TipoAcessoError** – Access type associated with equipment, cannot be removed.
- **InvalidParameterError** – Protocol value is invalid or none.
- **TipoAcessoNaoExisteError** – Access type doesn't exist.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.TipoEquipamento module

class networkapiclient.TipoEquipamento.**TipoEquipamento**(*networkapi_url*, *user*, *password*, *user_ldap=None*)
Bases: networkapiclient.GenericClient.GenericClient

inserir(*name*)
Inserts a new Equipment Type and returns its identifier

Parameters **name** – Equipment Type name. String with a minimum 3 and maximum of 100 characters

Returns Dictionary with the following structure:

```
{'tipo_equipamento': {'id': < id_equipment_ype >}}
```

Raises

- **InvalidParameterError** – Name is null and invalid.
- **EquipamentoError** – There is already a registered Equipment Type with the value of name.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response

listar()
List all Equipment Type.

Returns Dictionary with the following structure:

```
{'equipment_type' : [{'id': < id >,
'nome': < name >}, ... too Equipment Type ...] }
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response

networkapiclient.TipoRede module

class networkapiclient.TipoRede.**TipoRede**(*networkapi_url*, *user*, *password*, *user_ldap=None*)
Bases: networkapiclient.GenericClient.GenericClient

alterar(*id_net_type*, *name*)
Edit network type by its identifier.

Parameters

- **id_net_type** – Network type identifier.
- **name** – Network type name.

Returns None

Raises

- **InvalidParameterError** – Network type identifier and/or name is(are) none or invalid.
- **TipoRedeNaoExisteError** – Network type does not exist.
- **NomeTipoRedeDuplicadoError** – Network type name already exists.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

inserir(*name*)
Insert new network type and return its identifier.

Parameters **name** – Network type name.

Returns Following dictionary: {'net_type': {'id': < id >}}

Raises

- **InvalidParameterError** – Network type is none or invalid.
- **NomeTipoRedeDuplicadoError** – A network type with this name already exists.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

listar()
List all network types.

Returns Following dictionary:

```
{'net_type': [{'id': < id_tipo_rede >,
'name': < nome_tipo_rede >}, ... other network types ...]}
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

remover(*id_net_type*)

Remove network type by its identifier.

Parameters **id_net_type** – Network type identifier.

Returns None

Raises

- **TipoRedeNaoExisteError** – Network type does not exist.
- **TipoRedeError** – Network type is associated with network.
- **InvalidParameterError** – Network type is none or invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.TipoRoteiro module

class networkapiclient.TipoRoteiro.**TipoRoteiro**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: networkapiclient.GenericClient.GenericClient

alterar(*id_script_type*, *type*, *description*)

Change Script Type from by the identifier.

Parameters

- **id_script_type** – Identifier of the Script Type. Integer value and greater than zero.
- **type** – Script Type type. String with a minimum 3 and maximum of 40 characters
- **description** – Script Type description. String with a minimum 3 and maximum of 100 characters

Returns None

Raises

- **InvalidParameterError** – The identifier of Script Type, type or description is null and invalid.
- **TipoRoteiroNaoExisteError** – Script Type not registered.
- **NomeTipoRoteiroDuplicadoError** – Type script already registered with informed.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

inserir(*type*, *description*)

Inserts a new Script Type and returns its identifier.

Parameters

- **type** – Script Type type. String with a minimum 3 and maximum of 40 characters
- **description** – Script Type description. String with a minimum 3 and maximum of 100 characters

Returns Dictionary with the following structure:

```
{'script_type': {'id': < id_script_type >}}
```

Raises

- **InvalidParameterError** – Type or description is null and invalid.
- **NomeTipoRoteiroDuplicadoError** – Type script already registered with informed.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

listar()

List all Script Type.

Returns Dictionary with the following structure:

```
{'script_type': [{'id': < id >,
'tipo': < tipo >,
'descricao': < descricao >}, ...more Script Type...]}
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

remover(*id_script_type*)

Remove Script Type from by the identifier.

Parameters **id_script_type** – Identifier of the Script Type. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – The identifier of Script Type is null and invalid.
- **TipoRoteiroNaoExisteError** – Script Type not registered.
- **TipoRoteiroError** – Script type is associated with a script.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.Usuario module

class networkapiclient.Usuario.**Usuario**(*networkapi_url*, *user*, *password*, *user_ldap=None*)

Bases: networkapiclient.GenericClient.GenericClient

alterar(*id_user*, *user*, *password*, *nome*, *ativo*, *email*, *user_ldap*)

Change User from by the identifier.

Parameters

- **id_user** – Identifier of the User. Integer value and greater than zero.
- **user** – Username. String with a minimum 3 and maximum of 45 characters
- **password** – User password. String with a minimum 3 and maximum of 45 characters
- **nome** – User name. String with a minimum 3 and maximum of 200 characters
- **email** – User Email. String with a minimum 3 and maximum of 300 characters
- **ativo** – Status. 0 or 1
- **user_ldap** – LDAP Username. String with a minimum 3 and maximum of 45 characters

Returns None

Raises

- **InvalidParameterError** – The identifier of User, user, pwd, name, email or active is null and invalid.
- **UserUsuarioDuplicadoError** – There is already a registered user with the value of user.
- **UsuarioNaoExisteError** – User not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

authenticate(*username*, *password*, *is_ldap_user*)

Get user by username and password and their permissions.

Parameters

- **username** – Username. String with a minimum 3 and maximum of 45 characters
- **password** – User password. String with a minimum 3 and maximum of 45 characters

Returns Following dictionary:

```
{'user': {'id': < id >}
{'user': < user >}
{'nome': < nome >}
{'pwd': < pwd >}
{'email': < email >}
{'active': < active >}
{'permission':[ {'<function>': { 'write': <value>, 'read': <value>}, ... more function ... ]
```

Raises

- **InvalidParameterError** – The value of username or password is invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

authenticate_ldap(*username*, *password*)

Get user by username and password and their permissions.

Parameters

- **username** – Username. String with a minimum 3 and maximum of 45 characters
- **password** – User password. String with a minimum 3 and maximum of 45 characters

Returns Following dictionary:

```
{'user': {'id': < id >}
{'user': < user >}
{'nome': < nome >}
{'pwd': < pwd >}
{'email': < email >}
{'active': < active >}
{'permission':[ {'<function>': { 'write': <value>, 'read': <value>}, ... more function ... ]
```

Raises

- **InvalidParameterError** – The value of username or password is invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

change_password(*id_user*, *user_current_password*, *password*)

Change password of User from by the identifier.

Parameters

- **id_user** – Identifier of the User. Integer value and greater than zero.
- **user_current_password** – Senha atual do usuário.
- **password** – Nova Senha do usuário.

Returns None

Raises

- **UsuarioNaoExisteError** – User not registered.
- **InvalidParameterError** – The identifier of User is null and invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

get_by_id(*id_user*)

Get user by the identifier and their user groups. is_more -If more than 3 of groups of users or no, to control expansion Screen.

Returns Dictionary with the following structure:

```
{'usuario': [{'nome': < nome >,
'id': < id >,
'pwd': < pwd >,
'user': < user >,
'ativo': < ativo >,
'email': < email >,
'grupos': [nome_grupo, ...more user groups...],
'user_ldap': < user_ldap >}}
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

get_by_user_ldap(*user_name*)

Get user by the ldap name. is_more -If more than 3 of groups of users or no, to control expansion Screen.

Returns Dictionary with the following structure:

```
{'usuario': [{'nome': < nome >,
'id': < id >,
'pwd': < pwd >,
'user': < user >,
'ativo': < ativo >,
'email': < email >,
'grupos': [nome_grupo, ...more user groups...],
'user_ldap': < user_ldap >}}
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

inserir(*user*, *pwd*, *name*, *email*, *user_ldap*)
Inserts a new User and returns its identifier.

The user will be created with active status.

Parameters

- **user** – Username. String with a minimum 3 and maximum of 45 characters
- **pwd** – User password. String with a minimum 3 and maximum of 45 characters
- **name** – User name. String with a minimum 3 and maximum of 200 characters
- **email** – User Email. String with a minimum 3 and maximum of 300 characters
- **user_ldap** – LDAP Username. String with a minimum 3 and maximum of 45 characters

Returns Dictionary with the following structure:

```
{'usuario': {'id': < id_user >}}
```

Raises

- **InvalidParameterError** – The identifier of User, user, pwd, name or email is null and invalid.
- **UserUsuarioDuplicadoError** – There is already a registered user with the value of user.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

list_by_group(*id_ugroup*)
Search Users by Group User by identifier.

Parameters **id_ugroup** – Identifier of the Group User. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{'users': [{'nome': < nome >, 'grupos': < grupos_id >,
'email': < email >, 'pwd': < pwd >,
'user': < user >, 'ativo': < ativo >,
'id': < id >}, ... ] }
```

Raises

- **InvalidParameterError** – Group User is null and invalid.

- **UGrupoNotFoundError** – Group User not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

list_by_group_out(*id_ugroup*)
Search Users out group by Group User by identifier.

Parameters **id_ugroup** – Identifier of the Group User. Integer value and greater than zero.

Returns Dictionary with the following structure:

```
{'users': [{'nome': < nome >, 'grupos': < grupos_id >,
'email': < email >, 'pwd': < pwd >,
'user': < user >, 'ativo': < ativo >,
'id': < id >}, ... ] }
```

Raises

- **InvalidParameterError** – Group User is null and invalid.
- **UGrupoNotFoundError** – Group User not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

list_with_usergroup()
List all users and their user groups. is_more -If more than 3 of groups of users or no, to control expansion Screen.

Returns Dictionary with the following structure:

```
{'usuario': [{'nome': < nome >,
'id': < id >,
'pwd': < pwd >,
'user': < user >,
'ativo': < ativo >,
'email': < email >,
'is_more': <True ou False>,
'grupos': [nome_grupo, ...more user groups...]}, ...more user...]}
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

listar()
List all user.

Returns Dictionary with the following structure:

```
{'usuario': [{'nome': < nome >,
'id': < id >,
'pwd': < pwd >,
'user': < user >,
'ativo': < ativo >,
'email': < email >,
'user_ldap': < ldap user >}, ...more user...]}
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

remover(*id_user*)
Remove User from by the identifier.

Parameters **id_user** – Identifier of the User. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – The identifier of User is null and invalid.
- **UsuarioNaoExisteError** – User not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.UsuarioGrupo module

class networkapiclient.UsuarioGrupo.**UsuarioGrupo**(*networkapi_url*, *user*, *password*, *user_ldap=None*)
Bases: networkapiclient.GenericClient.GenericClient

inserir(*id_user*, *id_group*)
Create a relationship between User and Group.

Parameters

- **id_user** – Identifier of the User. Integer value and greater than zero.
- **id_group** – Identifier of the Group. Integer value and greater than zero.

Returns Dictionary with the following structure:

:: {'user_group': {'id': < id_user_group >}}

Raises

- **InvalidParameterError** – The identifier of User or Group is null and invalid.
- **GrupoUsuarioNaoExisteError** – UserGroup not registered.
- **UsuarioNaoExisteError** – User not registered.
- **UsuarioGrupoError** – User already registered in the group.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

remover(*id_user*, *id_group*)
Removes relationship between User and Group.

Parameters

- **id_user** – Identifier of the User. Integer value and greater than zero.
- **id_group** – Identifier of the Group. Integer value and greater than zero.

Returns None

Raises

- **UsuarioGrupoNaoExisteError** – Association between user and group not registered.
- **GrupoUsuarioNaoExisteError** – UserGroup not registered.
- **UsuarioNaoExisteError** – User not registered.
- **UsuarioGrupoError** – User already registered in the group.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

networkapiclient.Vip module

class networkapiclient.Vip.**Vip**(*networkapi_url*, *user*, *password*, *user_ldap=None*)
Bases: networkapiclient.GenericClient.GenericClient

edit_reals(*id_vip*, *method_bal*, *reals*, *reals_prioritys*, *reals_weights*, *alter_priority=0*)
Execute the script 'gerador_vips' several times with options -real, -add and -del to adjust vip request reals. :param id_vip: Identifier of the VIP. Integer value and greater than zero. :param method_bal: method_bal. :param reals: List of reals. Ex: [{'real_name':'Teste1', 'real_ip':'10.10.10.1'},{'real_name':'Teste2', 'real_ip':'10.10.10.2'}] :param reals_prioritys: List of reals_priority. Ex: ['1','5','3']. :param reals_weights: List of reals_weight. Ex: ['1','5','3']. :param alter_priority: 1 if priority has changed and 0 if hasn't changed. :return: None :raise VipNaoExisteError: Request VIP not registered. :raise InvalidParameterError: Identifier of the request is invalid or null VIP. :raise DataBaseError: Networkapi failed to access the database. :raise XMLError: Networkapi failed to generate the XML response. :raise EnvironmentVipError: The combination of finality, client and environment is invalid. :raise InvalidTimeoutValueError: The value of timeout is invalid. :raise InvalidBalMethodValueError: The value of method_bal is invalid. :raise InvalidCacheValueError: The value of cache is invalid. :raise InvalidPersistenceValueError: The value of persistence is invalid. :raise InvalidPriorityValueError: One of the priority values is invalid. :raise EquipamentoNaoExisteError: The equipment associated with this Vip Request doesn't exist. :raise IpEquipmentError: Association between equipment and ip of this Vip Request doesn't exist. :raise IpError: IP not registered. :raise RealServerPriorityError: Vip Request priority list has an error. :raise RealServerWeightError: Vip Request weight list has an error. :raise RealServerPortError: Vip Request port list has an error. :raise RealParameterValueError: Vip Request real server parameter list has an error. :raise RealServerScriptError: Vip Request real server script execution error.

networkapiclient.Vlan module

class networkapiclient.Vlan.**Vlan**(*networkapi_url*, *user*, *password*, *user_ldap=None*)
Bases: networkapiclient.GenericClient.GenericClient

adicionar_permissao(*id_vlan*, *nome_equipamento*, *nome_interface*)
Add communication permission for VLAN to trunk.

Run script 'configurador'.

Parameters

- **id_vlan** – VLAN identifier.
- **nome_equipamento** – Equipment name.
- **nome_interface** – Interface name.

Returns Following dictionary:

```
{'sucesso': {'codigo': < codigo >,
'descricao': {'stdout':< stdout >, 'stderr':< stderr >}}}
```

Raises

- **VlanNaoExisteError** – VLAN does not exist.
- **InvalidParameterError** – Vlan id is invalid or none.
- **InvalidParameterError** – Equipment name and/or interface name is invalid or none.
- **EquipamentoNaoExisteError** – Equipment does not exist.
- **LigacaoFrontInterfaceNaoExisteError** – There is no interface on front link of informed interface.
- **InterfaceNaoExisteError** – Interface does not exist or is not associated to equipment.
- **LigacaoFrontNaoTerminaSwitchError** – Interface does not have switch connected.
- **InterfaceSwitchProtegidaError** – Switch interface is protected.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.
- **ScriptError** – Failed to run the script.

allocate_IPv6(*name*, *id_network_type*, *id_environment*, *description*, *id_environment_vip=None*)

Inserts a new VLAN.

Parameters

- **name** – Name of Vlan. String with a maximum of 50 characters.
- **id_network_type** – Identifier of the Netwok Type. Integer value and greater than zero.
- **id_environment** – Identifier of the Environment. Integer value and greater than zero.
- **description** – Description of Vlan. String with a maximum of 200 characters.
- **id_environment_vip** – Identifier of the Environment Vip. Integer value and greater than zero.

Returns Following dictionary:

```
{'vlan': {'id': < id_vlan >,
'nome': < nome_vlan >,
'num_vlan': < num_vlan >,
'id_tipo_rede': < id_tipo_rede >,
'id_ambiente': < id_ambiente >,
'bloco1': < bloco1 >,
'bloco2': < bloco2 >,
'bloco3': < bloco3 >,
'bloco4': < bloco4 >,
'bloco5': < bloco5 >,
'bloco6': < bloco6 >,
'bloco7': < bloco7 >,
'bloco8': < bloco8 >,
'bloco': < bloco >,
'mask_bloco1': < mask_bloco1 >,
'mask_bloco2': < mask_bloco2 >,
'mask_bloco3': < mask_bloco3 >,
```

```
'mask_bloco4': < mask_bloco4 >,
'mask_bloco5': < mask_bloco5 >,
'mask_bloco6': < mask_bloco6 >,
'mask_bloco7': < mask_bloco7 >,
'mask_bloco8': < mask_bloco8 >,
'descricao': < descricao >,
'acl_file_name': < acl_file_name >,
'acl_valida': < acl_valida >,
'acl_file_name_v6': < acl_file_name_v6 >,
'acl_valida_v6': < acl_valida_v6 >,
'ativada': < ativada >}}
```

Raises

- **VlanError** – VLAN name already exists, VLAN name already exists, DC division of the environment invalid or does not exist VLAN number available.
- **VlanNaoExisteError** – VLAN not found.
- **TipoRedeNaoExisteError** – Network Type not registered.
- **AmbienteNaoExisteError** – Environment not registered.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **InvalidParameterError** – Name of Vlan and/or the identifier of the Environment is null or invalid.
- **IPNaoDisponivelError** – There is no network address is available to create the VLAN.
- **ConfigEnvironmentInvalidError** – Invalid Environment Configuration or not registered
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

allocate_without_network(*environment_id*, *name*, *description*, *vrf=None*)

Create new VLAN without add NetworkIPv4.

Parameters

- **environment_id** – ID for Environment.
- **name** – The name of VLAN.
- **description** – Some description to VLAN.

Returns Following dictionary:

```
{'vlan': {'id': < id_vlan >,
'nome': < nome_vlan >,
'num_vlan': < num_vlan >,
'id_ambiente': < id_ambiente >,
'descricao': < descricao >,
'acl_file_name': < acl_file_name >,
'acl_valida': < acl_valida >,
'acl_file_name_v6': < acl_file_name_v6 >,
'acl_valida_v6': < acl_valida_v6 >,
'ativada': < ativada > } }
```

Raises

- **VlanError** – Duplicate name of VLAN, division DC of Environment not found/invalid or VLAN number not available.
- **AmbienteNaoExisteError** – Environment not found.
- **InvalidParameterError** – Some parameter was invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

alocar(*nome*, *id_tipo_rede*, *id_ambiente*, *descricao*, *id_ambiente_vip=None*, *vrf=None*)
Inserts a new VLAN.

Parameters

- **nome** – Name of Vlan. String with a maximum of 50 characters.
- **id_tipo_rede** – Identifier of the Network Type. Integer value and greater than zero.
- **id_ambiente** – Identifier of the Environment. Integer value and greater than zero.
- **descricao** – Description of Vlan. String with a maximum of 200 characters.
- **id_ambiente_vip** – Identifier of the Environment Vip. Integer value and greater than zero.

Returns Following dictionary:

```
{'vlan': {'id': < id_vlan >,
'nome': < nome_vlan >,
'num_vlan': < num_vlan >,
'id_tipo_rede': < id_tipo_rede >,
'id_ambiente': < id_ambiente >,
'rede_oct1': < rede_oct1 >,
'rede_oct2': < rede_oct2 >,
'rede_oct3': < rede_oct3 >,
'rede_oct4': < rede_oct4 >,
'bloco': < bloco >,
'mascara_oct1': < mascara_oct1 >,
'mascara_oct2': < mascara_oct2 >,
'mascara_oct3': < mascara_oct3 >,
'mascara_oct4': < mascara_oct4 >,
'broadcast': < broadcast >,
'descricao': < descricao >,
'acl_file_name': < acl_file_name >,
'acl_valida': < acl_valida >,
'ativada': < ativada >}}
```

Raises

- **VlanError** – VLAN name already exists, VLAN name already exists, DC division of the environment invalid or does not exist VLAN number available.
- **VlanNaoExisteError** – VLAN not found.
- **TipoRedeNaoExisteError** – Network Type not registered.
- **AmbienteNaoExisteError** – Environment not registered.
- **EnvironmentVipNotFoundError** – Environment VIP not registered.
- **InvalidParameterError** – Name of Vlan and/or the identifier of the Environment is null or invalid.
- **IPNaoDisponivelError** – There is no network address is available to create the VLAN.

- **ConfigEnvironmentInvalidError** – Invalid Environment Configuration or not registered
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

apply_acl(*equipments*, *vlan*, *environment*, *network*)
Apply the file acl in equipments

Parameters

- **equipments** – list of equipments
- **vlan** – Vvlan
- **environment** – Environment
- **network** – v4 or v6

Raises Exception Failed to apply acl

Returns True case Apply and sysout of script

buscar(*id_vlan*)
Get VLAN by its identifier.

Parameters **id_vlan** – VLAN identifier.

Returns Following dictionary:

```
{'vlan': {'id': < id_vlan >,
'nome': < nome_vlan >,
'num_vlan': < num_vlan >,
'id_ambiente': < id_ambiente >,
'id_tipo_rede': < id_tipo_rede >,
'rede_oct1': < rede_oct1 >,
'rede_oct2': < rede_oct2 >,
'rede_oct3': < rede_oct3 >,
'rede_oct4': < rede_oct4 >,
'bloco': < bloco >,
'mascara_oct1': < mascara_oct1 >,
'mascara_oct2': < mascara_oct2 >,
'mascara_oct3': < mascara_oct3 >,
'mascara_oct4': < mascara_oct4 >,
'broadcast': < broadcast >,
'descricao': < descricao >,
'acl_file_name': < acl_file_name >,
'acl_valida': < acl_valida >,
'ativada': < ativada >}
OR {'id': < id_vlan >,
'nome': < nome_vlan >,
'num_vlan': < num_vlan >,
'id_tipo_rede': < id_tipo_rede >,
'id_ambiente': < id_ambiente >,
'bloco1': < bloco1 >,
'bloco2': < bloco2 >,
'bloco3': < bloco3 >,
'bloco4': < bloco4 >,
'bloco5': < bloco5 >,
'bloco6': < bloco6 >,
'bloco7': < bloco7 >,
'bloco8': < bloco8 >,
'bloco': < bloco >,
```

```
'mask_bloco1': < mask_bloco1 >,
'mask_bloco2': < mask_bloco2 >,
'mask_bloco3': < mask_bloco3 >,
'mask_bloco4': < mask_bloco4 >,
'mask_bloco5': < mask_bloco5 >,
'mask_bloco6': < mask_bloco6 >,
'mask_bloco7': < mask_bloco7 >,
'mask_bloco8': < mask_bloco8 >,
'broadcast': < broadcast >,
'descricao': < descricao >,
'acl_file_name': < acl_file_name >,
'acl_valida': < acl_valida >,
'acl_file_name_v6': < acl_file_name_v6 >,
'acl_valida_v6': < acl_valida_v6 >,
'ativada': < ativada >}}
```

Raises

- **VlanNaoExisteError** – VLAN does not exist.
- **InvalidParameterError** – VLAN id is none or invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

check_number_available(*id_environment*, *num_vlan*, *id_vlan*)

Checking if environment has a number vlan available

Parameters

- **id_environment** – Identifier of environment
- **num_vlan** – Vlan number
- **id_vlan** – Vlan indentifier (False if inserting a vlan)

Returns True is has number available, False if hasn't

Raises

- **AmbienteNaoExisteError** – Ambiente não cadastrado.
- **InvalidParameterError** – Invalid ID for VLAN.
- **VlanNaoExisteError** – VLAN not found.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

confirm_vlan(*number_net*, *id_environment_vlan*, *ip_version=None*)

Checking if the vlan insert need to be confirmed

Parameters

- **number_net** – Filter by vlan number column
- **id_environment_vlan** – Filter by environment ID related
- **ip_version** – Ip version for checking

Returns True is need confirmation, False if no need

Raises

- **AmbienteNaoExisteError** – Ambiente não cadastrado.
- **InvalidParameterError** – Invalid ID for VLAN.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

create_acl(*id_vlan*, *network_type*)

Create the file acl

Parameters

- **id_vlan** – Vlan Id
- **network_type** – v4 or v6

Raises

- **InvalidValueError** – Attrs invalids.
- **XMLError** – Networkapi failed to generate the XML response.
- **AclNotFoundError** – ACL not created.
- **VlanNotFoundError** – Vlan not registered.

Returns Following dictionary:

```
{'vlan': {
'acl_file_name': < acl_file_name >,
'ativada': < ativada >,
'acl_valida': < acl_valida >,
'nome': '< nome >',
'acl_file_name_v6': < acl_file_name_v6 >,
'acl_valida_v6': < acl_valida_v6 >,
'ambiente': < ambiente >,
'num_vlan': < num_vlan >,
'id': < id >,
'descricao': < descricao >
}}
```

create_ipv4(*id_network_ipv4*)

Create VLAN in layer 2 using script 'navlan'.

Parameters **id_network_ipv4** – NetworkIPv4 ID.

Returns Following dictionary:

```
{'sucesso': {'codigo': < codigo >,
'descricao': {'stdout':< stdout >, 'stderr':< stderr >}}}
```

Raises

- **NetworkIPv4NaoExisteError** – NetworkIPv4 not found.
- **EquipamentoNaoExisteError** – Equipament in list not found.
- **VlanError** – VLAN is active.
- **InvalidParameterError** – VLAN identifier is none or invalid.
- **InvalidParameterError** – Equipment list is none or empty.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

- **ScriptError** – Failed to run the script.

create_ipv6(*id_network_ipv6*)
Create VLAN in layer 2 using script 'navlan'.

Parameters **id_network_ipv6** – NetworkIPv6 ID.

Returns Following dictionary:

```
{'sucesso': {'codigo': < codigo >,
'descricao': {'stdout':< stdout >, 'stderr':< stderr >}}}
```

Raises

- **NetworkIPv6NaoExisteError** – NetworkIPv6 not found.
- **EquipamentoNaoExisteError** – Equipament in list not found.
- **VlanError** – VLAN is active.
- **InvalidParameterError** – VLAN identifier is none or invalid.
- **InvalidParameterError** – Equipment list is none or empty.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.
- **ScriptError** – Failed to run the script.

create_script_acl(*id_vlan*, *network_type*)
Generate the script acl

Parameters

- **id_vlan** – Vlan Id
- **network_type** – v4 or v6

Raises

- **InvalidValueError** – Attrs invalids.
- **XMLError** – Networkapi failed to generate the XML response.
- **VlanACLDuplicatedError** – ACL name duplicate.
- **VlanNotFoundError** – Vlan not registered.

Returns Following dictionary:

```
{'vlan': {
'id': < id >,
'nome': '< nome >',
'num_vlan': < num_vlan >,
'descricao': < descricao >
'acl_file_name': < acl_file_name >,
'ativada': < ativada >,
'acl_valida': < acl_valida >,
'acl_file_name_v6': < acl_file_name_v6 >,
'redeipv6': < redeipv6 >,
'acl_valida_v6': < acl_valida_v6 >,
'redeipv4': < redeipv4 >,
'ambiente': < ambiente >,
}}
```

create_vlan(*id_vlan*)
Set column 'ativada = 1'.

Parameters **id_vlan** – VLAN identifier.

Returns None

criar(*id_vlan*)
Create a VLAN with script 'navlan'.

Parameters **id_vlan** – VLAN identifier.

Returns Following dictionary:

```
{'sucesso': {'codigo': < codigo >,
'descricao': {'stdout':< stdout >, 'stderr':< stderr >}}}
```

Raises

- **VlanNaoExisteError** – VLAN does not exist.
- **EquipamentoNaoExisteError** – Equipment in list does not exist.
- **VlanError** – VLAN is active.
- **InvalidParameterError** – VLAN identifier is none or invalid.
- **InvalidParameterError** – Equipment list is none or empty.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.
- **ScriptError** – Failed to run the script.

deallocate(*id_vlan*)
Deallocate all relationships between Vlan.

Parameters **id_vlan** – Identifier of the VLAN. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – VLAN identifier is null and invalid.
- **VlanError** – VLAN is active.
- **VlanNaoExisteError** – VLAN not found.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

edit_vlan(*environment_id*, *name*, *number*, *description*, *acl_file*, *acl_file_v6*, *id_vlan*)
Edit a VLAN

Parameters

- **id_vlan** – ID for Vlan
- **environment_id** – ID for Environment.
- **name** – The name of VLAN.
- **description** – Some description to VLAN.
- **number** – Number of Vlan

- **acl_file** – Acl IPv4 File name to VLAN.
- **acl_file_v6** – Acl IPv6 File name to VLAN.

Returns None

Raises

- **VlanError** – VLAN name already exists, DC division of the environment invalid or there is no VLAN number available.
- **VlanNaoExisteError** – VLAN not found.
- **AmbienteNaoExisteError** – Environment not registered.
- **InvalidParameterError** – Name of Vlan and/or the identifier of the Environment is null or invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

find_vlans(*number*, *name*, *iexact*, *environment*, *net_type*, *network*, *ip_version*, *subnet*, *acl*, *pagination*)

Find vlans by all search parameters

Parameters

- **number** – Filter by vlan number column
- **name** – Filter by vlan name column
- **iexact** – Filter by name will be exact?
- **environment** – Filter by environment ID related
- **net_type** – Filter by network_type ID related
- **network** – Filter by each octs in network
- **ip_version** – Get only version (0:ipv4, 1:ipv6, 2:all)
- **subnet** – Filter by octs will search by subnets?
- **acl** – Filter by vlan acl column
- **pagination** – Class with all data needed to paginate

Returns Following dictionary:

```
{'vlan': {'id': < id_vlan >,
'nome': < nome_vlan >,
'num_vlan': < num_vlan >,
'id_ambiente': < id_ambiente >,
'descricao': < descricao >,
'acl_file_name': < acl_file_name >,
'acl_valida': < acl_valida >,
'acl_file_name_v6': < acl_file_name_v6 >,
'acl_valida_v6': < acl_valida_v6 >,
'ativada': < ativada >,
'ambiente_name': < divisao_dc-ambiente_logico-grupo_l3 >
'redeipv4': [ { all networkipv4 related } ],
'redeipv6': [ { all networkipv6 related } ] },
'total': {< total_registros >} }
```

Raises

- **InvalidParameterError** – Some parameter was invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

get(*id_vlan*)

Get a VLAN by your primary key. Network IPv4/IPv6 related will also be fetched.

Parameters **id_vlan** – ID for VLAN.

Returns Following dictionary:

```
{'vlan': {'id': < id_vlan >,
'nome': < nome_vlan >,
'num_vlan': < num_vlan >,
'id_ambiente': < id_ambiente >,
'descricao': < descricao >,
'acl_file_name': < acl_file_name >,
'acl_valida': < acl_valida >,
'acl_file_name_v6': < acl_file_name_v6 >,
'acl_valida_v6': < acl_valida_v6 >,
'ativada': < ativada >,
'redeipv4': [ { all networkipv4 related } ],
'redeipv6': [ { all networkipv6 related } ] } }
```

Raises

- **InvalidParameterError** – Invalid ID for VLAN.
- **VlanNaoExisteError** – VLAN not found.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

insert_vlan(*environment_id*, *name*, *number*, *description*, *acl_file*, *acl_file_v6*, *network_ipv4*, *network_ipv6*, *vrf=None*)

Create new VLAN

Parameters

- **environment_id** – ID for Environment.
- **name** – The name of VLAN.
- **description** – Some description to VLAN.
- **number** – Number of Vlan
- **acl_file** – Acl IPv4 File name to VLAN.
- **acl_file_v6** – Acl IPv6 File name to VLAN.
- **network_ipv4** – responsible for generating a network attribute ipv4 automatically.
- **network_ipv6** – responsible for generating a network attribute ipv6 automatically.

Returns Following dictionary:

```
{'vlan': {'id': < id_vlan >,
'nome': < nome_vlan >,
'num_vlan': < num_vlan >,
'id_ambiente': < id_ambiente >,
'descricao': < descricao >,
```

```
'acl_file_name': < acl_file_name >,
'acl_valida': < acl_valida >,
'ativada': < ativada >
'acl_file_name_v6': < acl_file_name_v6 >,
'acl_valida_v6': < acl_valida_v6 >, } }
```

Raises

- **VlanError** – VLAN name already exists, VLAN name already exists, DC division of the environment invalid or does not exist VLAN number available.
- **VlanNaoExisteError** – VLAN not found.
- **AmbienteNaoExisteError** – Environment not registered.
- **InvalidParameterError** – Name of Vlan and/or the identifier of the Environment is null or invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

invalidate(*id_vlan*)

Invalidates ACL - IPv4 of VLAN from its identifier.

Assigns 0 to 'acl_valida' and null to 'acl_file_name'.

Parameters **id_vlan** – Identifier of the Vlan. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – Vlan identifier is null and invalid.
- **VlanNaoExisteError** – Vlan not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

invalidate_ipv6(*id_vlan*)

Invalidates ACL - IPv6 of VLAN from its identifier.

Assigns 0 to 'acl_valida_v6' and null to 'acl_file_name_v6'.

Parameters **id_vlan** – Identifier of the Vlan. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – Vlan identifier is null and invalid.
- **VlanNaoExisteError** – Vlan not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

list_all()

List all vlans

Returns Following dictionary:

```
{'vlan': [{'id': < id_vlan >,
'name': < nome_vlan >} {... demais vlans ...} ] }
```

Raises

- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

listar_permissao(*nome_equipamento*, *nome_interface*)

List all VLANS having communication permission to trunk from a port in switch.

Run script 'configurador'.

```
The value of 'stdout' key of return dictionary can have a list of numbers or
number intervals of VLAN´s, comma separated. Examples of possible returns of 'stdout' below:
    - 100,103,111,...
    - 100-110,...
    - 100-110,112,115,...
    - 100,103,105-111,113,115-118,...
```

Parameters

- **nome_equipamento** – Equipment name.
- **nome_interface** – Interface name.

Returns Following dictionary:

```
{'sucesso': {'codigo': < codigo >,
'descricao': {'stdout':< stdout >, 'stderr':< stderr >}}}
```

Raises

- **InvalidParameterError** – Equipment name and/or interface name is invalid or none.
- **EquipamentoNaoExisteError** – Equipment does not exist.
- **LigacaoFrontInterfaceNaoExisteError** – There is no interface on front link of informed interface.
- **InterfaceNaoExisteError** – Interface does not exist or is not associated to equipment.
- **LigacaoFrontNaoTerminaSwitchError** – Interface does not have switch connected.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.
- **ScriptError** – Failed to run the script.

listar_por_ambiente(*id_ambiente*)

List all VLANs from an environment. ** The itens returning from network is there to be compatible with other system ** :param id_ambiente: Environment identifier.

Returns Following dictionary:

```
{'vlan': [{'id': < id_vlan >,
'nome': < nome_vlan >,
'num_vlan': < num_vlan >,
'ambiente': < id_ambiente >,
'descricao': < descricao >,
```

```
'acl_file_name': < acl_file_name >,
'acl_valida': < acl_valida >,
'acl_file_name_v6': < acl_file_name_v6 >,
'acl_valida_v6': < acl_valida_v6 >,
'ativada': < ativada >,
'id_tipo_rede': < id_tipo_rede >,
'rede_oct1': < rede_oct1 >,
'rede_oct2': < rede_oct2 >,
'rede_oct3': < rede_oct3 >,
'rede_oct4': < rede_oct4 >,
'bloco': < bloco >,
'mascara_oct1': < mascara_oct1 >,
'mascara_oct2': < mascara_oct2 >,
'mascara_oct3': < mascara_oct3 >,
'mascara_oct4': < mascara_oct4 >,
'broadcast': < broadcast >,} , ... other vlans ... ]}
```

Raises

- **InvalidParameterError** – Environment id is none or invalid.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

remove(*id_vlan*)

Remove a VLAN by your primary key. Execute script to remove VLAN

Parameters **id_vlan** – ID for VLAN.

Returns Following dictionary:

```
{'sucesso': {'codigo': < codigo >,
'descricao': {'stdout':< stdout >, 'stderr':< stderr >}}}
```

Raises

- **InvalidParameterError** – Identifier of the VLAN is invalid.
- **VlanNaoExisteError** – VLAN not found.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

remover_permissao(*id_vlan*, *nome_equipamento*, *nome_interface*)

Remove communication permission for VLAN to trunk.

Run script 'configurador'.

Parameters

- **id_vlan** – VLAN identifier.
- **nome_equipamento** – Equipment name.
- **nome_interface** – Interface name.

Returns Following dictionary:

```
{'sucesso': {'codigo': < codigo >,
'descricao': {'stdout':< stdout >, 'stderr':< stderr >}}}
```

Raises

- **VlanNaoExisteError** – VLAN does not exist.
- **InvalidParameterError** – VLAN id is none or invalid.
- **InvalidParameterError** – Equipment name and/or interface name is invalid or none.
- **EquipamentoNaoExisteError** – Equipment does not exist.
- **LigacaoFrontInterfaceNaoExisteError** – There is no interface on front link of informed interface.
- **InterfaceNaoExisteError** – Interface does not exist or is not associated to equipment.
- **LigacaoFrontNaoTerminaSwitchError** – Interface does not have switch connected.
- **InterfaceSwitchProtegidaError** – Switch interface is protected.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.
- **ScriptError** – Failed to run the script.

validar(*id_vlan*)

Validates ACL - IPv4 of VLAN from its identifier.

Assigns 1 to 'acl_valida'.

Parameters **id_vlan** – Identifier of the Vlan. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – Vlan identifier is null and invalid.
- **VlanNaoExisteError** – Vlan not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

validate_ipv6(*id_vlan*)

Validates ACL - IPv6 of VLAN from its identifier.

Assigns 1 to 'acl_valida_v6'.

Parameters **id_vlan** – Identifier of the Vlan. Integer value and greater than zero.

Returns None

Raises

- **InvalidParameterError** – Vlan identifier is null and invalid.
- **VlanNaoExisteError** – Vlan not registered.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.

verificar_permissao(*id_vlan*, *nome_equipamento*, *nome_interface*)

Check if there is communication permission for VLAN to trunk.

Run script 'configurador'.

The "stdout" key value of response dictionary is 1(one) if VLAN has permission, or 0(zero), otherwise.

Parameters

- **id_vlan** – VLAN identifier.
- **nome_equipamento** – Equipment name.
- **nome_interface** – Interface name.

Returns Following dictionary:

```
{'sucesso': {'codigo': < codigo >,
'descricao': {'stdout':< stdout >, 'stderr':< stderr >}}}
```

Raises

- **VlanNaoExisteError** – VLAN does not exist.
- **InvalidParameterError** – VLAN id is none or invalid.
- **InvalidParameterError** – Equipment name and/or interface name is invalid or none.
- **EquipamentoNaoExisteError** – Equipment does not exist.
- **LigacaoFrontInterfaceNaoExisteError** – There is no interface on front link of informed interface.
- **InterfaceNaoExisteError** – Interface does not exist or is not associated to equipment.
- **LigacaoFrontNaoTerminaSwitchError** – Interface does not have switch connected.
- **DataBaseError** – Networkapi failed to access the database.
- **XMLError** – Networkapi failed to generate the XML response.
- **ScriptError** – Failed to run the script.

networkapiclient.exception module

exception networkapiclient.exception.**AmbienteDuplicadoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**AmbienteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**AmbienteLogicoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**AmbienteLogicoNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**AmbienteNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**CantDissociateError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

returns exception whey trying to dissociate filter and equipment type, and some environment is using the filter

exception networkapiclient.exception.**ConfigEnvironmentInvalidError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**DataBaseError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**DetailedEnvironmentError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

returns exception to environment cant be removed because vlan has vip request association

exception networkapiclient.exception.**DireitoGrupoEquipamentoDuplicadoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**DireitoGrupoEquipamentoNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**DivisaoDcError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**DivisaoDcNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**EnvironmentVipError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

returns exception to environment vip.

exception networkapiclient.exception.**EnvironmentVipNotFoundError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

returns exception to environment research by primary key.

exception networkapiclient.exception.**EquipamentoAcessoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**EquipamentoAcessoNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**EquipamentoAmbienteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**EquipamentoAmbienteNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**EquipamentoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**EquipamentoGrupoNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**EquipamentoNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**EquipamentoRoteiroError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**EquipamentoRoteiroNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**EquipmentDontRemoveError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

class networkapiclient.exception.**ErrorHandler**
Bases: object

Classe que trata os códigos de erros retornados pela networkAPI e lança a exceção correspondente na networkAPI-Client.

errors = {1: <class 'networkapiclient.exception.DataBaseError'>, 2: <class 'networkapiclient.exception.ScriptError'>,

classmethod handle(*code*, *description*)
Recebe o código e a descrição do erro da networkAPI e lança a exceção correspondente.

Parameters

- **code** – Código de erro retornado pela networkAPI.
- **description** – Descrição do erro.

Returns None

exception networkapiclient.exception.**FilterDuplicateError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

returns exception of filter name already exist

exception networkapiclient.exception.**FilterEqTypeAssociationError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

returns exception of filter and equip type already exist

exception networkapiclient.exception.**FilterNotFoundError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

returns exception of filter search

exception networkapiclient.exception.**GroupDontRemoveError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**GrupoEquipamentoNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**GrupoL3Error**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**GrupoL3NaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**GrupoUsuarioNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**HealthCheckExpectJaCadastradoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**HealthCheckExpectNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**IPNaoDisponivelError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**InterfaceError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**InterfaceInvalidBackFrontError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**InterfaceNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**InterfaceSwitchProtegidaError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**InvalidBalMethodValueError**(*error*)
Bases: networkapiclient.exception.InvalidParameterError

returns exception of invalid balancing method value

exception networkapiclient.exception.**InvalidCacheValueError**(*error*)
Bases: networkapiclient.exception.InvalidParameterError

returns exception of invalid cache value

exception networkapiclient.exception.**InvalidFinalityValueError**(*error*)
Bases: networkapiclient.exception.InvalidParameterError

returns exception of invalid finality value

exception networkapiclient.exception.**InvalidParameterError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**InvalidPersistenceValueError**(*error*)
Bases: networkapiclient.exception.InvalidParameterError

returns exception of invalid persistence value

exception networkapiclient.exception.**InvalidPriorityValueError**(*error*)
Bases: networkapiclient.exception.InvalidParameterError

returns exception of invalid priority value

exception networkapiclient.exception.**InvalidRequestError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**InvalidTimeoutValueError**(*error*)
Bases: networkapiclient.exception.InvalidParameterError

returns exception of invalid timeout value

exception networkapiclient.exception.**IpEquipCantDissociateFromVip**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

Returns exception when trying to dissociate ip and equipment, but equipment is the last balancer for Vip Request

exception networkapiclient.exception.**IpEquipmentError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

returns exception of invalid ip and equipment association

exception networkapiclient.exception.**IpError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**IpNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**IpNotFoundByEquipAndVipError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

returns exception of real server error

exception networkapiclient.exception.**IpRangeAlreadyAssociation**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

returns exception whey trying to associate ip and equipment, and equipment having another ip in the same ip range

exception networkapiclient.exception.**LigacaoFrontInterfaceNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**LigacaoFrontNaoTerminaSwitchError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**MarcaNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**MarcarError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**ModeloEquipamentoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**ModeloEquipamentoNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**NetworkAPIClientError**(*error*)
Bases: exceptions.Exception

exception networkapiclient.exception.**NetworkIPRangeEnvError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

returns exception of two environments having the same ip range

exception networkapiclient.exception.**NomeAmbienteLogicoDuplicadoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**NomeDivisaoDcDuplicadoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**NomeGrupoEquipamentoDuplicadoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**NomeGrupoL3DuplicadoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**NomeGrupoUsuarioDuplicadoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**NomeInterfaceDuplicadoParaEquipamentoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**NomeMarcaDuplicadoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**NomeMarcaModeloDuplicadoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**NomeRackDuplicadoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**NomeRoteiroDuplicadoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**NomeTipoRedeDuplicadoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**NomeTipoRoteiroDuplicadoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**NotImplementedError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**NumeroRackDuplicadoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**OptionVipError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

returns exception to option vip.

exception networkapiclient.exception.**OptionVipNotFoundError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

returns exception to option research by primary key.

exception networkapiclient.exception.**PermissaoAdministrativaDuplicadaError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**PermissaoAdministrativaNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**PermissionNotFoundError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

returns exception of filter search

exception networkapiclient.exception.**ProtocoloTipoAcessoDuplicadoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**RackAllreadyConfigError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**RackAplicarError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**RackConfiguracaoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**RackNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**RacksError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**RealParameterValueError**(*error*)
Bases: networkapiclient.exception.InvalidParameterError

returns exception of invalid real server parameter

exception networkapiclient.exception.**RealServerPortError**(*error*)
Bases: networkapiclient.exception.InvalidParameterError

returns exception of invalid real server port list

exception networkapiclient.exception.**RealServerPriorityError**(*error*)
Bases: networkapiclient.exception.InvalidParameterError

returns exception of invalid real server priority list

exception networkapiclient.exception.**RealServerScriptError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

returns exception of real server script error

exception networkapiclient.exception.**RealServerWeightError**(*error*)
Bases: networkapiclient.exception.InvalidParameterError

returns exception of invalid real server weight list

exception networkapiclient.exception.**RelacionamentoInterfaceEquipamentoNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**RoteiroError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**RoteiroNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**ScriptError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**TipoAcessoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**TipoAcessoNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**TipoEquipamentoNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**TipoRedeError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**TipoRedeNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**TipoRoteiroError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**TipoRoteiroNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**UrlNotFoundError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**UserNotAuthenticatedError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**UserNotAuthorizedError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**UserUsuarioDuplicadoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**UsuarioError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**UsuarioGrupoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**UsuarioGrupoNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**UsuarioNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**ValorIndicacaoDireitoInvalidoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**ValorIndicacaoPermissaoInvalidoError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**VariableError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**VipAllreadyCreateError**(*error*)
Bases: networkapiclient.exception.VipError

exception networkapiclient.exception.**VipError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**VipIpError**(*error*)
Bases: networkapiclient.exception.VlanError

returns exception to ip cant removed from vip.

exception networkapiclient.exception.**VipNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**VipRequestBlockAlreadyInRule**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

Returns exception when trying to add a block that already exists in rule vip

exception networkapiclient.exception.**VipRequestNoBlockInRule**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

Returns exception when trying to add a block in rule vip that doesn't has any block

exception networkapiclient.exception.**VipVersaoIPError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**VlanAclExistenteError**(*error*)
Bases: networkapiclient.exception.VlanError

returns exception to find a Vlan with a ACL-file that already exists.

exception networkapiclient.exception.**VlanError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**VlanNaoExisteError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

exception networkapiclient.exception.**XMLError**(*error*)
Bases: networkapiclient.exception.NetworkAPIClientError

networkapiclient.rest module

exception networkapiclient.rest.**ConnectionError**(*cause*)
Bases: networkapiclient.rest.RestError

Caso ocorra algum erro de conexão com a NetworkAPI.

class networkapiclient.rest.**Rest**
Classe utilitária para chamada de webservices REST.

Implementa métodos utilitários para realizações de chamadas a webservices REST.

delete(*url*, *request_data*, *content_type=None*, *auth_map=None*)
Envia uma requisição DELETE para a URL informada.

Se auth_map é diferente de None, então deverá conter as chaves NETWORKAPI_PASSWORD e NETWORKAPI_USERNAME para realizar a autenticação na networkAPI.

As chaves e os seus valores são enviados no header da requisição.

Parameters

- **url** – URL para enviar a requisição HTTP.
- **request_data** – Descrição para enviar no corpo da requisição HTTP.

- **content_type** – Tipo do conteúdo enviado em request_data. O valor deste parâmetro será adicionado no header "Content-Type" da requisição.
- **auth_map** – Dicionário com as informações para autenticação na networkAPI.

Returns Retorna uma tupla contendo: (< código de resposta http >, < corpo da resposta >).

Raises

- **ConnectionError** – Falha na conexão com a networkAPI.
- **RestError** – Falha no acesso à networkAPI.

delete_map(*url*, *map=None*, *auth_map=None*)

Gera um XML a partir dos dados do dicionário e o envia através de uma requisição DELETE.

Parameters

- **url** – URL para enviar a requisição HTTP.
- **map** – Dicionário com os dados do corpo da requisição HTTP.
- **auth_map** – Dicionário com as informações para autenticação na networkAPI.

Returns Retorna uma tupla contendo: (< código de resposta http >, < corpo da resposta >).

Raises

- **ConnectionError** – Falha na conexão com a networkAPI.
- **RestError** – Falha no acesso à networkAPI.

get(*url*, *auth_map=None*)

Envia uma requisição GET para a URL informada.

Se auth_map é diferente de None, então deverá conter as chaves NETWORKAPI_PASSWORD e NETWORKAPI_USERNAME para realizar a autenticação na networkAPI. As chaves e os seus valores são enviados no header da requisição.

Parameters

- **url** – URL para enviar a requisição HTTP.
- **auth_map** – Dicionário com as informações para autenticação na networkAPI.

Returns Retorna uma tupla contendo: (< código de resposta http >, < corpo da resposta >).

Raises

- **ConnectionError** – Falha na conexão com a networkAPI.
- **RestError** – Falha no acesso à networkAPI.

get_full_url(*parsed_url*)

Returns url path with querystring

get_map(*url*, *auth_map=None*)

Envia uma requisição GET.

Parameters

- **url** – URL para enviar a requisição HTTP.
- **auth_map** – Dicionário com as informações para autenticação na networkAPI.

Returns Retorna uma tupla contendo: (< código de resposta http >, < corpo da resposta >).

Raises

- **ConnectionError** – Falha na conexão com a networkAPI.
- **RestError** – Falha no acesso à networkAPI.

post(*url*, *request_data*, *content_type=None*, *auth_map=None*)
Envia uma requisição POST para a URL informada.

Se auth_map é diferente de None, então deverá conter as chaves NETWORKAPI_PASSWORD e NETWORKAPI_USERNAME para realizar a autenticação na networkAPI.

As chaves e os seus valores são enviados no header da requisição.

Parameters

- **url** – URL para enviar a requisição HTTP.
- **request_data** – Descrição para enviar no corpo da requisição HTTP.
- **content_type** – Tipo do conteúdo enviado em request_data. O valor deste parâmetro será adicionado no header "Content-Type" da requisição.
- **auth_map** – Dicionário com as informações para autenticação na networkAPI.

Returns Retorna uma tupla contendo:

```
(< código de resposta http >, < corpo da resposta >).
```

Raises

- **ConnectionError** – Falha na conexão com a networkAPI.
- **RestError** – Falha no acesso à networkAPI.

post_map(*url*, *map*, *auth_map=None*)
Gera um XML a partir dos dados do dicionário e o envia através de uma requisição POST.

Parameters

- **url** – URL para enviar a requisição HTTP.
- **map** – Dicionário com os dados do corpo da requisição HTTP.
- **auth_map** – Dicionário com as informações para autenticação na networkAPI.

Returns Retorna uma tupla contendo: (< código de resposta http >, < corpo da resposta >).

Raises

- **ConnectionError** – Falha na conexão com a networkAPI.
- **RestError** – Falha no acesso à networkAPI.

put(*url*, *request_data*, *content_type=None*, *auth_map=None*)
Envia uma requisição PUT para a URL informada.

Se auth_map é diferente de None, então deverá conter as chaves NETWORKAPI_PASSWORD e NETWORKAPI_USERNAME para realizar a autenticação na networkAPI.

As chaves e os seus valores são enviados no header da requisição.

Parameters

- **url** – URL para enviar a requisição HTTP.
- **request_data** – Descrição para enviar no corpo da requisição HTTP.

- **content_type** – Tipo do conteúdo enviado em request_data. O valor deste parâmetro será adicionado no header "Content-Type" da requisição.
- **auth_map** – Dicionário com as informações para autenticação na networkAPI.

Returns Retorna uma tupla contendo:

`(< código de resposta http >, < corpo da resposta >).`

Raises

- **ConnectionError** – Falha na conexão com a networkAPI.
- **RestError** – Falha no acesso à networkAPI.

put_map(*url*, *map*, *auth_map=None*)

Gera um XML a partir dos dados do dicionário e o envia através de uma requisição PUT.

Parameters

- **url** – URL para enviar a requisição HTTP.
- **map** – Dicionário com os dados do corpo da requisição HTTP.
- **auth_map** – Dicionário com as informações para autenticação na networkAPI.

Returns Retorna uma tupla contendo: (< código de resposta http>, < corpo da resposta>).

Raises

- **ConnectionError** – Falha na conexão com a networkAPI.
- **RestError** – Falha no acesso à networkAPI.

unmarshall(*content*)

exception networkapiclient.rest.**RestError**(*cause*, *message*)

Bases: exceptions.Exception

Representa um erro ocorrido durante uma requisão REST.

class networkapiclient.rest.**RestRequest**(*url*, *method*, *user*, *password*, *user_ldap=None*)

Classe básica para requisições webservices REST à networkAPI

submit(*map*)

Envia a requisição HTTP de acordo com os parâmetros informados no construtor.

Parameters **map** – Dicionário com os dados do corpo da requisição.

Returns Retorna uma tupla contendo: (< código de resposta http >, < corpo da resposta >).

Raises

- **ConnectionError** – Falha na conexão com a networkAPI.
- **RestError** – Falha no acesso à networkAPI.

networkapiclient.utils module

networkapiclient.utils.**build_uri_with_ids**(*prefix*, *ids*)

networkapiclient.utils.**get_list_map**(*map*, *key*)
Se o mapa é diferente de None então retorna o mapa, caso contrário, cria o mapa {key:[]}.

O mapa criado terá o elemento 'key' com uma lista vazia.

Parameters

- **map** – Mapa onde o elemento tem como valor uma lista.
- **key** – Chave para criar o mapa {key:[]}

Returns Retorna um mapa onde o elemento tem como valor uma lista.

networkapiclient.utils.**is_valid_0_1**(*param*)
Checks if param is zero or one

networkapiclient.utils.**is_valid_int_param**(*param*)
Verifica se o parâmetro é um valor inteiro válido.

Parameters **param** – Valor para ser validado.

Returns True se o parâmetro tem um valor inteiro válido, ou False, caso contrário.

networkapiclient.utils.**is_valid_ip**(*address*)
Verifica se address é um endereço ip válido.

O valor é considerado válido se tiver no formato XXX.XXX.XXX.XXX, onde X é um valor entre 0 e 9.

Parameters **address** – Endereço IP.

Returns True se o parâmetro é um IP válido, ou False, caso contrário.

networkapiclient.utils.**is_valid_version_ip**(*param*)
Checks if the parameter is a valid ip version value.

Parameters **param** – Value to be validated.

Returns True if the parameter has a valid ip version value, or False otherwise.

networkapiclient.version_control module

networkapiclient.xml_utils module

exception networkapiclient.xml_utils.**InvalidNodeNameXMLError**(*cause*, *message*)
Bases: networkapiclient.xml_utils.XMLErrorUtils

Nome inválido para representá-lo como uma TAG de XML.

exception networkapiclient.xml_utils.**InvalidNodeTypeXMLError**(*cause*, *message*)
Bases: networkapiclient.xml_utils.XMLErrorUtils

Tipo inválido para o conteúdo de uma TAG de XML.

exception networkapiclient.xml_utils.**XMLErrorUtils**(*cause*, *message*)
Bases: exceptions.Exception

Representa um erro ocorrido durante o marshall ou unmarshall do XML.

networkapiclient.xml_utils.**dumps**(*map*, *root_name*, *root_attributes=None*)
Cria um string no formato XML a partir dos elementos do map.

Os elementos do mapa serão nós filhos do root_name.

Cada chave do map será um Nó no XML. E o valor da chave será o conteúdo do Nó.

Exemplos:

```
- Mapa: {'networkapi':1}
  XML: &lt;?xml version="1.0" encoding="UTF-8"?&gt;&lt;networkapi&gt;1&lt;/networkapi&gt;

- Mapa: {'networkapi':{'teste':1}}
  XML:  &lt;?xml version="1.0" encoding="UTF-8"?&gt;
  &lt;networkapi&gt;
  &lt;teste&gt;1&lt;/teste&gt;
  &lt;/networkapi&gt;

- Mapa: {'networkapi':{'teste01':01, 'teste02':02}}
  XML: &lt;?xml version="1.0" encoding="UTF-8"?&gt;
  &lt;networkapi&gt;
  &lt;teste01&gt;01&lt;/teste01&gt;
  &lt;teste02&gt;02&lt;/teste02&gt;
  &lt;/networkapi&gt;

- Mapa: {'networkapi':{'teste01':01, 'teste02':[02,03,04]}}
  XML: &lt;?xml version="1.0" encoding="UTF-8"?&gt;
  &lt;networkapi&gt;
  &lt;teste01&gt;01&lt;/teste01&gt;
  &lt;teste02&gt;02&lt;/teste02&gt;
  &lt;teste02&gt;03&lt;/teste02&gt;
  &lt;teste02&gt;04&lt;/teste02&gt;
  &lt;/networkapi&gt;

- Mapa: {'networkapi':{'teste01':01, 'teste02':{'a':1, 'b':2}}}
  XML: &lt;?xml version="1.0" encoding="UTF-8"?&gt;
  &lt;networkapi&gt;
  &lt;teste01&gt;01&lt;/teste01&gt;
  &lt;teste02&gt;
  &lt;a&gt;1&lt;/a&gt;
  &lt;b&gt;2&lt;/b&gt;
  &lt;/teste02&gt;
  &lt;/networkapi&gt;
```

Parameters

- **map** – Dicionário com os dados para serem convertidos em XML.
- **root_name** – Nome do nó root do XML.
- **root_attributes** – Dicionário com valores para serem adicionados como atributos para o nó root.

Returns XML

Raises

- **XMLErrorUtils** – Representa um erro ocorrido durante o marshall ou unmarshall do XML.
- **InvalidNodeNameXMLError** – Nome inválido para representá-lo como uma TAG de XML.
- **InvalidNodeTypeXMLError** – "Tipo inválido para o conteúdo de uma TAG de XML.

networkapiclient.xml_utils.**dumps_networkapi**(*map*, *version='1.0'*)

Idem ao método dump, porém, define que o nó root é o valor 'networkapi'.

Parameters

- **map** – Dicionário com os dados para serem convertidos em XML.
- **version** – Versão do nó networkapi. A versão será adicionada como atributo do nó.

Returns XML

Raises

- **XMLErrorUtils** – Representa um erro ocorrido durante o marshall ou unmarshall do XML.
- **InvalidNodeNameXMLError** – Nome inválido para representá-lo como uma TAG de XML.
- **InvalidNodeTypeXMLError** – "Tipo inválido para o conteúdo de uma TAG de XML.

networkapiclient.xml_utils.**loads**(*xml*, *force_list=None*)

Cria um dicionário com os dados do XML.

O dicionário terá como chave o nome do nó root e como valor o conteúdo do nó root. Quando o conteúdo de um nó é uma lista de nós então o valor do nó será um dicionário com uma chave para cada nó. Entretanto, se existir nós, de um mesmo pai, com o mesmo nome, então eles serão armazenados em uma mesma chave do dicionário que terá como valor uma lista.

O force_list deverá ter nomes de nós do XML que necessariamente terão seus valores armazenados em uma lista no dicionário de retorno.

:: Por exemplo: xml_1 = <?xml version="1.0" encoding="UTF-8"?> <networkapi versao="1.0"> <testes> <teste>1<teste> <teste>2<teste> </testes> </networkapi>

A chamada loads(xml_1), irá gerar o dicionário: {'networkapi':{'testes':{'teste':[1,2]}}}

xml_2 = <?xml version="1.0" encoding="UTF-8"?> <networkapi versao="1.0"> <testes> <teste>1<teste> </testes> </networkapi>

A chamada loads(xml_2), irá gerar o dicionário: {'networkapi':{'testes':{'teste':1}}}

A chamada loads(xml_2, ['teste']), irá gerar o dicionário: {'networkapi':{'testes':{'teste':[1]}}}

Ou seja, o XML_2 tem apenas um nó 'teste', porém, ao informar o parâmetro 'force_list' com o valor ['teste'], a chave 'teste', no dicionário, terá o valor dentro de uma lista.

Parameters

- **xml** – XML
- **force_list** – Lista com os nomes dos nós do XML que deverão ter seus valores armazenados em lista dentro da chave do dicionário de retorno.

Returns Dicionário com os nós do XML.

Raises XMLErrorUtils Representa um erro ocorrido durante o marshall ou unmarshall do XML.

networkapiclient.xml_utils.**remove_illegal_characters**(*xml*)

Module contents

CHAPTER 2

Get Client Factory Instance

For use V3 or V4 functions of GloboNetworkAPI client, you first need to instantiate the Client Factory.

First import ClientFactory class doing:

```
from networkapiclient.ClientFactory import ClientFactory
```

After it instantiate ClientFactory passing to it 5 (five) parameters in this order:

- **networkapi_url**: string representing URL that points to where GloboNetworkAPI service is running.
- **user**: string representing the name of the user that wants to access some GloboNetworkAPI functionality. Example: "networkapi_test".
- **password**: string representing the password of the user above specified that wants to access some GloboNetworkAPI functionality. Example: "networkapi_pwd".
- **user_ldap**: string representing the LDAP user of the registered user at GloboNetworkAPI Service. It's optional.
- **log_level**: string representing how client will manage LOG's. You can pass "ERROR", "WARN", "INFO", "DEBUG" or "TRACE". It's optional, if you don't pass anything the standard value will be "INFO".

Example:

```
client = ClientFactory("http://localhost:8000/", "networkapi_user", "networkapi_pwd", "networkapi_use
```

CHAPTER 3

Using GloboNetworkAPI Client V3

Using Environment module

Assuming that you have Client Factory instantiated (see *How to Instantiate Client Factory*) at **client** variable, in order to access methods relative to Environment module you need to call create_api_environment() at **client**.

Example:

```
env_module = client.create_api_environment()
```

For more information, please look GloboNetworkAPI documentation.

GET

The List of fields available at Environment module is:

- id
- name
- grupo_l3
- ambiente_logico
- divisao_dc
- filter
- acl_path
- ipv4_template
- ipv6_template
- link
- min_num_vlan_1
- max_num_vlan_1
- min_num_vlan_2
- max_num_vlan_2
- vrf
- default_vrf

- father_environment
- children
- configs
- routers
- equipments
- sdn_controllers

Obtain List of Environments through id's

Here you need to call get() method at env_module.

You can pass up to 5 parameters:

- **ids**: List containing identifiers of environments.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Environments.

Examples:

```
envs = env_module.get(ids=[1, 2, 3])

envs = env_module.get(ids=[1, 2, 3],
                      include=['name', 'divisao_dc'],
                      exclude=['id'],
                      kind='details')

envs = env_module.get(ids=[1, 2, 3],
                      fields=['id', 'name', 'grupo_l3'])
```

Obtain List of Environments through extended search

Here you need to call search() method at env_module.

You can pass up to 5 parameters:

- **search**: Dict containing QuerySets to find environments.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Environments.

Example:

```
search = {
    'extends_search': [{
        'divisao_dc': 1,
        'ambiente_logico__nome': 'AmbLog'
    }],
    'start_record': 0,
    'custom_search': '',
    'end_record': 25,
    'asorting_cols': [],
    'searchable_columns': []}
fields = ['id', 'name']

envs = env_module.search(search=search, fields=fields)
```

POST

The List of fields available for create an Environment is:

- grupo_l3 - **Mandatory**
- ambiente_logico - **Mandatory**
- divisao_dc - **Mandatory**
- filter
- min_num_vlan_1
- max_num_vlan_1
- min_num_vlan_2
- max_num_vlan_2
- ipv4_template
- ipv6_template
- link
- acl_path
- vrf
- father_environment
- default_vrf - **Mandatory**
- **configs**
    - subnet
    - new_prefix
    - type
    - network_type

Create List of Environments

Here you need to call create() method at env_module.

You need to pass 1 parameter:

- **environments**: List containing environments that you want to create.

Example:

```
envs_to_create = [
    {
        "grupo_l3": 1,
        "ambiente_logico": 2,
        "divisao_dc": 3,
        "filter": 1,
        "min_num_vlan_1": 1,
        "max_num_vlan_1": 500,
        "min_num_vlan_2": 1,
        "max_num_vlan_2": 500,
        "ipv4_template": "templatev4",
        "ipv6_template": "templatev6",
        "link": "http://environment",
        "acl_path": "path_to_acl",
        "vrf": "Test-Vrf",
        "father_environment": 1,
        "default_vrf": 1
    },
    {
        "grupo_l3": 1,
        "ambiente_logico": 2,
        "divisao_dc": 4,
        "default_vrf": 1,
        "configs": [
            {
                'subnet': 'febe:bebe:bebe:8200:0:0:0:0/57',
                'new_prefix': '64',
                'type': 'v6',
                'network_type': 8
            },
            {
                'subnet': '10.10.0.0/16',
                'new_prefix': '24',
                'type': 'v4',
                'network_type': 8
            }
        ]
    }
]

env_module.create(environments=envs_to_create)
```

PUT

The List of fields available for update an Environment is:

- id - **Mandatory**
- grupo_l3 - **Mandatory**
- ambiente_logico - **Mandatory**
- divisao_dc - **Mandatory**
- filter

- min_num_vlan_1
- max_num_vlan_1
- min_num_vlan_2
- max_num_vlan_2
- ipv4_template
- ipv6_template
- link
- acl_path
- vrf
- father_environment
- default_vrf - **Mandatory**
- **configs**
  - id
  - grupo_l3
  - ambiente_logico
  - divisao_dc
  - default_vrf

Update List of Environments

Here you need to call update() method at env_module.

You need to pass 1 parameter:

- **environments**: List containing environments that you want to update.

Example:

```
envs_to_update = [
    {
        "id": 1,
        "grupo_l3": 1,
        "ambiente_logico": 2,
        "default_vrf": 1,
        "divisao_dc": 3
    },
    {
        "id": 2,
        "grupo_l3": 1,
        "ambiente_logico": 2,
        "divisao_dc": 4,
        "default_vrf": 1,
        "configs": [
            {
                'id': 1,
                'subnet': 'febe:bebe:bebe:8200:0:0:0:0/57',
                'new_prefix': '64',
                'type': 'v6',
                'network_type': 8
```

```
            },
            {
                'subnet': '10.10.0.0/16',
                'new_prefix': '24',
                'type': 'v4',
                'network_type': 8
            }
        ]
    }
]

env_module.update(environments=envs_to_update)
```

DELETE

Delete List of Environments

Here you need to call delete() method at env_module.

You need to pass 1 parameter:

- **ids**: List containing identifiers of environments that you want to delete.

Example:

```
env_module.delete(ids=[1, 2, 3])
```

Using Environment Vip module

Assuming that you have Client Factory instantiated (see *How to Instantiate Client Factory*) at **client** variable, in order to access methods relative to Environment Vip module you need to call create_api_environment_vip() at **client**.

Example:

```
envvip_module = client.create_api_environment_vip()
```

For more information, please look GloboNetworkAPI documentation.

GET

The List of fields available at Environment Vip module is:

- id
- finalidade_txt
- cliente_txt
- ambiente_p44_txt
- description
- name
- conf
- optionsvip

- environments

Obtain List of Environment Vip through id's

Here you need to call get() method at envvip_module.

You can pass up to 5 parameters:

- **ids**: List containing identifiers of environment vips.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Environment Vips.

Examples:

```
env_vips = envvip_module.get(ids=[1, 2, 3])

env_vips = envvip_module.get(ids=[1, 2, 3],
                             include=['name', 'finalidade_txt'],
                             exclude=['id'],
                             kind='details')

env_vips = envvip_module.get(ids=[1, 2, 3],
                             fields=['id', 'ambiente_p44_txt', 'conf'])
```

Obtain List of Environment Vips through extended search

Here you need to call search() method at envvip_module.

You can pass up to 5 parameters:

- **search**: Dict containing QuerySets to find environment vips.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Environment Vips.

Example:

```
search = {
    'extends_search': [{
        'description__icontains': 'EnvVip-Test',
    }],
    'start_record': 0,
    'custom_search': '',
    'end_record': 25,
    'asorting_cols': [],
    'searchable_columns': []}
fields = ['id', 'name']
```

```
env_vips = envvip_module.search(search=search, fields=fields)
```

POST

The List of fields available for create an Environment Vip is:

- finalidade_txt - **Mandatory**
- cliente_txt - **Mandatory**
- ambiente_p44_txt - **Mandatory**
- description - **Mandatory**
- conf
- **optionsvip**
  - option
- **environments**
  - environment

Create List of Environment Vips

Here you need to call create() method at envvip_module.

You need to pass 1 parameter:

- **environments**: List containing environment vips that you want to create.

Example:

```
envvips_to_create = [
    {
        "finalidade_txt": "FIN-TEST-1",
        "cliente_txt": "CLIENT-TEST-1",
        "ambiente_p44_txt": "AMBP44-TEST-1",
        "conf": "anyconf"
    },
    {
        "finalidade_txt": "FIN-TEST-2",
        "cliente_txt": "CLIENT-TEST-2",
        "ambiente_p44_txt": "AMBP44-TEST-2",
        "optionsvip": [
            {
                "option": 1
            },
            {
                "option": 2
            }
        ],
        "environments": [
            {
                "environment": 1
            },
            {
                "environment": 2
            }
```

```
        ]
    }
]

envvip_module.create(environments=envvips_to_create)
```

PUT

The List of fields available for update an Environment Vip is:

- id - **Mandatory**
- finalidade_txt - **Mandatory**
- cliente_txt - **Mandatory**
- ambiente_p44_txt - **Mandatory**
- description - **Mandatory**
- conf
- **optionsvip**
  - option
- **environments**
  - environment

Update List of Environment Vips

Here you need to call update() method at envvip_module.

You need to pass 1 parameter:

- **environments**: List containing environment vips that you want to update.

Example:

```
envvips_to_update = [
    {
        "id": 1,
        "finalidade_txt": "FIN-TEST-1-NEW",
        "cliente_txt": "CLIENT-TEST-1-NEW",
        "ambiente_p44_txt": "AMBP44-TEST-1-NEW"
    },
    {
        "id": 2,
        "finalidade_txt": "FIN-TEST-2-NEW",
        "cliente_txt": "CLIENT-TEST-2-NEW",
        "ambiente_p44_txt": "AMBP44-TEST-2-NEW",
        "optionsvip": [
            {
                "option": 3
            },
            {
                "option": 4
            }
        ],
```

```
        "environments": [
            {
                "environment": 3
            },
            {
                "environment": 5
            }
        ]
    }
]

envvip_module.update(environments=envvips_to_update)
```

DELETE

Delete List of Environment Vips

Here you need to call delete() method at envvip_module.

You need to pass 1 parameter:

- **ids**: List containing identifiers of environment vips that you want to delete.

Example:

```
envvip_module.delete(ids=[1, 2, 3])
```

Using Equipment module

Assuming that you have Client Factory instantiated (see *How to Instantiate Client Factory*) at **client** variable, in order to access methods relative to Equipment module you need to call create_api_equipment() at **client**.

Example:

```
eqpt_module = client.create_api_equipment()
```

For more information, please look GloboNetworkAPI documentation.

GET

The List of fields available at Equipment module is:

- id
- name
- maintenance
- equipment_type
- model
- ipv4
- ipv6
- environments

- groups

Obtain List of Equipments through id's

Here you need to call get() method at eqpt_module.

You can pass up to 5 parameters:

- **ids**: List containing identifiers of equipments.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Equipments.

Examples:

```
epqts = eqpt_module.get(ids=[1, 2, 3])

epqts = eqpt_module.get(ids=[1, 2, 3],
                        include=['name', 'maintenance'],
                        exclude=['id'],
                        kind='basic')

epqts = eqpt_module.get(ids=[1, 2, 3],
                        fields=['id', 'name', 'model'])
```

Obtain List of Equipments through extended search

Here you need to call search() method at eqpt_module.

You can pass up to 5 parameters:

- **search**: Dict containing QuerySets to find equipments.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Equipments.

Example:

```
search = {
    'extends_search': [{
        'maintenance': false,
        'tipo_equipamento': 1
    }],
    'start_record': 0,
    'custom_search': '',
    'end_record': 25,
    'asorting_cols': [],
    'searchable_columns': []}
```

```
fields = ['id', 'name', 'model']

epqts = eqpt_module.search(search=search, fields=fields)
```

POST

The List of fields available for create an Equipment is:

- **environments**
  - id
  - is_router
  - is_controller
- equipment_type - **Mandatory**
- **groups**
  - id
- **ipv4**
  - id
- **ipv6**
  - id
- maintenance - **Mandatory**
- model - **Mandatory**
- name - **Mandatory**

Create List of Equipments

Here you need to call create() method at eqpt_module.

You need to pass 1 parameter:

- **equipments**: List containing equipments that you want to create.

Example:

```
eqpts_to_create = [
    {
        "name": "Eqpt-1",
        "maintenance": False,
        "equipment_type": 8,
        "model": 3,
        "environments": [
            {
                "id": 1,
                "is_router": True,
                "is_controller": False
            },
            {
                "id": 2,
                "is_router": False,
                "is_controller": False
```

```
            }
        ],
        "ipv4": [1, 2]
    },
    {
        "name": "Eqpt-2",
        "maintenance": False,
        "equipment_type": 9,
        "model": 3,
        "ipv6": [1, 2],
        "groups": [
            {
                "id": 1
            },
            {
                "id": 2
            }
        ]
    }
]

eqpt_module.create(equipments=eqpts_to_create)
```

PUT

The List of fields available for update an Equipment is:

- id - **Mandatory**
- **environments**
    - id
    - is_router
    - is_controller
- equipment_type - **Mandatory**
- **groups**
    - id
- **ipv4**
    - id
- **ipv6**
    - id
- maintenance - **Mandatory**
- model - **Mandatory**
- name - **Mandatory**

Update List of Equipments

Here you need to call update() method at eqpt_module.

You need to pass 1 parameter:

- **equipments**: List containing equipments that you want to update.

Example:

```
eqpts_to_update = [
    {
        "id": 1,
        "name": "Eqpt-1-Updated",
        "maintenance": False,
        "equipment_type": 2,
        "model": 2,
        "environments": [
            {
                "id": 2,
                "is_router": True,
                "is_controller": False
            }
        ],
        "ipv4": [3, 5, 7]

    },
    {
        "id": 2,
        "name": "Eqpt-2-Updated",
        "maintenance": False,
        "equipment_type": 7,
        "model": 2,
        "ipv6": [1, 2],
        "groups": [
            {
                "id": 2
            },
            {
                "id": 3
            }
        ]
    }
]

eqpt_module.update(equipments=eqpts_to_update)
```

DELETE

Delete List of Equipments

Here you need to call delete() method at eqpt_module.

You need to pass 1 parameter:

- **ids**: List containing identifiers of equipments that you want to delete.

Example:

```
eqpt_module.delete(ids=[1, 2, 3])
```

Using Server Pool module

Assuming that you have Client Factory instantiated (see *How to Instantiate Client Factory*) at **client** variable, in order to access methods relative to Server Pool module you need to call create_api_pool() at **client**.

Example:

```
pool_module = client.create_api_pool()
```

For more information, please look GloboNetworkAPI documentation.

GET

The List of fields available at Server Pool module is:

- id
- identifier
- default_port
- environment
- servicedownaction
- lb_method
- healthcheck
- default_limit
- server_pool_members
- pool_created
- vips
- dscp
- groups_permissions

Obtain List of Server Pools through id's

Here you need to call get() method at pool_module.

You can pass up to 5 parameters:

- **ids**: List containing identifiers of server pools.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Server Pools.

Examples:

```
pools = pool_module.get(ids=[1, 2, 3])
```

```
pools = pool_module.get(ids=[1, 2, 3],
                        include=['identifier', 'healthcheck'],
                        exclude=['environment'],
                        kind='details')

pools = pool_module.get(ids=[1, 2, 3],
                        fields=['id', 'identifier', 'default_port'])
```

Obtain List of Server Pools through extended search

Here you need to call search() method at pool_module.

You can pass up to 5 parameters:

- **search**: Dict containing QuerySets to find server pools.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Server Pools.

Example:

```
search = {
    'extends_search': [{
        "environment": 1
    }],
    'start_record': 0,
    'custom_search': '',
    'end_record': 25,
    'asorting_cols': [],
    'searchable_columns': []}
fields = ['id', 'identifier']

pools = pool_module.search(search=search, fields=fields)
```

POST

The List of fields available for create a Server Pool is:

- identifier - **Mandatory**
- default_port - **Mandatory**
- environment - **Mandatory**
- **servicedownaction - Mandatory**
  - id
  - name - **Mandatory**
- lb_method - **Mandatory**
- **healthcheck - Mandatory**
  - identifier - **Mandatory**

        - healthcheck_type - **Mandatory**
        - healthcheck_request - **Mandatory**
        - healthcheck_expect - **Mandatory**
        - destination - **Mandatory**
- default_limit - **Mandatory**
- **groups_permissions**
        - user_group - **Mandatory**
        - read - **Mandatory**
        - write - **Mandatory**
        - delete - **Mandatory**
        - change_config - **Mandatory**
- **users_permissions**
        - user - **Mandatory**
        - read - **Mandatory**
        - write - **Mandatory**
        - delete - **Mandatory**
        - change_config - **Mandatory**
- **server_pool_members - Mandatory**
        - **ipv6 - Mandatory**
            * id - **Mandatory**
            * ip_formated - **Mandatory**
        - **ip - Mandatory**
            * id - **Mandatory**
            * ip_formated - **Mandatory**
        - priority - **Mandatory**
        - weight - **Mandatory**
        - limit - **Mandatory**
        - port_real - **Mandatory**
        - member_status - **Mandatory**

Create List of Server Pools

Here you need to call create() method at pool_module.

You need to pass 1 parameter:

- **pools**: List containing server pools that you want to create.

Example:

```
pools_to_create = [
    {
        "lb_method": "least-conn",
        "server_pool_members": [

        ],
        "healthcheck": {
            "healthcheck_type": "TCP",
            "destination": "*:*",
            "healthcheck_expect": "",
            "identifier": "Test_1234",
            "healthcheck_request": ""
        },
        "environment": 4449,
        "servicedownaction": {
            "id": 1,
            "name": "something"
        },
        "default_port": 9943,
        "default_limit": 0,
        "identifier": "PoolTest"
    },
    {
        "lb_method": "least-conn",
        "server_pool_members":[
            {
                "port_real": 5564,
                "weight": 0,
                "ip": {
                    "ip_formated": "10.134.9.201",
                    "id": 4
                },
                "priority": 0,
                "limit": 0,
                "member_status": 7,
                "ipv6": {
                    "ip_formated": "fdbe:fdbe:0000:0000:0000:0000:0000:0002",
                    "id": 3
                }
            },
            {
                "port_real": 3456,
                "weight": 0,
                "ip": {
                    "ip_formated": "10.134.9.202",
                    "id": 5
                },
                "priority": 0,
                "limit": 0,
                "member_status": 7,
                "ipv6": {
                    "ip_formated": "fdbe:fdbe:0000:0000:0000:0000:0000:0002",
                    "id": 3
                }
            }
        ],
        "healthcheck":{
            "healthcheck_type": "HTTP",
```

```
            "destination": "*:14500",
            "healthcheck_expect": "",
            "identifier": "Test_8787",
            "healthcheck_request": ""
        },
        "environment": 543,
        "servicedownaction":{
            "id": 1,
            "name": "something"
        },
        "default_port": 12201,
        "default_limit": 0,
        "identifier": "PoolTest-2",
    }
]

pool_module.create(pools=pools_to_create)
```

PUT

The List of fields available for update a Server Pool is:

- id - **Mandatory**
- identifier - **Mandatory**
- default_port - **Mandatory**
- environment - **Mandatory**
- **servicedownaction - Mandatory**
    - id
    - name - **Mandatory**
- lb_method - **Mandatory**
- **healthcheck - Mandatory**
    - identifier - **Mandatory**
    - healthcheck_type - **Mandatory**
    - healthcheck_request - **Mandatory**
    - healthcheck_expect - **Mandatory**
    - destination - **Mandatory**
- default_limit - **Mandatory**
- **groups_permissions**
    - user_group - **Mandatory**
    - read - **Mandatory**
    - write - **Mandatory**
    - delete - **Mandatory**
    - change_config - **Mandatory**
- **users_permissions**

  - user - **Mandatory**
  - read - **Mandatory**
  - write - **Mandatory**
  - delete - **Mandatory**
  - change_config - **Mandatory**
- **server_pool_members - Mandatory**
  - **ipv6 - Mandatory**
    - id - **Mandatory**
    - ip_formated - **Mandatory**
  - **ip - Mandatory**
    - id - **Mandatory**
    - ip_formated - **Mandatory**
  - priority - **Mandatory**
  - weight - **Mandatory**
  - limit - **Mandatory**
  - port_real - **Mandatory**
  - member_status - **Mandatory**

Update List of Server Pools

Here you need to call update() method at pool_module.

You need to pass 1 parameter:

- **pools**: List containing server pools that you want to update.

Example:

```
pools_to_update = [
    {
        "id": 1,
        "lb_method": "least-conn",
        "server_pool_members": [

        ],
        "healthcheck": {
            "healthcheck_type": "TCP",
            "destination": "*:*",
            "healthcheck_expect": "",
            "identifier": "Test_12334",
            "healthcheck_request": ""
        },
        "environment": 449,
        "servicedownaction": {
            "id": 1,
            "name": "something"
        },
        "default_port": 993,
        "default_limit": 0,
```

```
        "identifier": "PoolTest-New",
        "users_permissions": [
            {
                "user": 1,
                "read": True,
                "write": False,
                "delete": False,
                "change_config": False
            }
        ],
        "groups_permissions": [
            {
                "user_group": 2,
                "read": True,
                "write": False,
                "delete": False,
                "change_config": False
            }
        ]
    },
    {
        "id": 2,
        "lb_method": "least-conn",
        "server_pool_members":[
            {
                "port_real": 554,
                "weight": 2,
                "ip": {
                    "ip_formated": "10.134.9.203",
                    "id": 6
                },
                "priority": 2,
                "limit": 0,
                "member_status": 7,
                "ipv6": {
                    "ip_formated": "fdbe:fdbe:0000:0000:0000:0000:0000:0002",
                    "id": 3
                }
            },
            {
                "port_real": 346,
                "weight": 1,
                "ip": {
                    "ip_formated": "10.134.9.205",
                    "id": 7
                },
                "priority": 2,
                "limit": 0,
                "member_status": 7,
                "ipv6": {
                    "ip_formated": "fdbe:fdbe:0000:0000:0000:0000:0000:0002",
                    "id": 3
                }
            }
        ],
        "healthcheck":{
            "healthcheck_type": "HTTP",
            "destination": "*:14500",
```

```
            "healthcheck_expect": "",
            "identifier": "Test_8787",
            "healthcheck_request": ""
        },
        "environment": 543,
        "servicedownaction":{
            "id": 1,
            "name": "something"
        },
        "default_port": 12201,
        "default_limit": 0,
        "identifier": "PoolTest-New-2",
    }
]

pool_module.update(pools=pools_to_update)
```

DELETE

Delete List of Server Pools

Here you need to call delete() method at pool_module.

You need to pass 1 parameter:

- **ids**: List containing identifiers of server pools that you want to delete.

Example:

```
pool_module.delete(ids=[1, 2, 3])
```

Using Vip Request module

Assuming that you have Client Factory instantiated (see *How to Instantiate Client Factory*) at **client** variable, in order to access methods relative to Vip Request module you need to call create_api_vip_request() at **client**.

Example:

```
vip_module = client.create_api_vip_request()
```

For more information, please look GloboNetworkAPI documentation.

GET

The List of fields available at Vip Request module is:

- id
- name
- service
- business
- environmentvip
- ipv4

- ipv6
- equipments
- default_names
- dscp
- ports
- options
- groups_permissions
- created

Obtain List of Vip Requests through id's

Here you need to call get() method at vip_module.

You can pass up to 5 parameters:

- **ids**: List containing identifiers of vip requests.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Vip Requests.

Examples:

```
vips = vip_module.get(ids=[1, 2, 3])

vips = vip_module.get(ids=[1, 2, 3],
                      include=['name', 'service'],
                      exclude=['id'],
                      kind='details')

vips = vip_module.get(ids=[1, 2, 3],
                      fields=['id', 'name', 'ipv4'])
```

Obtain List of Vip Requests through extended search

Here you need to call search() method at vip_module.

You can pass up to 5 parameters:

- **search**: Dict containing QuerySets to find vip requests.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Vip Requests.

Example:

```
search = {
    'extends_search': [
        {
            "ipv4__oct1": "192",
            "ipv4__oct2": "168",
            "created": true
        },
        {
            "ipv4__oct2": "168",
            "ipv4__oct3": "17",
            "created": false
        }
    ],
    'start_record': 0,
    'custom_search': '',
    'end_record': 25,
    'asorting_cols': [],
    'searchable_columns': []
}
fields = ['id', 'name']

vips = vip_module.search(search=search, fields=fields)
```

POST

The List of fields available for create an Vip Request is:

- business
- created
- environmentvip
- id
- ipv4
- ipv6
- name
- **options**
    - cache_group
    - persistence
    - timeout
    - traffic_return
- **ports**
    - id
    - **options**
        - l4_protocol
        - l7_protocol

        - **pools**
            - l7_rule - **Mandatory**
            - l7_value
            - order
            - server_pool - **Mandatory**
        - port
- service
- **groups_permissions**
        - user_group - **Mandatory**
        - read - **Mandatory**
        - write - **Mandatory**
        - delete - **Mandatory**
        - change_config - **Mandatory**
- **users_permissions**
        - user - **Mandatory**
        - read - **Mandatory**
        - write - **Mandatory**
        - delete - **Mandatory**
        - change_config - **Mandatory**

Create List of Vip Requests

Here you need to call create() method at vip_module.

You need to pass 1 parameter:

- **vips**: List containing vip requests that you want to create.

Example:

```
vips_to_create = [
    {
        "business": "some-business",
        "environmentvip": 2,
        "ipv4": 1,
        "ipv6": 44,
        "name": "vip.test.com",
        "options": {
            "cache_group": 2,
            "persistence": 3,
            "timeout": 41,
            "traffic_return": 20
        },
        "ports": [
            {
                "options": {
                    "l4_protocol": 32,
```

```
                    "l7_protocol": 31
                },
                "pools": [
                    {
                        "l7_rule": 34,
                        "order": 1,
                        "server_pool": 3
                    }
                ],
                "port": 8181
            },
            {
                "options": {
                    "l4_protocol": 33,
                    "l7_protocol": 34
                },
                "pools": [
                    {
                        "l7_rule": 37,
                        "order": 0,
                        "server_pool": 4
                    }
                ],
                "port": 9090
            }
        ],
        "service": "some-service"
    }
]

vip_module.create(vips=vips_to_create)
```

PUT

The List of fields available for update an Vip Request is:

- id - **Mandatory**
- business
- created
- environmentvip
- id
- ipv4
- ipv6
- name
- **options**
    - cache_group
    - persistence
    - timeout
    - traffic_return

- **ports**
  - id
  - **options**
    - l4_protocol
    - l7_protocol
  - **pools**
    - l7_rule - **Mandatory**
    - l7_value
    - order
    - server_pool - **Mandatory**
  - port
- service
- **groups_permissions**
  - user_group - **Mandatory**
  - read - **Mandatory**
  - write - **Mandatory**
  - delete - **Mandatory**
  - change_config - **Mandatory**
- **users_permissions**
  - user - **Mandatory**
  - read - **Mandatory**
  - write - **Mandatory**
  - delete - **Mandatory**
  - change_config - **Mandatory**

Update List of Vip Requests

Here you need to call update() method at vip_module.

You need to pass 1 parameter:

- **vips**: List containing vip requests that you want to update.

Example:

```
vips_to_update = [
    {
        "id": 1,
        "business": "some-business-2",
        "environmentvip": 3,
        "ipv4": 2,
        "ipv6": 43,
        "name": "vipnew.test.com",
        "options": {
```

```
            "cache_group": 1,
            "persistence": 3,
            "timeout": 40,
            "traffic_return": 9
        },
        "ports": [
            {
                "options": {
                    "l4_protocol": 2,
                    "l7_protocol": 1
                },
                "pools": [
                    {
                        "l7_rule": 24,
                        "order": 1,
                        "server_pool": 3
                    }
                ],
                "port": 8181
            },
            {
                "options": {
                    "l4_protocol": 3,
                    "l7_protocol": 4
                },
                "pools": [
                    {
                        "l7_rule": 27,
                        "order": 0,
                        "server_pool": 4
                    }
                ],
                "port": 9191
            }
        ],
        "service": "some-new-service"
    }
]

vip_module.update(vips=vips_to_update)
```

DELETE

Delete List of Vip Requests

Here you need to call delete() method at vip_module.

You need to pass 1 parameter:

- **ids**: List containing identifiers of vip requests that you want to delete.

Example:

```
vip_module.delete(ids=[1, 2, 3])
```

Using Vlan module

Assuming that you have Client Factory instantiated (see *How to Instantiate Client Factory*) at **client** variable, in order to access methods relative to Vlan module you need to call create_api_vlan() at **client**.

Example:

```
vlan_module = client.create_api_vlan()
```

For more information, please look GloboNetworkAPI documentation.

GET

The List of fields available at Vlan module is:

- id
- name
- num_vlan
- environment
- description
- acl_file_name
- acl_valida
- acl_file_name_v6
- acl_valida_v6
- active
- vrf
- acl_draft
- acl_draft_v6
- networks_ipv4
- networks_ipv6
- vrfs
- groups_permissions

Obtain List of Vlans through id's

Here you need to call get() method at vlan_module.

You can pass up to 5 parameters:

- **ids**: List containing identifiers of vlans.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Vlans.

Examples:

```
vlans = vlan_module.get(ids=[1, 2, 3])

vlans = vlan_module.get(ids=[1, 2, 3],
                        include=['name', 'vrf'],
                        exclude=['environment'],
                        kind='basic')

vlans = vlan_module.get(ids=[1, 2, 3],
                        fields=['id', 'name', 'vrf'])
```

Obtain List of Vlans through extended search

Here you need to call search() method at vlan_module.

You can pass up to 5 parameters:

- **search**: Dict containing QuerySets to find vlans.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Vlans.

Example:

```
search = {'extends_search': [{'num_vlan': 1}],
          'start_record': 0,
          'custom_search': '',
          'end_record': 25,
          'asorting_cols': [],
          'searchable_columns': []}
fields = ['id', 'name']

vlans = vlan_module.search(search=search, fields=fields)
```

POST

The List of fields available for create a Vlan is:

- name - **Mandatory**
- num_vlan
- environment - **Mandatory**
- acl_file_name
- acl_file_name_v6
- acl_valida
- acl_valida_v6

- active
- vrf
- acl_draft
- acl_draft_v6
- **create_networkv4**
  - network_type
  - environmentvip
  - prefix
- **create_networkv6**
  - network_type
  - environmentvip
  - prefix

Create List of Vlans

Here you need to call create() method at vlan_module.

You need to pass 1 parameter:

- **vlans**: List containing vlans that you want to create.

Example:

```
vlans_to_create = [
    {
        "name": "Vlan 1",
        "num_vlan": 3,
        "environment": 5,
        "active": True,
        "create_networkv4": {
            "network_type": 6,
            "environmentvip": 2,
            "prefix": 24
        }
    },
    {
        "name": "Vlan 2",
        "num_vlan": 4,
        "environment": 10,
        "active": True,
        "create_networkv4": {
            "network_type": 6,
            "environmentvip": 3,
            "prefix": 24
        }
    }
]

vlan_module.create(vlans=vlans_to_create)
```

PUT

The List of fields available for update a Vlan is:

- id - **Mandatory**
- name - **Mandatory**
- num_vlan - **Mandatory**
- environment - **Mandatory**
- description - **Mandatory**
- acl_file_name - **Mandatory**
- acl_valida - **Mandatory**
- acl_file_name_v6 - **Mandatory**
- acl_valida_v6 - **Mandatory**
- active - **Mandatory**
- vrf - **Mandatory**
- acl_draft - **Mandatory**
- acl_draft_v6 - **Mandatory**

Update List of Vlans

Here you need to call update() method at vlan_module.

You need to pass 1 parameter:

- **vlans**: List containing vlans that you want to update.

Example:

```
vlans_to_update = [
    {
        "id": 1,
        "name": "Vlan 1 changed",
        "num_vlan": 3,
        "environment": 5,
        "description": "",
        "acl_file_name": "",
        "acl_valida": false ,
        "acl_file_name_v6": "",
        "acl_valida_v6": false,
        "active": false,
        "vrf": 'VrfTest',
        "acl_draft": "",
        "acl_draft_v6": ""
    },
    {
        "id": 2,
        "name": "Vlan changed",
        "num_vlan": 4,
        "environment": 10,
        "description": "",
        "acl_file_name": "",
```

```
        "acl_valida": false ,
        "acl_file_name_v6": "",
        "acl_valida_v6": false,
        "active": false,
        "vrf": 'VrfTest',
        "acl_draft": "",
        "acl_draft_v6": ""
    }
]

vlan_module.update(vlans=vlans_to_update)
```

DELETE

Delete List of Vlans

Here you need to call delete() method at vlan_module.

You need to pass 1 parameter:

- **ids**: List containing identifiers of vlans that you want to delete.

Example:

```
vlan_module.delete(ids=[1, 2, 3])
```

Using NetworkIPv4 module

Assuming that you have Client Factory instantiated (see *How to Instantiate Client Factory*) at **client** variable, in order to access methods relative to NetworkIPv4 module you need to call create_api_network_ipv4() at **client**.

Example:

```
netipv4_module = client.create_api_network_ipv4()
```

For more information, please look GloboNetworkAPI documentation.

GET

The List of fields available at NetworkIPv4 module is:

- id
- oct1
- oct2
- oct3
- oct4
- prefix
- networkv4
- mask_oct1
- mask_oct2

- mask_oct3
- mask_oct4
- mask_formated
- broadcast
- vlan
- network_type
- environmentvip
- active
- dhcprelay
- cluster_unit

Obtain List of NetworkIPv4's through id's

Here you need to call get() method at netipv4_module.

You can pass up to 5 parameters:

- **ids**: List containing identifiers of NetworkIPv4's.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of NetworkIPv4's.

Examples:

```
netipv4s = netipv4_module.get(ids=[1, 2, 3])

netipv4s = netipv4_module.get(ids=[1, 2, 3],
                              include=['oct1', 'oct2'],
                              exclude=['oct3'],
                              kind='details')

netipv4s = netipv4_module.get(ids=[1, 2, 3],
                              fields=['id', 'oct1', 'oct2', 'networkv4'])
```

Obtain List of NetworkIPv4's through extended search

Here you need to call search() method at netipv4_module.

You can pass up to 5 parameters:

- **search**: Dict containing QuerySets to find NetworkIPv4's.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of NetworkIPv4's.

Example:

```
search = {
    'extends_search': [{
        {
            'oct1': 10,
        },
        {
            'oct1': 172,
        }
    }],
    'start_record': 0,
    'custom_search': '',
    'end_record': 25,
    'asorting_cols': [],
    'searchable_columns': []}
fields = ['id', 'oct1', 'oct2', 'vlan']

netipv4s = netipv4_module.search(search=search, fields=fields)
```

POST

The List of fields available for create an NetworkIPv4 is:

- oct1
- oct2
- oct3
- oct4
- prefix
- mask_oct1
- mask_oct2
- mask_oct3
- mask_oct4
- vlan - **Mandatory**
- network_type
- environmentvip
- cluster_unit
- active

Create List of NetworkIPv4's

Here you need to call create() method at netipv4_module.

You need to pass 1 parameter:

- **networkipv4s**: List containing NetworkIPv4's that you want to create.

Example:

```
netipv4s_to_create = [
    {
        "vlan": 1
    },
    {
        "oct1": 10,
        "oct2": 10,
        "oct3": 10,
        "oct4": 0,
        "prefix": 24,
        "mask_oct1": 255,
        "mask_oct2": 255,
        "mask_oct3": 255,
        "mask_oct4": 0,
        "vlan": 2,
        "network_type": 3,
        "environmentvip": 2,
        "cluster_unit": "anything"
    }
]

netipv4_module.create(networkipv4s=netipv4s_to_create)
```

PUT

The List of fields available for update an NetworkIPv4 is:

- id - **Mandatory**
- network_type - **Mandatory**
- environmentvip
- cluster_unit
- active

Update List of NetworkIPv4's

Here you need to call update() method at netipv4_module.

You need to pass 1 parameter:

- **networkipv4s**: List containing ipv4s that you want to update.

Example:

```
netipv4s_to_update = [
    {
        "id": 1,
        "networktype": 5
    },
    {
        "id": 2,
        "active": True,
        "network_type": 4,
        "environmentvip": 5,
```

```
        "cluster_unit": "anything"
    }
]

netipv4_module.update(networkipv4s=netipv4s_to_update)
```

DELETE

Delete List of NetworkIPv4's

Here you need to call delete() method at netipv4_module.

You need to pass 1 parameter:

- **ids**: List containing identifiers of NetworkIPv4's that you want to delete.

Example:

```
netipv4_module.delete(ids=[1, 2, 3])
```

Using NetworkIPv6 module

Assuming that you have Client Factory instantiated (see *How to Instantiate Client Factory*) at **client** variable, in order to access methods relative to NetworkIPv6 module you need to call create_api_network_ipv6() at **client**.

Example:

```
netipv6_module = client.create_api_network_ipv6()
```

For more information, please look GloboNetworkAPI documentation.

GET

The List of fields available at NetworkIPv6 module is:

- id
- block1
- block2
- block3
- block4
- block5
- block6
- block7
- block8
- prefix
- networkv6
- mask1

- mask2
- mask3
- mask4
- mask5
- mask6
- mask7
- mask8
- mask_formated

Obtain List of NetworkIPv6's through id's

Here you need to call get() method at netipv6_module.

You can pass up to 5 parameters:

- **ids**: List containing identifiers of NetworkIPv6's.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of NetworkIPv6's.

Examples:

```
netipv6s = netipv6_module.get(ids=[1, 2, 3])

netipv6s = netipv6_module.get(ids=[1, 2, 3],
                              include=['block1', 'block2'],
                              exclude=['block3'],
                              kind='details')

netipv6s = netipv6_module.get(ids=[1, 2, 3],
                              fields=['id', 'block1', 'block2', 'networkv6'])
```

Obtain List of NetworkIPv6's through extended search

Here you need to call search() method at netipv6_module.

You can pass up to 5 parameters:

- **search**: Dict containing QuerySets to find NetworkIPv6's.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of NetworkIPv6's.

Example:

```
search = {
    'extends_search': [{
        {
            "block1": "fefe"
        },
        {
            "block1": "fdbe"
        }
    }],
    'start_record': 0,
    'custom_search': '',
    'end_record': 25,
    'asorting_cols': [],
    'searchable_columns': []}
fields = ['id', 'block1', 'block2', 'mask_formated']

netipv6s = netipv6_module.search(search=search, fields=fields)
```

POST

The List of fields available for create an NetworkIPv6 is:

- block1
- block2
- block3
- block4
- block5
- block6
- block7
- block8
- prefix
- mask
- mask
- mask
- mask
- mask
- mask
- mask
- mask
- vlan - **Mandatory**
- network_type
- environmentvip

- cluster_unit
- active

Create List of NetworkIPv6's

Here you need to call create() method at netipv6_module.

You need to pass 1 parameter:

- **networkipv6s**: List containing NetworkIPv6's that you want to create.

Example:

```
netipv6s_to_create = [
    {
        "vlan": 1
    },
    {
        "block1": "fdbe",
        "block2": "fdbe",
        "block3": "a0a0",
        "block4": "a0a0",
        "block5": "0000",
        "block6": "0000",
        "block7": "0000",
        "block8": "0000",
        "prefix": 64,
        "mask1": "ffff",
        "mask2": "ffff",
        "mask3": "ffff",
        "mask4": "ffff",
        "mask5": "0000",
        "mask6": "0000",
        "mask7": "0000",
        "mask8": "0000",
        "vlan": 2,
        "network_type": 3,
        "environmentvip": 2,
        "active": False,
        "cluster_unit": "anything"
    }
]

netipv6_module.create(networkipv6s=netipv6s_to_create)
```

PUT

The List of fields available for update an NetworkIPv6 is:

- id - **Mandatory**
- network_type - **Mandatory**
- environmentvip
- cluster_unit
- active

Update List of NetworkIPv6's

Here you need to call update() method at netipv6_module.

You need to pass 1 parameter:

- **networkipv6s**: List containing ipv6s that you want to update.

Example:

```
netipv6s_to_update = [
    {
        "id": 1,
        "networktype": 5
    },
    {
        "id": 2,
        "active": True,
        "network_type": 4,
        "environmentvip": 5,
        "cluster_unit": "anything"
    }
]

netipv6_module.update(networkipv6s=netipv6s_to_update)
```

DELETE

Delete List of NetworkIPv6's

Here you need to call delete() method at netipv6_module.

You need to pass 1 parameter:

- **ids**: List containing identifiers of NetworkIPv6's that you want to delete.

Example:

```
netipv6_module.delete(ids=[1, 2, 3])
```

Using IPv4 module

Assuming that you have Client Factory instantiated (see *How to Instantiate Client Factory*) at **client** variable, in order to access methods relative to IPv4 module you need to call create_api_ipv4() at **client**.

Example:

```
ipv4_module = client.create_api_ipv4()
```

For more information, please look GloboNetworkAPI documentation.

GET

The List of fields available at IPv4 module is:

- id

- ip_formated
- oct1
- oct2
- oct3
- oct4
- networkipv4
- description
- equipments
- vips
- server_pool_members

Obtain List of IPv4's through id's

Here you need to call get() method at ipv4_module.

You can pass up to 5 parameters:

- **ids**: List containing identifiers of IPv4's.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of IPv4's.

Examples:

```
ipv4s = ipv4_module.get(ids=[1, 2, 3])

ipv4s = ipv4_module.get(ids=[1, 2, 3],
                        include=['oct1', 'oct2'],
                        exclude=['oct3'],
                        kind='details')

ipv4s = ipv4_module.get(ids=[1, 2, 3],
                        fields=['id', 'oct1', 'oct2', 'networkipv4'])
```

Obtain List of IPv4's through extended search

Here you need to call search() method at ipv4_module.

You can pass up to 5 parameters:

- **search**: Dict containing QuerySets to find IPv4's.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of IPv4's.

Example:

```
search = {
    'extends_search': [{
        {
            'oct1': 10,
        },
        {
            'oct1': 172,
        }
    }],
    'start_record': 0,
    'custom_search': '',
    'end_record': 25,
    'asorting_cols': [],
    'searchable_columns': []}
fields = ['id', 'oct1', 'oct2', 'vips']

ipv4s = ipv4_module.search(search=search, fields=fields)
```

POST

The List of fields available for create an IPv4 is:

- oct1
- oct2
- oct3
- oct4
- networkipv4 - **Mandatory**
- description
- **equipments**
    - id

Create List of IPv4's

Here you need to call create() method at ipv4_module.

You need to pass 1 parameter:

- **ipv4s**: List containing IPv4's that you want to create.

Example:

```
ipv4s_to_create = [
    {
        "oct1": 10,
        "oct2": 10,
        "oct3": 0,
        "oct4": 2,
        "description": "IP 2",
        "networkipv4": 1
    },
```

```
    {
        "description": "IP 1",
        "networkipv4": 2,
        "equipments": [
            {
                "id": 1
            },
            {
                "id": 2
            }
        ]
    }
]

ipv4_module.create(ipv4s=ipv4s_to_create)
```

PUT

The List of fields available for update an IPv4 is:

- id - **Mandatory**
- description
- **equipments**
    - id

Update List of IPv4's

Here you need to call update() method at ipv4_module.

You need to pass 1 parameter:

- **ipv4s**: List containing ipv4s that you want to update.

Example:

```
ipv4s_to_update = [
    {
        "id": 1,
        "description": "New-Desc-1"
    },
    {
        "id": 2,
        "equipments": [
            {
                "id": 1
            },
            {
                "id": 2
            }
        ]
    }
]

ipv4_module.update(ipv4s=ipv4s_to_update)
```

DELETE

Delete List of IPv4's

Here you need to call delete() method at ipv4_module.

You need to pass 1 parameter:

- **ids**: List containing identifiers of IPv4's that you want to delete.

Example:

```
ipv4_module.delete(ids=[1, 2, 3])
```

Using IPv6 module

Assuming that you have Client Factory instantiated (see *How to Instantiate Client Factory*) at **client** variable, in order to access methods relative to IPv6 module you need to call create_api_ipv6() at **client**.

Example:

```
ipv6_module = client.create_api_ipv6()
```

For more information, please look GloboNetworkAPI documentation.

GET

The List of fields available at IPv6 module is:

- id
- ip_formated
- block1
- block2
- block3
- block4
- block5
- block6
- block7
- block8
- networkipv6
- description
- equipments
- vips
- server_pool_members

Obtain List of IPv6's through id's

Here you need to call get() method at ipv6_module.

You can pass up to 5 parameters:

- **ids**: List containing identifiers of IPv6's.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of IPv6's.

Examples:

```
ipv6s = ipv6_module.get(ids=[1, 2, 3])

ipv6s = ipv6_module.get(ids=[1, 2, 3],
                        include=['block1', 'block2'],
                        exclude=['block3'],
                        kind='details')

ipv6s = ipv6_module.get(ids=[1, 2, 3],
                        fields=['id', 'block1', 'block2', 'networkipv6'])
```

Obtain List of IPv6's through extended search

Here you need to call search() method at ipv6_module.

You can pass up to 5 parameters:

- **search**: Dict containing QuerySets to find IPv6's.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of IPv6's.

Example:

```
search = {
    'extends_search': [{
        {
            "block1": "fefe"
        },
        {
            "block1": "fdfd"
        }
    }],
    'start_record': 0,
    'custom_search': '',
    'end_record': 25,
    'asorting_cols': [],
```

```
    'searchable_columns': []}
fields = ['id', 'block1', 'block2', 'vips']

ipv6s = ipv6_module.search(search=search, fields=fields)
```

POST

The List of fields available for create an IPv6 is:

- block1
- block2
- block3
- block4
- block5
- block6
- block7
- block8
- networkipv6 - **Mandatory**
- description
- **equipments**
    - id

Create List of IPv6's

Here you need to call create() method at ipv6_module.

You need to pass 1 parameter:

- **ipv6s**: List containing IPv6's that you want to create.

Example:

```
ipv6s_to_create = [
    {

        "block1": "fefe"
        "block2": "a0a0"
        "block3": "a0a0"
        "block4": "a0a0"
        "block5": "0000"
        "block6": "0000"
        "block7": "0000"
        "block8": "0002"
        "description": "IP 2",
        "networkipv6": 1
    },
    {
        "description": "IP 1",
        "networkipv6": 2,
        "equipments": [
```

```
            {
                "id": 1
            },
            {
                "id": 2
            }
        ]
    }
]

ipv6_module.create(ipv6s=ipv6s_to_create)
```

PUT

The List of fields available for update an IPv6 is:

- id - **Mandatory**
- description
- **equipments**
    - id

Update List of IPv6's

Here you need to call update() method at ipv6_module.

You need to pass 1 parameter:

- **ipv6s**: List containing ipv6s that you want to update.

Example:

```
ipv6s_to_update = [
    {
        "id": 1,
        "description": "New-Desc-1"
    },
    {
        "id": 2,
        "equipments": [
            {
                "id": 1
            },
            {
                "id": 2
            }
        ]
    }
]

ipv6_module.update(ipv6s=ipv6s_to_update)
```

DELETE

Delete List of IPv6's

Here you need to call delete() method at ipv6_module.

You need to pass 1 parameter:

- **ids**: List containing identifiers of IPv6's that you want to delete.

Example:

```
ipv6_module.delete(ids=[1, 2, 3])
```

Using Object Group Permission module

Assuming that you have Client Factory instantiated (see *How to Instantiate Client Factory*) at **client** variable, in order to access methods relative to Object Group Permission module you need to call create_api_object_group_permission() at **client**.

Example:

```
ogp_module = client.create_api_object_group_permission()
```

For more information, please look GloboNetworkAPI documentation.

GET

The List of fields available at Object Group Permission module is:

- id
- user_group
- object_type
- object_value
- read
- write
- change_config
- delete

Obtain List of Object Group Permissions through id's

Here you need to call get() method at ogp_module.

You can pass up to 5 parameters:

- **ids**: List containing identifiers of Object Group Permissions.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Object Group Permissions.

Examples:

```
ogps = ogp_module.get(ids=[1, 2, 3])

ogps = ogp_module.get(ids=[1, 2, 3],
                      include=['read'],
                      exclude=['write', 'delete'],
                      kind='basic')

ogps = ogp_module.get(ids=[1, 2, 3],
                      fields=['id', 'read', 'delete'])
```

Obtain List of Object Group Permissions through extended search

Here you need to call search() method at ogp_module.

You can pass up to 5 parameters:

- **search**: Dict containing QuerySets to find Object Group Permissions.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Object Group Permissions.

Example:

```
search = {
    'extends_search': [{
        "read": True
    }],
    'start_record': 0,
    'custom_search': '',
    'end_record': 25,
    'asorting_cols': [],
    'searchable_columns': []}
fields = ['id', 'user_group', 'read']

ogps = ogp_module.search(search=search, fields=fields)
```

POST

The List of fields available for create an Object Group Permission is:

- user_group - **Mandatory**
- object_type - **Mandatory**
- object_value - **Mandatory**
- read - **Mandatory**
- write - **Mandatory**
- change_config - **Mandatory**

- delete - **Mandatory**

Create List of Object Group Permissions

Here you need to call create() method at ogp_module.

You need to pass 1 parameter:

- **ogps**: List containing Object Group Permissions that you want to create.

Example:

```
ogps_to_create = [
    {
        "user_group": 1,
        "object_type": 2,
        "object_value": 1,
        "read": True,
        "write": True,
        "change_config": True,
        "delete": False
    },
    {
        "user_group": 1,
        "object_type": 2,
        "object_value": 4,
        "read": True,
        "write": True,
        "change_config": True,
        "delete": False
    }
]

ogp_module.create(ogps=ogps_to_create)
```

PUT

The List of fields available for update an Object Group Permission is:

- id - **Mandatory**
- read
- write
- change_config
- delete

Update List of Object Group Permissions

Here you need to call update() method at ogp_module.

You need to pass 1 parameter:

- **ogps**: List containing Object Group Permissions that you want to update.

Example:

```
ogps_to_update = [
    {
        "id": 1,
        "read": False,
        "write": False,
        "change_config": True,
        "delete": False
    },
    {
        "id": 2,
        "read": False,
        "write": False,
        "change_config": True,
        "delete": False
    }
]

ogp_module.update(ogps=ogps_to_update)
```

DELETE

Delete List of Object Group Permissions

Here you need to call delete() method at ogp_module.

You need to pass 1 parameter:

- **ids**: List containing identifiers of Object Group Permissions that you want to delete.

Example:

```
ogp_module.delete(ids=[1, 2, 3])
```

Using Object Group Permission General module

Assuming that you have Client Factory instantiated (see *How to Instantiate Client Factory*) at **client** variable, in order to access methods relative to Object Group Permission General module you need to call create_api_object_group_permission_general() at **client**.

Example:

```
ogpg_module = client.create_api_object_group_permission_general()
```

For more information, please look GloboNetworkAPI documentation.

GET

The List of fields available at Object Group Permission General module is:

- id
- user_group
- object_type
- read

- write
- change_config
- delete

Obtain List of Object Group Permissions General through id's

Here you need to call get() method at ogpg_module.

You can pass up to 5 parameters:

- **ids**: List containing identifiers of Object Group Permissions General.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Object Group Permissions General.

Examples:

```
ogpgs = ogpg_module.get(ids=[1, 2, 3])

ogpgs = ogpg_module.get(ids=[1, 2, 3],
                        include=['read'],
                        exclude=['write', 'delete'],
                        kind='basic')

ogpgs = ogpg_module.get(ids=[1, 2, 3],
                        fields=['id', 'read', 'delete'])
```

Obtain List of Object Group Permissions General through extended search

Here you need to call search() method at ogpg_module.

You can pass up to 5 parameters:

- **search**: Dict containing QuerySets to find Object Group Permissions General.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Object Group Permissions General.

Example:

```
search = {
    'extends_search': [{
        "read": True
    }],
    'start_record': 0,
    'custom_search': '',
    'end_record': 25,
```

```
    'asorting_cols': [],
    'searchable_columns': []}
fields = ['id', 'user_group', 'read']

ogpgs = ogpg_module.search(search=search, fields=fields)
```

POST

The List of fields available for create an Object Group Permission General is:

- user_group - **Mandatory**
- object_type - **Mandatory**
- read - **Mandatory**
- write - **Mandatory**
- change_config - **Mandatory**
- delete - **Mandatory**

Create List of Object Group Permissions General

Here you need to call create() method at ogpg_module.

You need to pass 1 parameter:

- **ogpgs**: List containing Object Group Permissions General that you want to create.

Example:

```
ogpgs_to_create = [
    {
        "user_group": 1,
        "object_type": 2,
        "read": True,
        "write": True,
        "change_config": True,
        "delete": False
    },
    {
        "user_group": 1,
        "object_type": 2,
        "read": True,
        "write": True,
        "change_config": True,
        "delete": False
    }
]

ogpg_module.create(ogpgs=ogpgs_to_create)
```

PUT

The List of fields available for update an Object Group Permission General is:

- id - **Mandatory**

- read
- write
- change_config
- delete

Update List of Object Group Permissions General

Here you need to call update() method at ogpg_module.

You need to pass 1 parameter:

- **ogpgs**: List containing Object Group Permissions General that you want to update.

Example:

```
ogpgs_to_update = [
    {
        "id": 1,
        "read": False,
        "write": False,
        "change_config": True,
        "delete": False
    },
    {
        "id": 2,
        "read": False,
        "write": False,
        "change_config": True,
        "delete": False
    }
]

ogpg_module.update(ogpgs=ogpgs_to_update)
```

DELETE

Delete List of Object Group Permissions General

Here you need to call delete() method at ogpg_module.

You need to pass 1 parameter:

- **ids**: List containing identifiers of Object Group Permissions General that you want to delete.

Example:

```
ogpg_module.delete(ids=[1, 2, 3])
```

Using Object Type module

Assuming that you have Client Factory instantiated (see *How to Instantiate Client Factory*) at **client** variable, in order to access methods relative to Object Type module you need to call create_api_object_type() at **client**.

Example:

```
ot_module = client.create_api_object_type()
```

For more information, please look GloboNetworkAPI documentation.

GET

The List of fields available at Object Type module is:

- id
- name

Obtain List of Object Types through id's

Here you need to call get() method at ot_module.

You can pass up to 5 parameters:

- **ids**: List containing identifiers of Object Types.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Object Types.

Examples:

```
ots = ot_module.get(ids=[1, 2, 3])
```

Obtain List of Object Types through extended search

Here you need to call search() method at ot_module.

You can pass up to 5 parameters:

- **search**: Dict containing QuerySets to find Object Types.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Object Types.

Example:

```
search = {
    'extends_search': [{
        "name": "Vrf"
    }],
    'start_record': 0,
    'custom_search': '',
    'end_record': 25,
```

```
    'asorting_cols': [],
    'searchable_columns': []}
fields = ['id', 'name']

ots = ot_module.search(search=search, fields=fields)
```

Using Vrf module

Assuming that you have Client Factory instantiated (see *How to Instantiate Client Factory*) at **client** variable, in order to access methods relative to Vrf module you need to call create_api_vrf() at **client**.

Example:

```
vrf_module = client.create_api_vrf()
```

For more information, please look GloboNetworkAPI documentation.

GET

The List of fields available at Vrf module is:

- id
- internal_name
- vrf

Obtain List of Vrfs through id's

Here you need to call get() method at vrf_module.

You can pass up to 5 parameters:

- **ids**: List containing identifiers of vrfs.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Vrfs.

Examples:

```
vrfs = vrf_module.get(ids=[1, 2, 3])
```

Obtain List of Vrfs through extended search

Here you need to call search() method at vrf_module.

You can pass up to 5 parameters:

- **search**: Dict containing QuerySets to find vrfs.
- **include**: Array containing fields to include on response.

- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between “basic” or “details”.

The response will be a dict with a list of Vrfs.

Example:

```
search = {
    'extends_search': [{
        "vrf__contains": "Default"
    }],
    'start_record': 0,
    'custom_search': '',
    'end_record': 25,
    'asorting_cols': [],
    'searchable_columns': []}
fields = ['id', 'name']

vrfs = vrf_module.search(search=search, fields=fields)
```

POST

The List of fields available for create an Vrf is:

- vrf - **Mandatory**
- internal_name - **Mandatory**

Create List of Vrfs

Here you need to call create() method at vrf_module.

You need to pass 1 parameter:

- **vrfs**: List containing vrfs that you want to create.

Example:

```
vrfs_to_create = [
    {
        "vrf": "VrfTest-1",
        "internal_name": "VrfTest-1"
    },
    {
        "vrf": "VrfTest-2",
        "internal_name": "VrfTest-2"
    }
]

vrf_module.create(vrfs=vrfs_to_create)
```

PUT

The List of fields available for update an Vrf is:

- id - **Mandatory**

- vrf - **Mandatory**
- internal_name - **Mandatory**

Update List of Vrfs

Here you need to call update() method at vrf_module.

You need to pass 1 parameter:

- **vrfs**: List containing vrfs that you want to update.

Example:

```
vrfs_to_update = [
    {
        "id": 1,
        "vrf": "VrfTest-1",
        "internal_name": "VrfTest-1"
    },
    {
        "id": 2,
        "vrf": "VrfTest-2",
        "internal_name": "VrfTest-2"
    }
]

vrf_module.update(vrfs=vrfs_to_update)
```

DELETE

Delete List of Vrfs

Here you need to call delete() method at vrf_module.

You need to pass 1 parameter:

- **ids**: List containing identifiers of vrfs that you want to delete.

Example:

```
vrf_module.delete(ids=[1, 2, 3])
```

CHAPTER 4

Using GloboNetworkAPI Client V4

Using AS module

Assuming that you have Client Factory instantiated (see *How to Instantiate Client Factory*) at **client** variable, in order to access methods relative to AS module you need to call create_api_v4_as() at **client**.

Example:

```
ans_module = client.create_api_v4_as()
```

For more information, please look GloboNetworkAPI documentation.

GET

The List of fields available at AS module is:

- id
- name
- description
- equipments

Obtain List of ASNs through id's

Here you need to call get() method at ans_module.

You can pass up to 5 parameters:

- **ids**: List containing identifiers of ASNs.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of ASNs.

Examples:

```
asns = ans_module.get(ids=[1, 2, 3])

asns = ans_module.get(ids=[1, 2, 3],
                      kind='basic')

asns = ans_module.get(ids=[1, 2, 3],
                      fields=['id', 'name'])
```

Obtain List of ASNs through extended search

Here you need to call search() method at ans_module.

You can pass up to 5 parameters:

- **search**: Dict containing QuerySets to find ASNs.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of ASNs.

Example:

```
search = {
    'extends_search': [{
        "name": "AS_BGP"
    }],
    'start_record': 0,
    'custom_search': '',
    'end_record': 25,
    'asorting_cols': [],
    'searchable_columns': []}
fields = ['id', 'name']

asns = ans_module.search(search=search, fields=fields)
```

POST

The List of fields available for create an AS is:

- name - **Mandatory**
- description - **Mandatory**

Create List of ASNs

Here you need to call create() method at ans_module.

You need to pass 1 parameter:

- **asns**: List containing ASNs that you want to create.

Example:

```
asns_to_create = [
    {
        "name": "11",
        "descripton": "AS-11"
    },
    {
        "name": "12",
        "descripton": "AS-12"
    }
]

ans_module.create(asns=asns_to_create)
```

PUT

The List of fields available for update an AS is:

- id - **Mandatory**
- name - **Mandatory**
- description - **Mandatory**

Update List of ASNs

Here you need to call update() method at ans_module.

You need to pass 1 parameter:

- **asns**: List containing ASNs that you want to update.

Example:

```
asns_to_update = [
    {
        "id": 1,
        "name": "13",
        "descripton": "AS-13"
    },
    {
        "id": 2,
        "name": "14",
        "descripton": "AS-14"
    }
]

ans_module.update(asns=asns_to_update)
```

DELETE

Delete List of ASNs

Here you need to call delete() method at ans_module.

You need to pass 1 parameter:

- **ids**: List containing identifiers of ASNs that you want to delete.

Example:

```
ans_module.delete(ids=[1, 2, 3])
```

Using Equipment module

Assuming that you have Client Factory instantiated (see *How to Instantiate Client Factory*) at **client** variable, in order to access methods relative to Equipment module you need to call create_api_v4_equipment() at **client**.

Example:

```
eqpt_module = client.create_api_v4_equipment()
```

For more information, please look GloboNetworkAPI documentation.

GET

The List of fields available at Equipment module is:

- id
- name
- maintenance
- equipment_type
- model
- ipsv4
- ipsv6
- environments
- groups
- id_as

Obtain List of Equipments through id's

Here you need to call get() method at eqpt_module.

You can pass up to 5 parameters:

- **ids**: List containing identifiers of equipments.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Equipments.

Examples:

```
epqts = eqpt_module.get(ids=[1, 2, 3])
```

```
epqts = eqpt_module.get(ids=[1, 2, 3],
                        include=['name', 'maintenance', 'id_as'],
                        exclude=['id'],
                        kind='basic')

epqts = eqpt_module.get(ids=[1, 2, 3],
                        fields=['id', 'name', 'model'])
```

Obtain List of Equipments through extended search

Here you need to call search() method at eqpt_module.

You can pass up to 5 parameters:

- **search**: Dict containing QuerySets to find equipments.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Equipments.

Example:

```
search = {
    'extends_search': [{
        'maintenance': false,
        'tipo_equipamento': 1
    }],
    'start_record': 0,
    'custom_search': '',
    'end_record': 25,
    'asorting_cols': [],
    'searchable_columns': []}
fields = ['id', 'name', 'model', 'id_as', 'ipsv4']

epqts = eqpt_module.search(search=search, fields=fields)
```

POST

The List of fields available for create an Equipment is:

- **environments**
  - id
  - is_router
  - is_controller
- equipment_type - **Mandatory**
- **groups**
  - id
- **ipsv4**

        - **ipv4**
            - id
        - **virtual_interface**
            - id
- **ipsv6**
        - **ipv6**
            - id
        - **virtual_interface**
            - id
- maintenance - **Mandatory**
- model - **Mandatory**
- name - **Mandatory**
- id_as

Create List of Equipments

Here you need to call create() method at eqpt_module.

You need to pass 1 parameter:

- **equipments**: List containing equipments that you want to create.

Example:

```
eqpts_to_create = [
    {
        "name": "Eqpt-1",
        "maintenance": False,
        "equipment_type": 8,
        "model": 3,
        "environments": [
            {
                "id": 1,
                "is_router": True,
                "is_controller": False
            },
            {
                "id": 2,
                "is_router": False,
                "is_controller": False
            }
        ],
        "ipsv4": [
            {
                "ipv4": {
                    "id": 1
                },
                "virtual_interface": {
                    "id": 4
                }
            },
```

```
            {
                "ipv4": {
                    "id": 2
                },
                "virtual_interface": {
                    "id": None
                }
            }
        ],
        "id_as": 2
    },
    {
        "name": "Eqpt-2",
        "maintenance": False,
        "equipment_type": 9,
        "model": 3,
        "ipsv6": [
            {
                "ipv6": {
                    "id": 1
                },
                "virtual_interface": {
                    "id": 4
                }
            },
            {
                "ipv6": {
                    "id": 2
                },
                "virtual_interface": {
                    "id": None
                }
            }
        ],
        "groups": [
            {
                "id": 1
            },
            {
                "id": 2
            }
        ]
    }
]

eqpt_module.create(equipments=eqpts_to_create)
```

PUT

The List of fields available for update an Equipment is:

- id - **Mandatory**
- **environments**
    - id
    - is_router

        - is_controller
- equipment_type - **Mandatory**
- **groups**
        - id
- **ipsv4**
        - **ipv4**
            * id
        - **virtual_interface**
            * id
- **ipsv6**
        - **ipv6**
            * id
        - **virtual_interface**
            * id
- maintenance - **Mandatory**
- model - **Mandatory**
- name - **Mandatory**
- id_as

Update List of Equipments

Here you need to call update() method at eqpt_module.

You need to pass 1 parameter:

- **equipments**: List containing equipments that you want to update.

Example:

```
eqpts_to_update = [
    {
        "id": 1,
        "name": "Eqpt-1-Updated",
        "maintenance": False,
        "equipment_type": 2,
        "model": 2,
        "environments": [
            {
                "id": 2,
                "is_router": True,
                "is_controller": False
            }
        ],
        "id_as": 3,
        "ipsv4": [
            {
                "ipv4": {
                    "id": 1
```

```
                },
                "virtual_interface": {
                    "id": 4
                }
            },
            {
                "ipv4": {
                    "id": 5
                },
                "virtual_interface": {
                    "id": None
                }
            }
        ]
    },
    {
        "id": 2,
        "name": "Eqpt-2-Updated",
        "maintenance": False,
        "equipment_type": 7,
        "model": 2,
        "ipsv6": [
            {
                "ipv6": {
                    "id": 1
                },
                "virtual_interface": {
                    "id": 4
                }
            },
            {
                "ipv6": {
                    "id": 3
                },
                "virtual_interface": {
                    "id": None
                }
            }
        ],
        "groups": [
            {
                "id": 2
            },
            {
                "id": 3
            }
        ]
    }
]

eqpt_module.update(equipments=eqpts_to_update)
```

DELETE

Delete List of Equipments

Here you need to call delete() method at eqpt_module.

You need to pass 1 parameter:

- **ids**: List containing identifiers of equipments that you want to delete.

Example:

```
eqpt_module.delete(ids=[1, 2, 3])
```

Using IPv4 module

Assuming that you have Client Factory instantiated (see *How to Instantiate Client Factory*) at **client** variable, in order to access methods relative to IPv4 module you need to call create_api_v4_ipv4() at **client**.

Example:

```
ipv4_module = client.create_api_v4_ipv4()
```

For more information, please look GloboNetworkAPI documentation.

GET

The List of fields available at IPv4 module is:

- id
- ip_formated
- oct1
- oct2
- oct3
- oct4
- networkipv4
- description
- equipments
- vips
- server_pool_members

Obtain List of IPv4's through id's

Here you need to call get() method at ipv4_module.

You can pass up to 5 parameters:

- **ids**: List containing identifiers of IPv4's.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.

- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between “basic” or “details”.

The response will be a dict with a list of IPv4’s.

Examples:

```
ipv4s = ipv4_module.get(ids=[1, 2, 3])

ipv4s = ipv4_module.get(ids=[1, 2, 3],
                        include=['oct1', 'oct2'],
                        exclude=['oct3'],
                        kind='details')

ipv4s = ipv4_module.get(ids=[1, 2, 3],
                        fields=['id', 'oct1', 'oct2', 'networkipv4'])
```

Obtain List of IPv4’s through extended search

Here you need to call search() method at ipv4_module.

You can pass up to 5 parameters:

- **search**: Dict containing QuerySets to find IPv4’s.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between “basic” or “details”.

The response will be a dict with a list of IPv4’s.

Example:

```
search = {
    'extends_search': [{
        {
            'oct1': 10,
        },
        {
            'oct1': 172,
        }
    }],
    'start_record': 0,
    'custom_search': '',
    'end_record': 25,
    'asorting_cols': [],
    'searchable_columns': []}
fields = ['id', 'oct1', 'oct2', 'vips']

ipv4s = ipv4_module.search(search=search, fields=fields)
```

POST

The List of fields available for create an IPv4 is:

- oct1

- oct2
- oct3
- oct4
- networkipv4 - **Mandatory**
- description
- **equipments**
    - **equipment**
        - id
    - **virtual_interface**
        - id

Create List of IPv4's

Here you need to call create() method at ipv4_module.

You need to pass 1 parameter:

- **ipv4s**: List containing IPv4's that you want to create.

Example:

```
ipv4s_to_create = [
    {
        "oct1": 10,
        "oct2": 10,
        "oct3": 0,
        "oct4": 2,
        "description": "IP 2",
        "networkipv4": 1
    },
    {
        "description": "IP 1",
        "networkipv4": 2,
        "equipments": [
            {
                "equipment": {
                    "id": 1
                },
                "virtual_interface": {
                    "id": 1
                }
            },
            {
                "equipment": {
                    "id": 2
                },
                "virtual_interface": {
                    "id": 2
                }
            }
        ]
    }
]
```

```
ipv4_module.create(ipv4s=ipv4s_to_create)
```

PUT

The List of fields available for update an IPv4 is:

- id - **Mandatory**
- description
- **equipments**
    - **equipment**
        - id
    - **virtual_interface**
        - id

Update List of IPv4's

Here you need to call update() method at ipv4_module.

You need to pass 1 parameter:

- **ipv4s**: List containing ipv4s that you want to update.

Example:

```
ipv4s_to_update = [
    {
        "id": 1,
        "description": "New-Desc-1"
    },
    {
        "id": 2,
        "equipments": [
            {
                "equipment": {
                    "id": 2
                },
                "virtual_interface": {
                    "id": 2
                }
            },
            {
                "equipment": {
                    "id": 4
                },
                "virtual_interface": {
                    "id": 4
                }
            }
        ]
    }
]

ipv4_module.update(ipv4s=ipv4s_to_update)
```

DELETE

Delete List of IPv4's

Here you need to call delete() method at ipv4_module.

You need to pass 1 parameter:

- **ids**: List containing identifiers of IPv4's that you want to delete.

Example:

```
ipv4_module.delete(ids=[1, 2, 3])
```

Using IPv6 module

Assuming that you have Client Factory instantiated (see *How to Instantiate Client Factory*) at **client** variable, in order to access methods relative to IPv6 module you need to call create_api_v4_ipv6() at **client**.

Example:

```
ipv6_module = client.create_api_v4_ipv6()
```

For more information, please look GloboNetworkAPI documentation.

GET

The List of fields available at IPv6 module is:

- id
- ip_formated
- block1
- block2
- block3
- block4
- block5
- block6
- block7
- block8
- networkipv6
- description
- equipments
- vips
- server_pool_members

Obtain List of IPv6's through id's

Here you need to call get() method at ipv6_module.

You can pass up to 5 parameters:

- **ids**: List containing identifiers of IPv6's.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of IPv6's.

Examples:

```
ipv6s = ipv6_module.get(ids=[1, 2, 3])

ipv6s = ipv6_module.get(ids=[1, 2, 3],
                        include=['block1', 'block2'],
                        exclude=['block3'],
                        kind='details')

ipv6s = ipv6_module.get(ids=[1, 2, 3],
                        fields=['id', 'block1', 'block2', 'networkipv6'])
```

Obtain List of IPv6's through extended search

Here you need to call search() method at ipv6_module.

You can pass up to 5 parameters:

- **search**: Dict containing QuerySets to find IPv6's.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of IPv6's.

Example:

```
search = {
    'extends_search': [{
        {
            "block1": "fefe"
        },
        {
            "block1": "fdfd"
        }
    }],
    'start_record': 0,
    'custom_search': '',
    'end_record': 25,
    'asorting_cols': [],
```

```
    'searchable_columns': []}
fields = ['id', 'block1', 'block2', 'vips']

ipv6s = ipv6_module.search(search=search, fields=fields)
```

POST

The List of fields available for create an IPv6 is:

- block1
- block2
- block3
- block4
- block5
- block6
- block7
- block8
- networkipv6 - **Mandatory**
- description
- **equipments**
  - **equipment**
    - id
  - **virtual_interface**
    - id

Create List of IPv6's

Here you need to call create() method at ipv6_module.

You need to pass 1 parameter:

- **ipv6s**: List containing IPv6's that you want to create.

Example:

```
ipv6s_to_create = [
    {

        "block1": "fefe"
        "block2": "a0a0"
        "block3": "a0a0"
        "block4": "a0a0"
        "block5": "0000"
        "block6": "0000"
        "block7": "0000"
        "block8": "0002"
        "description": "IP 2",
        "networkipv6": 1
```

```
    },
    {
        "description": "IP 1",
        "networkipv6": 2,
        "equipments": [
            {
                "equipment": {
                    "id": 1
                },
                "virtual_interface": {
                    "id": 1
                }
            },
            {
                "equipment": {
                    "id": 2
                },
                "virtual_interface": {
                    "id": 2
                }
            }
        ]
    }
]

ipv6_module.create(ipv6s=ipv6s_to_create)
```

PUT

The List of fields available for update an IPv6 is:

- id - **Mandatory**
- description
- **equipments**
    - **equipment**
        - id
    - **virtual_interface**
        - id

Update List of IPv6's

Here you need to call update() method at ipv6_module.

You need to pass 1 parameter:

- **ipv6s**: List containing ipv6s that you want to update.

Example:

```
ipv6s_to_update = [
    {
        "id": 1,
        "description": "New-Desc-1"
```

```
    },
    {
        "id": 2,
        "equipments": [
            {
                "equipment": {
                    "id": 2
                },
                "virtual_interface": {
                    "id": 2
                }
            },
            {
                "equipment": {
                    "id": 4
                },
                "virtual_interface": {
                    "id": 4
                }
            }
        ]
    }
]

ipv6_module.update(ipv6s=ipv6s_to_update)
```

DELETE

Delete List of IPv6's

Here you need to call delete() method at ipv6_module.

You need to pass 1 parameter:

- **ids**: List containing identifiers of IPv6's that you want to delete.

Example:

```
ipv6_module.delete(ids=[1, 2, 3])
```

Using Neighbor module

Assuming that you have Client Factory instantiated (see *How to Instantiate Client Factory*) at **client** variable, in order to access methods relative to Neighbor module you need to call create_api_v4_neighbor() at **client**.

Example:

```
neighbor_module = client.create_api_v4_neighbor()
```

For more information, please look GloboNetworkAPI documentation.

GET

The List of fields available at Neighbor module is:

- id
- remote_as
- remote_ip
- password
- maximum_hops
- timer_keepalive
- timer_timeout
- description
- soft_reconfiguration
- community
- remove_private_as
- next_hop_self
- kind
- created
- virtual_interface

Obtain List of Neighbors through id's

Here you need to call get() method at neighbor_module.

You can pass up to 5 parameters:

- **ids**: List containing identifiers of neighbors.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Neighbors.

Examples:

```
neighbors = neighbor_module.get(ids=[1, 2, 3])

neighbors = neighbor_module.get(ids=[1, 2, 3],
                                include=['virtual_interface', 'kind'],
                                exclude=['id'])

neighbors = neighbor_module.get(ids=[1, 2, 3],
                                fields=['id', 'remote_as', 'remote_ip'])
```

Obtain List of Neighbors through extended search

Here you need to call search() method at neighbor_module.

You can pass up to 5 parameters:

- **search**: Dict containing QuerySets to find neighbors.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Neighbors.

Example:

```
search = {
    'extends_search': [{
        "community": false
    }],
    'start_record': 0,
    'custom_search': '',
    'end_record': 25,
    'asorting_cols': [],
    'searchable_columns': []}
fields = ['id', 'remote_as']

neighbors = neighbor_module.search(search=search, fields=fields)
```

POST

The List of fields available for create an Neighbor is:

- remote_as
- remote_ip
- password
- maximum_hops
- timer_keepalive
- timer_timeout
- description
- soft_reconfiguration
- community
- remove_private_as
- next_hop_self
- kind
- virtual_interface - **Mandatory**

Create List of Neighbors

Here you need to call create() method at neighbor_module.

You need to pass 1 parameter:

- **neighbors**: List containing neighbors that you want to create.

Example:

```
neighbors_to_create = [
    {
        "remote_as": "203",
        "remote_ip": "10.10.0.2",
        "password": "Test-pwd",
        "maximum_hops": "5",
        "timer_keepalive": "3",
        "timer_timeout": "60",
        "description": "any",
        "soft_reconfiguration": False,
        "community": True,
        "remove_private_as": True,
        "next_hop_self": False,
        "kind": "I",
        "virtual_interface": 1,
    },
    {
        "remote_as": "203",
        "remote_ip": "10.10.0.3",
        "password": "Test-pwd",
        "maximum_hops": "5",
        "timer_keepalive": "3",
        "timer_timeout": "60",
        "description": "any",
        "soft_reconfiguration": False,
        "community": True,
        "remove_private_as": True,
        "next_hop_self": True,
        "kind": "E",
        "virtual_interface": 2
    }
]

neighbor_module.create(neighbors=neighbors_to_create)
```

PUT

The List of fields available for update an Neighbor is:

- id - **Mandatory**
- remote_as
- remote_ip
- password
- maximum_hops
- timer_keepalive
- timer_timeout
- description
- soft_reconfiguration
- community

- remove_private_as
- next_hop_self
- kind
- virtual_interface - **Mandatory**

Update List of Neighbors

Here you need to call update() method at neighbor_module.

You need to pass 1 parameter:

- **neighbors**: List containing neighbors that you want to update.

Example:

```
neighbors_to_update = [
    {
        "id": 1,
        "remote_as": "203",
        "remote_ip": "10.10.0.4",
        "password": "Test-pwd",
        "maximum_hops": "5",
        "timer_keepalive": "4",
        "timer_timeout": "70",
        "description": "any",
        "soft_reconfiguration": False,
        "community": True,
        "remove_private_as": True,
        "next_hop_self": False,
        "kind": "I",
        "virtual_interface": 3,
    },
    {
        "id": 2,
        "remote_as": "203",
        "remote_ip": "10.10.0.5",
        "password": "Test-pwd-2",
        "maximum_hops": "7",
        "timer_keepalive": "3",
        "timer_timeout": "70",
        "description": "any",
        "soft_reconfiguration": True,
        "community": True,
        "remove_private_as": True,
        "next_hop_self": True,
        "kind": "E",
        "virtual_interface": 2
    }
]

neighbor_module.update(neighbors=neighbors_to_update)
```

DELETE

Delete List of Neighbors

Here you need to call delete() method at neighbor_module.

You need to pass 1 parameter:

- **ids**: List containing identifiers of neighbors that you want to delete.

Example:

```
neighbor_module.delete(ids=[1, 2, 3])
```

Using Virtual Interface module

Assuming that you have Client Factory instantiated (see *How to Instantiate Client Factory*) at **client** variable, in order to access methods relative to Virtual Interface module you need to call create_api_v4_virtual_interface() at **client**.

Example:

```
vi_module = client.create_api_v4_virtual_interface()
```

For more information, please look GloboNetworkAPI documentation.

GET

The List of fields available at Virtual Interface module is:

- id
- name
- vrf

Obtain List of Virtual Interfaces through id's

Here you need to call get() method at vi_module.

You can pass up to 5 parameters:

- **ids**: List containing identifiers of virtual interfaces.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Virtual Interfaces.

Examples:

```
vis = vi_module.get(ids=[1, 2, 3])

vis = vi_module.get(ids=[1, 2, 3],
                    kind='basic')
```

```
vis = vi_module.get(ids=[1, 2, 3],
                    fields=['name'])
```

Obtain List of Virtual Interfaces through extended search

Here you need to call search() method at vi_module.

You can pass up to 5 parameters:

- **search**: Dict containing QuerySets to find virtual interfaces.
- **include**: Array containing fields to include on response.
- **exclude**: Array containing fields to exclude on response.
- **fields**: Array containing fields to override default fields.
- **kind**: string where you can choose between "basic" or "details".

The response will be a dict with a list of Virtual Interfaces.

Example:

```
search = {
    'extends_search': [{
        "vrf__id": 1,
        "name__contains": "abc"
    }],
    'start_record': 0,
    'custom_search': '',
    'end_record': 25,
    'asorting_cols': [],
    'searchable_columns': []}
fields = ['id', 'name']

vis = vi_module.search(search=search, fields=fields)
```

POST

The List of fields available for create an Virtual Interface is:

- vrf - **Mandatory**
- name - **Mandatory**

Create List of Virtual Interfaces

Here you need to call create() method at vi_module.

You need to pass 1 parameter:

- **virtual_interfaces**: List containing virtual interfaces that you want to create.

Example:

```
vis_to_create = [
    {
        "vrf": 1,
        "name": "Virt-1"
```

```
    },
    {
        "vrf": 2,
        "name": "Virt-2"
    }
]

vi_module.create(virtual_interfaces=vis_to_create)
```

PUT

The List of fields available for update an Virtual Interface is:

- id - **Mandatory**
- vrf - **Mandatory**
- name - **Mandatory**

Update List of Virtual Interfaces

Here you need to call update() method at vi_module.

You need to pass 1 parameter:

- **virtual_interfaces**: List containing virtual interfaces that you want to update.

Example:

```
vis_to_update = [
    {
        "id": 1,
        "vrf": 1,
        "name": "Virt-3"
    },
    {
        "id": 2,
        "vrf": 4,
        "name": "Virt-2"
    }
]

vi_module.update(virtual_interfaces=vis_to_update)
```

DELETE

Delete List of Virtual Interfaces

Here you need to call delete() method at vi_module.

You need to pass 1 parameter:

- **ids**: List containing identifiers of virtual interfaces that you want to delete.

Example:

```
vi_module.delete(ids=[1, 2, 3])
```

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*

n

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

X