ibtidelib Documentation

Release 0.8.6

Chris Marrison

Dec 17, 2021

Contents:

CHAPTER 1

Introduction

The Infoblox BloxOne suite of applications provides a RESTful API that is published using Swagger on https://csp.infoblox.com/apidoc along with other Infoblox APIs.

This module aims to provide a class hierarchy to simplify access to these published APIs, performing the 'heavy lifting' whilst providing full access to to their functionality. This is achieved by providing simple wrappers that enable you to take the swagger documented object paths, fields and where appropriate JSON body from the documentation and pass them to simple get, create, delete and update methods. These methods simply return a *requests* response object.

In addition, useful utility methods are provided for common tasks such as getting an object id, by defining the object key and value match pair. This is combined with several (currently) undocumented API calls.

Some basic configuration, such a base url, API version and API key are read from an ini file. An example of which is provided. When instantiating/initialising this will read config.ini by default. Alternatively a path can be provided.

PyPi: https://pypi.org/project/bloxone/

GitHub: https://github.com/ccmarris/python-bloxone

CHAPTER 2

License

CHAPTER 3

Requirements

Important: `bloxone` requires Python 3.x and was developed under Python 3.8. As such it is unlikely to work with Python 2.x.

3.1 Additional Modules

In addition to the standard Python 3 Modules `bloxone` makes use of the following packages:

- `requests`
- `pyyaml`

3.2 Complete List Of Modules

Modules used by bloxone

```
import logging
import configparser
import requests
import json
import datetime
import yaml
```

CHAPTER 4

Installation

Important: Please ensure you have Python 3 plus the required modules as defined in the *Requirements* section.

Note: This was developed under Python 3.8 but may work on earlier versions, but has not been tested. It is suggested a minimum version of 3.6, but may work with earlier versions.

Important: Mac users will need the xcode command line utilities installed to use pip3, etc. If you need to install these use the command:

```
$ xcode-select --install
```

4.1 Installing Python

You can install the latest version of Python 3.x by downloading the appropriate installer for your system from python.org.

Note: If you are running MacOS Catalina (or later) Python 3 comes pre-installed. Previous versions only come with Python 2.x by default and you will therefore need to install Python 3 as above or via Homebrew, Ports, etc.

By default the python command points to Python 2.x, you can check this using the command:

```
$ python -V
```

To specifically run Python 3, use the command:

```
$ python3
```

Note: If you are installing Python on Windows, be sure to check the box to have Python added to your PATH if the installer offers such an option (it's normally off by default).

4.2 Installing `bloxone`

1. Install from PyPI

The `bloxone` module is available on PyPI and can simply be installed using pip/pip3:

```
pip3 install bloxone <--user>
```

1. Intall bloxone as local package from source

If you have downloaded the source from GitHub then `bloxone` has been provided as an installable package using pip.

In your appropriate python3 environment you can therefore install `bloxone` using the pip command from the root of the sourcetree. For example:

```
pip3 install dist/bloxone-<version>-py3-none-any.whl --user

or

pip3 install dist/bloxone-<version>.tar.gz --user
```

Note: Use of pip ensures module dependencies and allows for uninstallation and upgrades to be handled cleanly.

4.3 Uninstalling `bloxone`

You can use pip to unintsall the library. For example:

```
pip3 uninstall bloxone
```

CHAPTER 5

Usage and Examples

5.1 Inifile configuration

A sample inifile for the bloxone module is shared as *bloxone.ini* and follows the following format provided below:

```
[BloxOne]
url = 'https://csp.infoblox.com'
api_version = 'v1'
api_key = '<you API Key here>'
```

You can therefore simply add your API Key, and this is ready for the bloxone module used by the automation demo script. Legacy, interactive and service API keys are supported.

5.2 Overview

The aim of this module is to provide simple object based access to the BloxOne APIs and make it as simple as possible to code given the available swagger documentation.

There are several classes/subclasses that provide this access. The base class is `b1`. This acts as a parent class for the BloxOne Application APIs.

The specific API 'trees' are then split in to subclasses of `b1`:

`b1ddi` Access to the BloxOne DDI API with core methods for *get*, *create*, *delete* and *update* in addition to specific task orientated helper methods.

`b1td` Access to the Infoblox TIDE API with a generic *get* method plus specific task orientated helper methods.

`b1tdc` Access to the BloxOne Threat Defence Cloud API with a generic *get*, *create*, *delete* methods plus specific task orientated helper methods.

`b1tdep` Access to the BloxOne Threat Defence Cloud API with a generic *get*, *create*, *delete* and *update* methods plus specific task orientated helper methods.

b1tddfp Access to the BloxOne Threat Defence Cloud API with a generic *get*, and *update* methods plus specific task orientated helper methods.

b1tdlad Access to the BloxOne Threat Defence Cloud API with a generic *get*, method.

b1anycast Access to the BloxOne Anycast API with a generic *get*, *create*, *delete* and *update* methods plus specific task

b1authn Access to the BloxOne On-Prem Authentication Service API with generic *get*, *create*, *delete* and *update* methods plus specific task

b1bootstrap Access to the BloxOne On-Prem Bootstrap App API with generic *get*, *create*, *delete* and *update* methods plus specific task

b1cdc Access to the BloxOne Data Connector API with generic *get*, *create*, *delete* and *update* methods plus specific task

b1diagnostics Allows the user to execute remote commands on an OPH via the API

b1oph Access to the BloxOne On Prem Host API with generic *get*, *create*, *delete* and *update* methods plus specific tasks to allow simple status reporting, App control, etc.

b1platform Methods to provide access to users and audit log information

b1sw Access to the BloxOne Software Upgrade Scheduling API with generic *get*, *create*, *delete* and *update* methods plus specific task

b1ztp Access to the BloxOne On Prem Host Host Activation API with generic *get*, *create*, *delete* and *update* methods plus specific task

In addition to the API interfaces a set of data handling functions is provided in the `utils` sub-module.

5.3 Basic Usage

Using BloxOne DDI as an example, the basic usage structure for a *get* is:

```
import bloxone
b1ddi = bloxone.b1ddi(<ini file>)
response = b1ddi.get(<object path>)
if response.status_code in b1ddi.return_codes_ok:
    print(response.text)
else:
    print(response.status_code)
```

Similarly for the other core functions, and classes. For details on method specific parameters, please see the *class documentation*

For debugging purposes, the `bloxone` module supports logging using `logging` using DEBUG.

Warning: I have attempted to keep debugging clean, however, there is still potential for the debug output to produce full data dumps of API responses.

5.4 Generic API Wrapper

It is also possible to use the bloxone.b1 class as a generic API wrapper with public methods for *get*, *create*, *update* and *delete*. These can be used to pass a full URL, and where appropriate body and parameters. It is of course possible to

build the URL using the attributes of this class, in addition to manually entering the full url:

```
import bloxone
b1 = bloxone.b1(<ini file>)
url = 'https://csp.infoblox.com/api/ddi/v1/ipam/ip_space'
response = b1.get(url)
print(response.json())

url = b1.ddi_url + '/ipam/ip_space'
response = b1.get(url, _filter='name=="test_ip_space"')
print(response.json())
```

5.5 Examples

Although the basic flow of: instantiating the class with a configuration ini file; access the attributes or methods, with *get* almost being universal as a method, and using the swagger object paths to access the required resource. Specific examples for each of the classes, and their use, is shown in more detail in the following documents, as well as the usage of `utils`:

5.5.1 Examples for class: b1ddi

The aim of this class is to provide simple object based access to the BloxOne DDI API and make it as simple as possible to code given the available swagger documentation.

5.5.1.1 Basic Usage

For BloxOne DDI therefore the basic usage structure for a *get* is:

```
import bloxone
b1ddi = bloxone.b1ddi(<cini file>)
response = b1ddi.get(<object path>)
if response.status_code in b1ddi.return_codes_ok:
    print(response.text)
else:
    print(response.status_code)
```

With create and update the key difference is that a JSON body is supplied:

```
payload = '{ "address": "10.0.0.0", "cidr": "24" }'
response = b1ddi.create('/ipam/subnet', body=payload)
if repsonse.status_code in b1ddi.return_codes_ok:
    print(repsonse.text)
else:
    print(response.status_code)
    print(response.text)
```

For a complete list of supported methods and details around parameters, please see the *class documentation*

5.5.1.2 Examples

Todo: These examples are placeholders, useful, but actual example set here is on the todo.

```
# Note this is a set of rough examples to show method calls

import bloxone
import logging

log=logging.getLogger(__name__)
# logging.basicConfig(level=logging.DEBUG)

# Create a BloxOne DDI Object
b1ddi = bloxone.b1ddi()

# Your can specify ini filename/path
# b1ddi = bloxone.b1ddi('/Users/<username/<path>/<inifile>')

# Show API Key
b1ddi.api_key
b1ddi.api_version

# Generic Get wrapper using "Swagger Path for object"
# response = b1ddi.get('<swagger path>')
response = b1ddi.get('/ipam/ip_space')

# Response object handling
response.status_code
response.text
response.json()

# Using custom parameters
response = b1ddi.get('/dns/view', _fields="name,id")
response.json()

# Example using _filter
response = b1ddi.get('/ipam/ip_space', _filter='name=="space-name"')

# Example with multiple API parameters
response = b1ddi.get('/ipam/subnet', _tfilter="Owner==marrison", _fields="address")
response = b1ddi.get('/ipam/subnet', _tfilter="Owner~mar")

# Get ID from key/value pair
id = b1ddi.get_id('/dns/auth_zone', key="fqdn", value="home.")
# Example Result: '80b0e234-8d5b-465b-8c98-e9430c5d83a9'

id = b1ddi.get_id('/ipam/ip_space', key="name", value="marrison-lab", include_
↪path=True)
# 'ipam/ip_space/fd388619-b013-11ea-b956-ca543bd8c483'

# Get DHCP Option IDs as a dictionary
options = b1ddi.get_option_ids()
options['43']
# 'dhcp/option_code/44bbac08-c518-11ea-b9d9-06bf0d811d6d'

# Get data for zone
r = b1ddi.get_zone_child(parent="zone", name="home.", fields="name,record_type,record_
↪data")
```

(continues on next page)

(continued from previous page)

```
# Get all on_prem_hosts
# Create b1platform object
b1p = bloxone.b1platform()
response = b1p.on_prem_hosts()
response.text

# Using tag filters
response = b1p.on_prem_hosts(_tfilter="Owner==marrison")
response.text

# Get all records for a 'named' zone
response = b1ddi.get_zone_child(name="home.")
response.text

# Get all zones in a view by view name
response = b1ddi.get_zone_child(name="marrison-dns-view1")
response.text

# Create Examples body = ( '{ "name": "my-ip-space", "tags": { "Owner":
                            "marrison" }}' )
r = b1ddi.create('/ipam/ip_space', body=body)
r.text

# '{"result":{"asm_config":{"asm_threshold":90,"enable":true,"enable_notification
↪":true,"forecast_period":14,"growth_factor":20,"growth_type":"percent","history":30,
↪"min_total":10,"min_unused":10,"reenable_date":"1970-01-01T00:00:00Z"},"asm_scope_
↪flag":0,"comment":"","dhcp_config":{"allow_unknown":true,"filters":[],"ignore_list
↪":[],"lease_time":3600},"dhcp_options":[],"id":"ipam/ip_space/edfb2cde-c2fc-11ea-
↪b5c8-3670d2b79356","inheritance_sources":null,"name":"marrison-test","tags":null,
↪"threshold":{"enabled":false,"high":0,"low":0},"utilization":{"abandon_utilization
↪":0,"abandoned":"0","dynamic":"0","free":"0","static":"0","total":"0","used":"0",
↪"utilization":0}}}'

r = b1ddi.get_object_by_key('/ipam/ip_space', key="name", value="marrison-lab")
r.text
# '{"result":{"asm_config":{"asm_threshold":90,"enable":true,"enable_notification
↪":true,"forecast_period":14,"growth_factor":20,"growth_type":"percent","history":30,
↪"min_total":10,"min_unused":10,"reenable_date":"1970-01-01T00:00:00Z"},"asm_scope_
↪flag":0,"comment":"","dhcp_config":{"allow_unknown":true,"filters":[],"ignore_list
↪":[],"lease_time":43200},"dhcp_options":[],"id":"ipam/ip_space/fd388619-b013-11ea-
↪b956-ca543bd8c483","inheritance_sources":null,"name":"marrison-lab","tags":{
↪"Location":"Hampshire, UK","Owner":"marrison"},"threshold":{"enabled":false,"high
↪":0,"low":0},"utilization":{"abandon_utilization":0,"abandoned":"0","dynamic":"40",
↪"free":"65491","static":"5","total":"65536","used":"45","utilization":0}}}'


# Update tags on an on_prem_hosts object example
# Create a b1platform object
b1p = bloxone.b1platform('/Users/marrison/bin/tide.ini')
# Note: this will change the "tags" i.e. replace the "tags" with the "tags" in the␣
↪update body
body = '{"display_name":"marrison-hw-ddi1", "tags":{"Location":"Hampshire, UK","Owner
↪":"marrison","host/deployment_type":"APPLIANCE","host/k8s":"false","host/ophid":
↪"63f2b1c3f80455d87186aa054e87f1a9"}}'
# Call the update method
response = b1p.update('/on_prem_hosts', id="97290", body=body)
```

5.5.2 b1diagnostics Usage

The `b1diagnostics` provides the ability to run remote commands on an OPH via the API and download the results.

5.5.2.1 Examples

Todo: These are simple examples to show you usage of the class. More comprehensive documentation is on the todo.

```
from pprint import pprint
import bloxone

# Instantiate class with ini file as argument
diag = bloxone.b1diagnostics('<path to ini>')

# Show remote commands and args
pprint(diag.commands)

# Check a commmand is supported
if diag.is_command('dns_test'):
    print('dns_test is a supported command')

# Get supported arguments
cmd_args = diag.get_args('dns_test')

# Set up dictionary containing required args
args = {'domain_name': 'www.google.com'}

# Execute command and get id
id = diag.execute_task('dns_test', args=args, ophname='youroph-name')

# Get the JSON form of the task results
response = diag.get_task_results(id)
pprint(response.json())

# 'Download' the results (returns text/plain)
text = diag.download_task_results(id).text
pprint(text)

# Get the raw request object for the API call
response = diag.execute_task('dns_test',
                             args=args,
                             ophname='youroph-name',
                             id_only=False)

# Run a privileged task
id = diag.execute_task('reboot',
                       ophname='youroph-name',
                       priv=True)
response = diag.get_task_results(id)

# Use the ophid rather than name of the OPH (perhaps you already have it)
b1oph = bloxone.b1oph('<path to ini>')
ophid = b1oph.get_ophid(name='youroph-name')

id = diag.execute_task('dns_test', args=args, ophid=ophid)
```

5.5.3 b1td Usage

5.5.3.1 Examples

Todo: These examples are placeholders, useful, but actual example set here is on the todo.

```
bloxone.utils.reverse_labels("www.infoblox.com")

import bloxone
bloxone.__version__
t = bloxone.b1td('/Users/marrison/configs/emea.ini')
t.tide_url
t.threat_classes().json()
t.threat_properties().json()
t.threat_properties(threatclass="malwareC2").json()
t.threat_properties(threatclass="malwareC2").json()
t.threat_counts().json()
t.historical_threat_counts().json()
t.default_ttl().json()
t.dossier_target_types().json()
t.dossier_sources().json()
t.dossier_target_sources().json()
t.dossierquery("eicar.co").json()
t.dossierquery([ "eicar.co", "pwn.af" ]).json()
t.dossierquery([ "eicar.co", "pwn.af" ], sources="atp").json()
t.dossierquery([ "eicar.co", "pwn.af" ], sources=["atp","whois"]).json()
t.expand_mitre_vector('DGA').json()
t.threat_actor('APT1').json()
```

5.5.4 b1tdc Usage

5.5.4.1 Examples

Todo: These examples are placeholders, useful, but actual example set here is on the todo.

```
import bloxone
tdc = bloxone.b1tdc('/Users/marrison/configs/emea.ini')
tdc.get('/access_codes').json()
tdc.get('/cert_download_urls').json()
tdc.get('/content_categories').json()
tdc.get('/threat_feeds').json()
```

5.5.5 b1td Usage

5.5.5.1 Examples

Todo: These examples are placeholders, useful, but actual example set here is on the todo.

```
bloxone.utils.reverse_labels("www.infoblox.com")

import bloxone
t = bloxone.b1td('/Users/marrison/configs/emea.ini')
t.version
bloxone.__version__

import bloxone
tdc = bloxone.b1tdc('/Users/marrison/configs/emea.ini')
tdc.get('/access_codes').json()
tdc.get('/cert_download_urls').json()
tdc.get('/content_categories').json()
lad = bloxone.b1lad('/Users/marrison/configs/emea.ini')
lad = bloxone.b1tdlad('/Users/marrison/configs/emea.ini')
lad.get('/lookalike_domains').json()
tdc.get('/threat_feeds').json()
import readline; print('\n'.join([str(readline.get_history_item(i + 1)) for i in␣
↪range(readline.get_current_history_length())]))
>>>
```

5.5.6 b1tddfp Usage

5.5.6.1 Examples

Todo: These examples are placeholders, useful, but actual example set here is on the todo.

```
import bloxone
dfp = bloxone.b1tddfp('/Users/marrison/configs/emea.ini')
response = dfp.get('/dfps')
response = dfp.get('/dfps', id=dfp_id)
```

5.5.7 b1tdlad Usage

5.5.7.1 Examples

Todo: These examples are placeholders, useful, but actual example set here is on the todo.

```
import bloxone
lad = bloxone.b1tdlad('/Users/marrison/configs/emea.ini')
lad.get('/lookalike_domains').json()
```

5.5.8 b1oph Usage

The `b1oph` provides generic calls to manage API calls for OPH management. Additional 'helper' methods such as get_ophid(), and the ability to see the status of OPHs, Apps, and provide App control.

Todo: Rename OPH

5.5.8.1 Examples

Todo: These are simple examples to show you usage of the class. More comprehensive documentation is on the todo.

```
from pprint import pprint
import bloxone

# Instantiate class with ini file as argument
oph = bloxone.b1oph('<path to ini>')

# Get the ophid
ophid = b1oph.get_ophid(name='youroph-name')


# Get status for all OPHs
>>> pprint.pprint(b1oph.oph_status_summary())
# Get status for specific OPH
>>> pprint.pprint(b1oph.oph_status_summary(name="my-oph-name"))
{'my-oph-name': {'applications': {'Anycast': 'disabled - stopped',
                                  'CDC': 'disabled - stopped',
                                  'CDC_version': 'v2.1.3',
                                  'DFP': 'disabled - stopped',
                                  'DFP_version': 'v2.1.5',
                                  'DHCP': 'active',
                                  'DHCP_version': 'v3.1.8',
                                  'DNS': 'active',
                                  'DNS_version': 'v3.1.4',
                                  'NGC': 'disabled - stopped'},
                 'host_type': 'BloxOne Appliance - B105',
                 'id': '97310',
                 'ip_address': '192.168.1.102',
                 'last_seen': '2021-11-04T19:46:55.942540Z',
                 'nat_ip': None,
                 'status': {'Application Management': 'Online',
                            'OPH State': 'Online',
                            'Platform Management': 'Online'},
                 'version': 'v4.3.6'}}

# Get status for individual app on specified OPH
>>> b1oph.get_app_state(name="my-oph-name", app="DNS")
'active'
# Get status for individual app on specified OPH
>>> b1oph.get_app_state(name="my-oph-name", app="CDC")
'disabled - stopped'

# Perform an action on a an App for specified OPH
>>> b1oph.manage_app(name="my-oph-name", app="CDC", action="start")
False
>>> b1oph.manage_app(name="my-oph-name", app="CDC", action="enable")
True
>>> b1oph.get_app_state(name="non-existent-oph", app="DNS")
```

(continues on next page)

(continued from previous page)

```
ERROR:root:OPH: non-existant-oph not found
'OPH: non-existent-oph not found'

# Specific methods can also be directly called
>>> b1oph.get_app_state(name="my-oph-name", app="CDC")
'stopped'
>>> b1oph.get_app_state(name="my-oph-name", app="CDC")
'stopped'
>>> b1oph.manage_app(name="my-oph-name", app="CDC", action="disable")
True
>>> b1oph.get_app_state(name="my-oph-name", app="CDC")
'disabled - stopped'
>>> b1oph.enable_app(name="my-oph-name", app="CDC")
True
>>> b1oph.disable_app(name="my-oph-name", app="CDC")
True

>>> b1oph.manage_app(name="my-oph-name", app="CDC", action="blah")
ERROR:root:Action: blah not supported
False
```

5.5.9 b1platform Usage

The `b1platform` provides access to unsupported/undocumented plaform specific API functions. These should therefore, only be used at the users own risk. It should also be noted that these could change at any time.

5.5.9.1 Examples

Todo: These are simple examples to show you usage of the class. More comprehensive documentation is on the todo.

```
from pprint import pprint
import bloxone

# Instantiate class with ini file as argument
platform = bloxone.b1platform('<path to ini>')

# Get current user details
repsonse = platform.get_current_user()

# Get account membership for current user
response = platform.get_current_user_accounts()

# Get current tenant name
name = platform.get_current_tenant()

# Get list of users
response = platform.get_users()

# Retrieve the audit log
audit_log = platform.auditlog()

# Audit user accounts (uses domain of current user)
```

(continues on next page)

(continued from previous page)

```
list_of_non_complient_users = platform.audit_users()

# Audit user accounts (provide list of domains)
list = platform.audit_users(domains=['infoblox.com', 'mydomain.com'])
```

5.5.10 Additional Utilities

The `utils` sub-module contains a set of data Utilities for data validation and normalisation. Typically used for IoCs when using the *`bloxone.b1td`* methods for searches against TIDE.

For performance purposes when handling bulk data, the `data_type()`, `validate_fqdn()` and `validata_url()` functions use pre-compliled regexes that are created using the `buildregex()` function, that returns host and url regexes as a tuple.

These can then be passed to the appropriate data function.

For example:

```
import bloxone
host_regex, url_regex = bloxone.utils.buildregex()
qdata = "my.host.name.com"
data_type = bloxone.utils.data_type(qdata, host_regex, url_regex)
# Result = 'host'
qdata = "http://my.host.name.com"
data_type = bloxone.utils.data_type(qdata, host_regex, url_regex)
# Result = 'url'
```

The remaining classes generally provide generic interfaces for *get*, *create*, *update* and *delete*. Usage follows the same format of instantiating the class with an ini file and accessing the generic methods using using the 'swagger' path for the appropriate object.

CHAPTER 6

Classes Documentation

6.1 bloxone.b1 Class

class bloxone.**b1**(*cfg_file='config.ini'*)

Parent Class to simplify access to the BloxOne APIs for subclasses Can also be used to genericallly access the API

Raises

- *IniFileSectionError*
- *IniFileKeyError*
- *APIKeyFormatError*
- FileNotFoundError

create(*url*, *body=''*)

Generic create object wrapper

Parameters

- **url** (*str*) – Full URL
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

delete(*url*, *id=''*, *body=''*)

Generic delete object wrapper

Parameters

- **url** (*str*) – Full URL
- **id** (*str*) – Object id to delete
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

get(*url*, *id=''*, *action=''*, ***params*)
Generic get object wrapper

Parameters

- **url** (*str*) – Full URL
- **id** (*str*) – Optional Object ID
- **action** (*str*) – Optional object action, e.g. "nextavailableip"

Returns Requests response object

Return type response object

post(*url*, *id=''*, *action=''*, *body=''*, ***params*)
Generic Post object wrapper

Parameters

- **url** (*str*) – Full URL
- **id** (*str*) – Optional Object ID
- **action** (*str*) – Optional object action, e.g. "nextavailableip"

Returns Requests response object

Return type response object

replace(*url*, *id=''*, *body=''*)
Generic create object wrapper

Parameters

- **url** (*str*) – Full URL
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

update(*url*, *id=''*, *body=''*)
Generic create object wrapper

Parameters

- **url** (*str*) – Full URL
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

class bloxone.**IniFileSectionError**
Exception for missing section in ini file

class bloxone.**IniFileKeyError**
Exception for missing key in ini file

class bloxone.**APIKeyFormatError**
Exception for API key format mismatch

6.2 b1anycast Class

class bloxone.**b1anycast**(*cfg_file='config.ini'*)
Class to simplify access to the BloxOne Platform APIs

create(*objpath*, *body=''*)
Generic create object wrapper for platform objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

delete(*objpath*, *id=''*)
Generic delete object wrapper for platform objects

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Object id to delete

Returns Requests response object

Return type response object

get(*objpath*, *id=''*, *action=''*, ***params*)
Generic get object wrapper for platform calls

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Optional Object ID
- **action** (*str*) – Optional object action, e.g. "nextavailableip"

Returns Requests response object

Return type response object

get_id(*objpath*, ***, *key=''*, *value=''*, *include_path=False*)
Get object id using key/value pair

Parameters

- **objpath** (*str*) – Swagger object path
- **key** (*str*) – name of key to match
- **value** (*str*) – value to match
- **include_path** (*bool*) – Include path to object id

Returns object id or ""

Return type id (str)

update(*objpath*, *id=''*, *body=''*)
Generic create object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

6.3 b1authn Class

class bloxone.**b1authn**(*cfg_file='config.ini'*)

Class to simplify access to the BloxOne Platform APIs

create(*objpath*, *body=''*)

Generic create object wrapper for platform objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

delete(*objpath*, *id=''*)

Generic delete object wrapper for platform objects

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Object id to delete

Returns Requests response object

Return type response object

get(*objpath*, *id=''*, *action=''*, ***params*)

Generic get object wrapper for platform calls

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Optional Object ID
- **action** (*str*) – Optional object action, e.g. "nextavailableip"

Returns Requests response object

Return type response object

get_id(*objpath*, ***, *key=''*, *value=''*, *include_path=False*)

Get object id using key/value pair

Parameters

- **objpath** (*str*) – Swagger object path
- **key** (*str*) – name of key to match
- **value** (*str*) – value to match
- **include_path** (*bool*) – Include path to object id

Returns object id or “”

Return type id (str)

update(*objpath*, *id=”*, *body=”*)
Generic create object wrapper for platform objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

6.4 b1bootstrap Class

class bloxone.**b1bootstrap**(*cfg_file='config.ini'*)
Class to simplify access to the BloxOne Platform APIs

create(*objpath*, *body=”*)
Generic create object wrapper for platform objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

delete(*objpath*, *id=”*)
Generic delete object wrapper for platform objects

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Object id to delete

Returns Requests response object

Return type response object

get(*objpath*, *id=”*, *action=”*, ***params*)
Generic get object wrapper for platform calls

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Optional Object ID
- **action** (*str*) – Optional object action, e.g. “nextavailableip”

Returns Requests response object

Return type response object

get_id(*objpath*, ***, *key=”*, *value=”*, *include_path=False*)
Get object id using key/value pair

Parameters

- **objpath** (*str*) – Swagger object path
- **key** (*str*) – name of key to match
- **value** (*str*) – value to match
- **include_path** (*bool*) – Include path to object id

Returns object id or “”

Return type id (str)

update(*objpath*, *id=”*, *body=”*)

Generic create object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

6.5 b1cdc Class

class bloxone.**b1cdc**(*cfg_file='config.ini'*)

BloxOne DDI API Wrapper Class

add_tag(*objpath*, *id*, *tagname=”*, *tagvalue=”*)

Method to add a tag to an existing object Note: PUT/update Not Implemented in API as yet

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Object ID
- **tagname** (*str*) – Name of tag to add
- **tagvalue** (*str*) – Value to associate with tag

Returns Requests response object

Return type response object

create(*objpath*, *body=”*)

Generic create object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

delete(*objpath*, *id=”*)

Generic delete object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Object id to delete

Returns Requests response object

Return type response object

delete_tag(*objpath*, *id=''*, *tagname=''*)
Method to delete a tag from an existing On Prem Host

Parameters

- **objpath** (*str*) – Swagger object path
- **tagname** (*str*) – Name of tag to add

Returns Requests response object

Return type response object

get(*objpath*, *id=''*, *action=''*, ***params*)
Generic get object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Optional Object ID
- **action** (*str*) – Optional object action, e.g. "nextavailableip"

Returns Requests response object

Return type response object

get_id(*objpath*, ***, *key=''*, *value=''*, *include_path=False*)
Get object id using key/value pair

Parameters

- **objpath** (*str*) – Swagger object path
- **key** (*str*) – name of key to match
- **value** (*str*) – value to match
- **include_path** (*bool*) – Include path to object id

Returns object id or ""

Return type id (str)

get_object_by_key(*objpath*, ***, *key=''*, *value=''*, *include_path=False*)
Get object using key/value pair

Parameters

- **objpath** (*str*) – Swagger object path
- **key** (*str*) – name of key to match
- **value** (*str*) – value to match

Returns object id or ""

Return type id (str)

get_tags(*objpath*, *id=''*)
Get tags for an object id

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – id of object

Returns

Dictionary of current tags or empty dict if none

Return type tags (dict)

Todo:

- make generic, however, this requires the below
- Lookup dictionary of 'required fields' per object type

---

replace(*objpath*, *id=''*, *body=''*)
Generic create object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

search_response(*response*, *key=''*, *value=''*, *include_path=False*)
Get object id using key/value pair by searching a Request response object.

Parameters

- **object** (*response*) – Request response obj
- **key** (*str*) – name of key to match
- **value** (*str*) – value to match
- **include_path** (*bool*) – Include path to object id

Returns object id or ""

Return type id (str)

update(*objpath*, *id=''*, *body=''*)
Generic create object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

6.6 b1ddi Class

class bloxone.**b1ddi**(*cfg_file='config.ini'*)
BloxOne DDI API Wrapper Class

add_tag(*objpath*, *id*, *tagname=''*, *tagvalue=''*)
Method to add a tag to an existing object Note: PUT/update Not Implemented in API as yet

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Object ID
- **tagname** (*str*) – Name of tag to add
- **tagvalue** (*str*) – Value to associate with tag

Returns Requests response object

Return type response object

create(*objpath*, *body=''*)
Generic create object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

delete(*objpath*, *id=''*)
Generic delete object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Object id to delete

Returns Requests response object

Return type response object

delete_tag(*objpath*, *id=''*, *tagname=''*)
Method to delete a tag from an existing On Prem Host

Parameters

- **objpath** (*str*) – Swagger object path
- **tagname** (*str*) – Name of tag to add

Returns Requests response object

Return type response object

get(*objpath*, *id=''*, *action=''*, ***params*)
Generic get object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path

- **id** (*str*) – Optional Object ID
- **action** (*str*) – Optional object action, e.g. "nextavailableip"

Returns Requests response object

Return type response object

get_id(*objpath*, *, *key=''*, *value=''*, *include_path=False*)
Get object id using key/value pair

Parameters

- **objpath** (*str*) – Swagger object path
- **key** (*str*) – name of key to match
- **value** (*str*) – value to match
- **include_path** (*bool*) – Include path to object id

Returns object id or ""

Return type id (str)

get_object_by_key(*objpath*, *, *key=''*, *value=''*, *include_path=False*)
Get object using key/value pair

Parameters

- **objpath** (*str*) – Swagger object path
- **key** (*str*) – name of key to match
- **value** (*str*) – value to match

Returns object id or ""

Return type id (str)

get_option_ids(*option_space=''*)
Return a dictionary of DHCP Option IDs Based on idea/code from John Neerdael

Parameters **option_space** (*str*) – Optional Option Space ID

Returns Dictionary keyed on option number of ids

Return type option_ids (dict)

get_tags(*objpath*, *id=''*)
Get tags for an object id

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – id of object

Returns

Dictionary of current tags or empty dict if none

Return type tags (dict)

Todo:

- make generic, however, this requires the below
- Lookup dictionary of 'required fields' per object type

get_zone_child(*parent='zone'*, *name=''*, *id=''*, *fields=''*)
Method to retrieve Zones in specified view

Parameters

- **name** (*str*) – BloxOne object id
- **id** (*str*) – BloxOne object id
- ****params** (*dict*) – Generic API parameters

Returns Requests response object

Return type response object

post(*objpath*, *id=''*, *action=''*, *body=''*, ***params*)
Generic POST object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Optional Object ID
- **action** (*str*) – Optional object action, e.g. "nextavailableip"
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

replace(*objpath*, *id=''*, *body=''*)
Generic create object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

search_response(*response*, *key=''*, *value=''*, *include_path=False*)
Get object id using key/value pair by searching a Request response object.

Parameters

- **object** (*response*) – Request response obj
- **key** (*str*) – name of key to match
- **value** (*str*) – value to match
- **include_path** (*bool*) – Include path to object id

Returns object id or ""

Return type id (str)

update(*objpath*, *id=''*, *body=''*)
Generic create object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path

- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

6.7 b1diagnostics Class

class bloxone.**b1diagnostics**(*cfg_file='config.ini'*)

Class to simplify access to the BloxOne Platform APIs

delete(*objpath*, *id=''*)

Generic delete object wrapper for platform objects

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Object id to delete

Returns Requests response object

Return type response object

download_task_results(*taskid*)

Get the results for specidied task

Parameters **taskid** (*str*) – id of executed task

Returns Requests response object if id_only=False

Return type response object

Note:

execute_task(*command*, *args={}*, *ophname=None*, *ophid=None*, *id_only=True*, *priv=False*)

Execute remote command on an OPH

Parameters

- **cmd** (*str*) – Command to execute
- **args** (*dict*) – Command arguments
- **ophname** (*str*) – Name of OPH to execute command on (or supply ophid)
- **ophid** (*str*) – (Optional) ophid of OPH for cmd execution
- **id_only** (*bool*) – default of True
- **priv** (*bool*) – Run privileged task, default of False

Returns id string of task if id_only=True (defult) response object: Requests response object if id_only=False

Raises

- TypeError Exception if required options not supplied
- KeyErro Exception if ophname is not found (and ophid not supplied)
- Command_Not_Supported Exception if command is not valid
- Unknown_Argument Exception if arguments do not match required

Todo: [ ] Enhance logic to run /priviledgetask or /task Awaiting API enhancement to determine priv versus non-priv [ ] Enhance args check for required arguments Awaiting API enhancement for arg to determine required versus optional arguments

get(*objpath*, *id=''*, *action=''*, ***params*)
Generic get object wrapper for platform calls

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Optional Object ID
- **action** (*str*) – Optional object action, e.g. "nextavailableip"

Returns Requests response object

Return type response object

get_args(*command*)
Check the args for a command

Parameters **command** (*str*) – Command to retrieve argyments for

Returns Disctionary of arguments or empty dictionary if none.

Raises Command_Not_Supported Exception if command is not available

get_id(*objpath*, ***, *key=''*, *value=''*, *include_path=False*)
Get object id using key/value pair

Parameters

- **objpath** (*str*) – Swagger object path
- **key** (*str*) – name of key to match
- **value** (*str*) – value to match
- **include_path** (*bool*) – Include path to object id

Returns object id or ""

Return type id (str)

get_remote_commands()
Get set of possible remote commands and parameters

Returns Requests response object

Return type response object

get_task_result(*taskid*)
Get the results for specidied task

Parameters **taskid** (*str*) – id of executed task

Returns Requests response object if id_only=False

Return type response object

is_command(*command*)
Check whether command is valid

Parameters **command** (*str*) – command to check

Returns boolean

post (*objpath*, *body=''*)
Generic create object wrapper for platform objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

update (*objpath*, *id=''*, *body=''*)
Generic create object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

6.8 b1oph Class

class bloxone.**b1oph** (*cfg_file='config.ini'*)
Class to simplify access to the BloxOne Platform APIs

app_process_control (*name=''*, *app=''*, *action=''*)
Start or stop an application process

Parameters

- **name** (*str*) – display_name of OPH
- **app** (*str*) – App Name, e.g. DNS

Returns bool

create (*objpath*, *body=''*)
Generic create object wrapper for platform objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

delete (*objpath*, *id=''*)
Generic delete object wrapper for platform objects

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Object id to delete

Returns Requests response object

Return type response object

disable_app(*name=''*, *app=''*)
Disable specified app on named OPH

Parameters

- **name** (*str*) – display_name of OPH
- **app** (*str*) – App Name, e.g. DNS

Returns bool

enable_app(*name=''*, *app=''*)
Enable specified app on named OPH

Parameters

- **name** (*str*) – display_name of OPH
- **app** (*str*) – App Name, e.g. DNS

Returns bool

get(*objpath*, *id=''*, *action=''*, ***params*)
Generic get object wrapper for platform calls

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Optional Object ID
- **action** (*str*) – Optional object action, e.g. "nextavailableip"

Returns Requests response object

Return type response object

get_app_state(*name*, *app*)
Get status of application for an OPH

Parameters

- **name** (*str*) – display_name of OPH
- **app** (*str*) – App Name, e.g. DNS

Returns Status or error msg as text

Return type app_status (str)

get_id(*objpath*, ***, *key=''*, *value=''*, *include_path=False*)
Get object id using key/value pair

Parameters

- **objpath** (*str*) – Swagger object path
- **key** (*str*) – name of key to match
- **value** (*str*) – value to match
- **include_path** (*bool*) – Include path to object id

Returns object id or ""

Return type id (str)

get_ophid(*name=''*)
Return the ophid of named OPH

Parameters **name** (*str*) – display name of OPH

Returns ophid of the specified OPH

Return type ophid(str)

get_tags(*objpath*, *id=''*)
Get tags for an object id

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – id of object

Returns

Dictionary of current tags or empty dict if none

Return type tags (dict)

Todo:

- make generic, however, this requires the below
- Lookup dictionary of 'required fields' per object type

---

manage_app(*name=''*, *app=''*, *action='status'*)
Perform action on named OPH for specified app

Parameters

- **name** (*str*) – display_name of OPH
- **app** (*str*) – App Name, e.g. DNS
- **action** (*str*) – action to perform for app

Returns bool

on_prem_hosts(***params*)
Method to retrieve On Prem Hosts (undocumented)

Parameters ****params** (*dict*) – Generic API parameters

Returns Requests response object

Return type response object

oph_add_tag(*id=''*, *tagname=''*, *tagvalue=''*)
Method to add a tag to an existing On Prem Host

Parameters

- **objpath** (*str*) – Swagger object path
- **tagname** (*str*) – Name of tag to add
- **tagvalue** (*str*) – Value to associate

Returns Requests response object

Return type response object

oph_app_status(*oph_data*)
Translate App status info in JSON data for an OPH

Parameters **oph_data** (*dict*) – Data for individual OPH

Returns Dict of translated status elements

Return type oph_apps

oph_delete_tag(*id=''*, *tagname=''*)
Method to delete a tag from an existing On Prem Host

Parameters

- **objpath** (*str*) – Swagger object path
- **tagname** (*str*) – Name of tag to add

Returns Requests response object

Return type response object

oph_status(*oph_data*)
Translate status info in JSON data for an OPH

Parameters **oph_data** (*dict*) – Data for individual OPH

Returns Dict of translated status elements

Return type oph_status

oph_status_rpt(*json_data*, *tags=False*)
Build status report from GET /on_prem_hosts data

Parameters

- **json_data** (*json*) – API JSON data for On Prem Hosts call
- **tags** (*bool*) – Include tags in response, default False

Returns Dict of status elements

Return type rpt

oph_status_summary(*name=''*, *id=''*, *tags=False*)
Get OPH status information for one or more OPHs

Parameters

- **name** (*ste*) – Display name of OPH
- **id** (*str*) – id of a specific OPH
- **tags** (*bool*) – include tags in report

Returns Dict of translated status elements

Return type rpt

oph_uptime(*name=''*)

patch(*objpath*, *id=''*, *body=''*)
Generic create object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

update(*objpath*, *id=''*, *body=''*)
Generic create object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

6.9 b1platform Class

class bloxone.**b1platform**(*cfg_file='config.ini'*)
Class now reused for BloxOne Platform Methods, e.g. Audit Log Management of BloxOne On Prem Hosts is via the b1oph Class b1oph class is inherited here for compatibility

audit_users(*domains=[]*)
Audit User Data for non compliant email domains

Parameters **domain** (*list*) – List of valid email domains

Returns List of User Data (json)

auditlog(***params*)
Get the audit log

Parameters ****params** (*dict*) – Generic API parameters

Returns audit_log (list); list of dict

get_current_tenant(***params*)
Get name of current tenant

Parameters ****params** (*dict*) – Generic API parameters

Returns string containing name of tenant or '' on failure

get_current_user(***params*)
Get Current User Data

Parameters ****params** (*dict*) – Generic API parameters

Returns Requests response object

Return type response object

get_current_user_accounts(***params*)
Get Current Users Accounts Data

Parameters ****params** (*dict*) – Generic API parameters

Returns Requests response object

Return type response object

get_full_auditlog(***params*)

get_users(***params*)
Get User Data

Parameters ****params** (*dict*) – Generic API parameters

Returns Requests response object

Return type response object

6.10 b1dw Class

class bloxone.**b1sw**(*cfg_file='config.ini'*)
Class to simplify access to the BloxOne Platform APIs

create(*objpath*, *body=''*)
Generic create object wrapper for platform objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

delete(*objpath*, *id=''*)
Generic delete object wrapper for platform objects

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Object id to delete

Returns Requests response object

Return type response object

get(*objpath*, *id=''*, *action=''*, ***params*)
Generic get object wrapper for platform calls

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Optional Object ID
- **action** (*str*) – Optional object action, e.g. "nextavailableip"

Returns Requests response object

Return type response object

get_id(*objpath*, ***, *key=''*, *value=''*, *include_path=False*)
Get object id using key/value pair

Parameters

- **objpath** (*str*) – Swagger object path
- **key** (*str*) – name of key to match
- **value** (*str*) – value to match

- **include_path** (*bool*) – Include path to object id

Returns object id or “”

Return type id (str)

update(*objpath*, *id=”*, *body=”*)
Generic create object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

6.11 b1td Class

class bloxone.**b1td**(*cfg_file='config.ini'*)
BloxOne ThreatDefence API Wrapper Covers TIDE and Dossier

default_ttl()

dossier_sources()
Get Sources for Dossier

Returns Requests response object

Return type response object

dossier_target_sources(*type='host'*)
Get supported target types for Dossier

Parameters **type** (*str*) – target type

Returns Request response object

Return type response object

dossier_target_types()
Get supported target types for Dossier

Returns Request response object

Return type response object

dossierquery(*query*, *type='host'*, *sources='all'*, *wait=True*)
Simple Dossier Query

Parameters

- **query** (*str or list*) – single query or list of same type
- **type** (*str*) – “host”, “ip” or “url”
- **sources** (*str*) – set of sources or “all”

Returns Requests response object

Return type response object

expand_mitre_vector(*mitre*)
Expand MITRE Vector details

Parameters **mitre** (*str*) – MITRE Vector

Returns Requests response object

Return type response object

get(*objpath*, ***params*)
Generic get object wrapper for TIDE data objects

Parameters

- **objpath** (*str*) – Swagger object path
- **action** (*str*) – Optional object action

Returns Requests response object

Return type response object

historical_threat_counts()
Query Infoblox TIDE for historical threat counts

Returns Requests response object

Return type response object

post(*objpath*, *body=''*)
Generic create object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

querytide(*datatype*, *query*, ***params*)
Query Infoblox TIDE for all avaialble threat data related to query.

Parameters

- **datatype** (*str*) – "host", "ip" or "url"
- **query** (*str*) – query data

Returns Requests response object

Return type response object

querytideactive(*datatype*, *query*, ***params*)
Query Infoblox TIDE for "active" threat data i.e. threat data that has not expired at time of call

Parameters

- **datatype** (*str*) – "host", "ip" or "url"
- **query** (*str*) – query data

Returns Requests response object

Return type response object

querytidestate(*datatype*, *query*, ***params*)
Query Infoblox TIDE State Tables for specific query

Parameters

- **datatype** (*str*) – "host", "ip" or "url"
- **query** (*str*) – query data

Returns Requests response object

Return type response object

threat_actor(*name*)
Get Threat Actor details

Parameters **name** (*str*) – Name of Threat Actor

Returns Requests response object

Return type response object

threat_classes(***params*)
Get list of threat classes

Parameters:

Returns Requests response object

Return type response object

threat_counts()
Query Infoblox TIDE for active threat counts

Returns Requests response object

Return type response object

threat_properties(*threatclass=''*, ***params*)
Get list of threat properties

Parameters **threatclass** (*str*) – Threat Class

Returns Requests response object

Return type response object

tideactivefeed(*datatype*, *profile=''*, *threatclass=''*, *threatproperty=''*, ***params*)
Bulk "active" threat intel download from Infoblox TIDE state tables for specified datatype.

Parameters

- **datatype** (*str*) – "host", "ip" or "url"
- **profile** (*str, optional*) – Data provider
- **threatclass** (*str, optional*) – tide data class
- **threatproperty** (*str, optional*) – tide data property

Returns Requests response object

Return type response object

tidedatafeed(*datatype*, *profile=''*, *threatclass=''*, *threatproperty=''*, ***params*)
Bulk threat intel download from Infoblox TIDE for specified datatype. Please use wisely.

Parameters

- **datatype** (*str*) – "host", "ip" or "url"
- **profile** (*str, optional*) – Data provider
- **threatclass** (*str, optional*) – tide data class

- **threatproperty** (*str, optional*) – tide data property

Returns Requests response object

Return type response object

6.12 b1tdc Class

class bloxone.**b1tdc**(*cfg_file='config.ini'*)

BloxOne ThreatDefence Cloud API Wrapper

create(*objpath*, *body=''*)

Generic create object wrapper for Threat Defense Cloud

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

delete(*objpath*, *id=''*, *body=''*)

Generic delete object wrapper for Threat Defense Cloud

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Object id to delete
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

get(*objpath*, *action=''*, ***params*)

Generic get object wrapper for Threat Defense Cloud

Parameters

- **objpath** (*str*) – Swagger object path
- **action** (*str*) – Optional object action

Returns Requests response object

Return type response object

get_id(*objpath*, ***, *key=''*, *value=''*, *include_path=False*)

Get object id using key/value pair

Parameters

- **objpath** (*str*) – Swagger object path
- **key** (*str*) – name of key to match
- **value** (*str*) – value to match

Returns object id or ""

Return type id (str)

get_object_by_key(*objpath*, ***, *key=''*, *value=''*, *include_path=False*)
Get object using key/value pair

Parameters

- **objpath** (*str*) – Swagger object path
- **key** (*str*) – name of key to match
- **value** (*str*) – value to match

Returns object id or ""

Return type id (str)

post(*objpath*, *body=''*)
Generic create object wrapper for Threat Defense Cloud

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

put(*objpath*, *id=''*, *body=''*)
Generic put object wrapper for Threat Defense Cloud

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Object id to update
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

update(*objpath*, *id=''*, *body=''*)
Generic create object wrapper for Threat Defense Cloud

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

6.13 b1tdep Class

class bloxone.**b1tdep**(*cfg_file='config.ini'*)

create(*objpath*, *body=''*)
Generic create object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

delete(*objpath*, *id=''*)
Generic delete object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Object id to delete

Returns Requests response object

Return type response object

get(*objpath*, *id=''*, *action=''*, ***params*)
Generic get object wrapper for Threat Defense objects

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Optional Object ID
- **action** (*str*) – Optional object action, e.g. "nextavailableip"

Returns Requests response object

Return type response object

get_id(*objpath*, ***, *key=''*, *value=''*, *include_path=False*)
Get object id using key/value pair

Parameters

- **objpath** (*str*) – Swagger object path
- **key** (*str*) – name of key to match
- **value** (*str*) – value to match

Returns object id or ""

Return type id (str)

get_object_by_key(*objpath*, ***, *key=''*, *value=''*, *include_path=False*)
Get object using key/value pair

Parameters

- **objpath** (*str*) – Swagger object path
- **key** (*str*) – name of key to match
- **value** (*str*) – value to match

Returns object id or ""

Return type id (str)

update(*objpath*, *id=''*, *body=''*)
Generic create object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

6.14 b1tddfp Class

class bloxone.**b1tddfp**(*cfg_file='config.ini'*)

get(*objpath*, *id=''*, *action=''*, ***params*)
Generic get object wrapper for Threat Defense objects

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Optional Object ID
- **action** (*str*) – Optional object action, e.g. "nextavailableip"

Returns Requests response object

Return type response object

get_id(*objpath*, ***, *key=''*, *value=''*, *include_path=False*)
Get object id using key/value pair

Parameters

- **objpath** (*str*) – Swagger object path
- **key** (*str*) – name of key to match
- **value** (*str*) – value to match

Returns object id or ""

Return type id (str)

get_object_by_key(*objpath*, ***, *key=''*, *value=''*, *include_path=False*)
Get object using key/value pair

Parameters

- **objpath** (*str*) – Swagger object path
- **key** (*str*) – name of key to match
- **value** (*str*) – value to match

Returns object id or ""

Return type id (str)

update(*objpath*, *id=''*, *body=''*)
Generic create object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

6.15 b1tdlad Class

class bloxone.**b1tdlad**(*cfg_file='config.ini'*)

get(*objpath*, ***params*)
Generic get object wrapper for Threat Defense objects

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Optional Object ID
- **action** (*str*) – Optional object action, e.g. "nextavailableip"

Returns Requests response object

Return type response object

6.16 b1ztp Class

class bloxone.**b1ztp**(*cfg_file='config.ini'*)
Class to simplify access to the BloxOne Platform APIs

create(*objpath*, *body=''*)
Generic create object wrapper for platform objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

delete(*objpath*, *id=''*)
Generic delete object wrapper for platform objects

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Object id to delete

Returns Requests response object

Return type response object

get(*objpath*, *id=''*, *action=''*, ***params*)
Generic get object wrapper for platform calls

Parameters

- **objpath** (*str*) – Swagger object path
- **id** (*str*) – Optional Object ID

- **action** (*str*) – Optional object action, e.g. "nextavailableip"

Returns Requests response object

Return type response object

get_id(*objpath*, *, *key=''*, *value=''*, *include_path=False*)
Get object id using key/value pair

Parameters

- **objpath** (*str*) – Swagger object path
- **key** (*str*) – name of key to match
- **value** (*str*) – value to match
- **include_path** (*bool*) – Include path to object id

Returns object id or ""

Return type id (str)

update(*objpath*, *id=''*, *body=''*)
Generic create object wrapper for ddi objects

Parameters

- **objpath** (*str*) – Swagger object path
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

CHAPTER 7

DHCP Utilities

7.1 DHCP Encoding Class

class bloxone.**dhcp_encode**

Class to assist with Hex Encoding of DHCP Options and sub_options

binary_to_hex(*data*)

Format hex string of binary/hex encoded data

Parameters **data** (*str*) – data to format

Returns hex encoding as string

boolean_to_hex(*flag*)

Encode boolean value as single hex byte

Parameters **flag** (*bool/str*) – True or False as bool or text

Returns hex encoding as string

empty_to_hex(*data*)

Return empyt hex string ‘’

Parameters **data** (*str*) – Data not to encode (should be empty)

Returns Empty String ‘’

encode_data(*sub_opt*, *padding=False*, *pad_bytes=1*)

Encode the data section of a sub_option definition

Parameters

- **sub_opt** (*dict*) – Dict containing sub option details Must include ‘data’ and ‘type’ keys
- **padding** (*bool*) – Whether extra ‘null’ termination bytes are req.
- **pad_bytes** (*int*) – Number of null bytes to append

Returns Hex encoded data for specified data-type as string

encode_dhcp_option(*sub_opt_defs=[]*, *padding=False*, *pad_bytes=1*, *encapsulate=False*, *id=None*, *prefix=''*)
Encode list of DHCP Sub Options to Hex

Parameters

- **sub_opt_defs** (*list*) – List of Sub Option definition dictionaries
- **padding** (*bool*) – Whether extra 'null' termination bytes are req.
- **pad_bytes** (*int*) – Number of null bytes to append
- **encapsulate** (*bool*) – Add id and total length as prefix
- **id** (*int*) – option code to prepend
- **prefix** (*str*) – String value to prepend to encoded options

Returns Encoded suboption as a hex string

encode_sub_option(*sub_opt*, *data_only=False*, *padding=False*, *pad_bytes=1*)
Encode individual sub option

Parameters

- **sub_opt** (*dict*) – Sub Option Definition, as dict.
- **data_only** (*bool*) – Encode data portion only if True (Note the sub_opt dict is also checked for the 'data-only' key)
- **padding** (*bool*) – Whether extra 'null' termination bytes are req.
- **pad_bytes** (*int*) – Number of null bytes to append

Returns Encoded suboption as a hex string

fqdn_to_hex(*fqdn*)
Encode an fdqn in RFC 1035 Section 3.1 formatted hex

Parameters **fqdn** (*str*) – hostname to encode

Returns hex encoding as string

hex_length(*hex_string*)
Encode Option Length in hex (1-octet)

Parameters **hex_string** (*str*) – Octet Encoded Hex String

Returns Number of Hex Octects as hex encoded string

int_to_hex(*i*, *size=8*)
Encode integer of specified size as signed int in hex

Parameters

- **i** (*int*) – integer value to encode
- **size** (*int*) – size in bits [8, 16, 32]

Returns hex encoding as string

ip_to_hex(*ip*)
Encode an IPv4 or IPv6 address to hex

Parameters **ip** (*str*) – IPv4 or IPv6 address as a string

Returns hex encoding as string

ipv4_address_to_hex(*ipv4*)
Encode an IPv4 address to hex

Parameters **ipv4** (*str*) – IPv4 address as a string

Returns hex encoding as string

ipv6_address_to_hex(*ipv6*)
Encode an IPv6 address to hex

Parameters **ipv6** (*str*) – IPv4 or IPv6 address as a string

Returns hex encoding as string

optcode_to_hex(*optcode*)
Encode Option Code in hex (1-octet)

Parameters **optcode** (*str/int*) – Option Code

Returns hex encoding as string

string_to_hex(*string*)
Encode a text string to hex

Parameters **string** (*str*) – text string

Returns hex encoding as string

tests()
Run through encoding methods and output example results

uint_to_hex(*i*, *size=8*)
Encode integer of specified size as unsigned int in hex Uses 2's compliment if supplied with negative number

Parameters

- **i** (*int*) – integer value to encode
- **size** (*int*) – size in bits [8, 16, 32]

Returns hex encoding as string

validate_ip(*ip*)
Validate input data is a valid IP address (Supports both IPv4 and IPv6)

Parameters **ip** (*str*) – ip address as a string

Returns Return True for valid and False otherwise

Return type bool

7.2 Usage and Examples for dhcputils (encode)

The DHCP Utils provides classes to assist with the encoding and decoding of DHCP options in to/from hexidecimal. The primary use case is for sub-option encoding for custom option spaces, in particular, Option 43 encoding.

7.2.1 Encoding Class

The *bloxone.dhcp_encode()* class provides a set of *encode* methods to enable the encoding of defined DHCP sub options and data.

To encode the specific data-types a set of *type_to_hex* methods are provided. These will take a specific input data=type and apply the required encoding methodology to generate a compliant hexidecimal string for

7.2.2 Usage

The core aim of the class is to provide a simpler interface to encode DHCP sub options defined as a dictionary and provide an encoding.

To ensure flexibility the encoding is broken down in to three methods: `encode_dhcp_option()`, `encode_sub_option()` and `encode_data()`.

Sub-options are defined as a dictionary, with three required keys:

'code' the sub option code

'type' the data-type

'data' the data to encode using the specified type

And several optional keys:

'array' Boolean to indicate whether the data should be encoded as an array, e.g. array of IP addresses

'data-only' Boolen to indicate that only the data in the sub option should be encoded. i.e. do not add the encoded option code and length

For example:

```
sub1 = { 'code': 1, 'type': 'string', 'data': 'ABCDEFG'}
sub2 = { 'code': 2, 'type': 'ipv4_address',
         'data': '10.10.10.10,10.11.11.11', 'array': True }
```

Since there is typically more that one sub-option required, these can be added to a list that is processed, and encoded, by the `encode_dhcp_option()` method:

```
options = [ sub1, sub2 ]
```

An example, with example output is shown below:

```
import bloxone

de = bloxone.dhcp_encode()

# Set up the sub-options
sub1 = { 'code': 1, 'type': 'string', 'data': 'ABCDEFG'}
sub2 = { 'code': 2, 'type': 'ipv4_address',
         'data': '10.10.10.10,10.11.11.11', 'array': True }

# Create list for option to be encoded together
options = [ sub1, sub2 ]

# Encode e.g. for Option 43
h = de.encode_dhcp_option(options)
print(h)

'''
>>> print(h)
0107414243444546470208 0a0a0a0a0a0b0b0b
'''
```

(continues on next page)

(continued from previous page)

```
# Show the encoding for just one sub-option
print(de.encode_sub_option(sub2))

'''
>>> print(de.encode_sub_option(sub2))
02080a0a0a0a0a0b0b0b
'''

test_data = { 'code': '101', 'type': 'fqdn', 'data': 'www.example.com' }
hex_data = encode_data(test_data)

'''
>>> print(de.encode_data(test_data))
037777770765786 16d706c6503636f6d00
'''
```

Return a list of supported data types:

```
>>> import bloxone
>>> de = bloxone.dhcp_encode()
>>> de.opt_types
['string', 'ipv4_address', 'ipv6_address', 'boolean', 'int8', 'uint8',
 'int16', 'uint16', 'int32', 'uint32', 'fqdn', 'binary', 'empty']
```

Each of the supported data-types has a specific method of the format *type_to_hex()*. These can be directly access and typically support data both in its native format and as a string:

```
de.string_to_hex('Hello World')
de.ip4_address_to_hex('192.168.1.101')
de.fqdn_to_hex('www.infoblox.com')
de.int8_to_hex('22') or int8_to_hex(22)

etc
```

Specific functions to return the length of the hexidecimal string in hex and encoding of the option code (basically a wrapper of int8), are also provided:

```
h = de.ip4_address_to_hex('10.10.10.10')
de.hex_length(h)
de.optcode_to_hex(125)
```

A `tests()` method is also provided that will show example encodings for each data-type and option encodings:

```
>>>de.tests()
Encoding types supported: ['string', 'ipv4_address', 'ipv6_address', 'boolean', 'int8
↪', 'uint8', 'int16', 'uint16', 'int32', 'uint32', 'fqdn', 'binary', 'empty']

Non-array tests:
Type: string: AABBDDCCEEDD-aabbccddeeff, Encoded:␣
↪414142424444434345454444 2d6161626263636465656666, Length(hex): 19
Type: ipv4_address: 10.10.10.10, Encoded: 0a0a0a0a, Length(hex): 04
Type: ipv4_address: 10.10.10.10,11.11.11.11, Encoded: 0a0a0a0a0b0b0b0b, Length(hex):␣
↪08
Type: boolean: True, Encoded: 01, Length(hex): 01
Type: int8: 22, Encoded: 16, Length(hex): 01
```

(continues on next page)

(continued from previous page)

```
Type: int8: -22, Encoded: 96, Length(hex): 01
Type: uint8: 22, Encoded: 116, Length(hex): 01
Type: int16: 33, Encoded: 0021, Length(hex): 02
Type: int16: 33, Encoded: 0021, Length(hex): 02
Type: uint16: 33, Encoded: 10021, Length(hex): 02
Type: int32: 44, Encoded: 0000002c, Length(hex): 04
Type: uint32: -44, Encoded: fffffffd4, Length(hex): 04
Type: uint32: 44, Encoded: 10000002c, Length(hex): 04
Type: fqdn: www.infoblox.com, Encoded: 0377777708696e666f626c6f7803636f6d00,␣
↪Length(hex): 12
Type: binary: DEADBEEF, Encoded: deadbeef, Length(hex): 04
Type: empty: , Encoded: , Length(hex): 00
Type: ipv6_address: 2001:DB8::1, Encoded: 20010db8000000000000000000000001,␣
↪Length(hex): 10
Type: ipv6_address: 2001:DB8::1,2001:DB8::2, Encoded:␣
↪20010db8000000000000000000000120010db8000000000000000000000002, Length(hex): 20

Padding test (1 byte), type string: AABBCCDD 414142424343444400
Full encoding of sample:␣
↪011941414242444443434545444442d6161626263636464656566660204 0a0a0a0a03080a0a0a0a0b0b0b0b040101050116
>>>
```

7.3 DHCP Decoding Class

class bloxone.**dhcp_decode**

Class to assist with Hex Encoding of DHCP Options and sub_options

check_data_type(*optcode*, *sub_defs=[]*)

Get data_type for optcode from sub optino definitions

Parameters

- **optcode** (*int*) – Option code to check
- **sub_defs** (*list of dict*) – sub option definitions to cross reference

Returns data_type as str

decode_data(*data*, *data_type='string'*, *padding=False*, *pad_bytes=1*, *array=False*)

decode_dhcp_option(*hex_string*, *sub_opt_defs=[]*, *padding=False*, *pad_bytes=1*, *encapsulated=False*, *id=None*, *prefix=''*)

Attempt to decode DHCP options from hex representation

Parameters

- **sub_opt_defs** (*list*) – List of Sub Option definition dictionaries
- **padding** (*bool*) – Whether extra 'null' termination bytes are req.
- **pad_bytes** (*int*) – Number of null bytes to append
- **encapsulate** (*bool*) – Add id and total length as prefix
- **id** (*int*) – option code to prepend
- **prefix** (*str*) – String value to prepend to encoded options

Returns Encoded suboption as a hex string

get_name (*optcode*, *sub_defs=[]*)
Get data_type for optcode from sub optino definitions

Parameters

- **optcode** (*int*) – Option code to check
- **sub_defs** (*list of dict*) – sub option definitions to cross reference

Returns name as str

guess_data_type (*subopt*, *padding=False*)

hex_length (*hex_string*)
Encode Option Length in hex (1-octet)

Parameters **hex_string** (*str*) – Octet Encoded Hex String

Returns Number of Hex Octects as hex encoded string

hex_string_to_list (*hex_string*)
Take a hex string and convert in to a list

Parameters **hex_string** (*str*) – Hex represented as string

Returns list of hex bytes

hex_to_array_of_ip (*hex_string*)
Decode array of IPv4 or IPv6 addresses to CSV string

Parameters **hex_string** (*str*) – Hex representation of an array of IPv4 or IPv6

Returns IP Addresses in a CSV string

hex_to_binary (*data*)
Format hex string of binary/hex encoded data

Parameters **data** (*str*) – data to format

Returns hex encoding as string

hex_to_boolean (*hex_string*)
Decode Hex value as a string to 'true' or 'false'

Parameters **hex_string** (*str*) – Hex value as a str

Returns string representation of a boolean

hex_to_empty (*data*)
Return empyt hex string ''

Parameters **data** (*str*) – Data not to encode (should be empty)

Returns Empty String ''

hex_to_fqdn (*hex_string*)
Decode RFC 1035 Section 3.1 formatted hexa to fqdn

Parameters **hex_string** (*str*) – hex encoded fqdn

Returns fqdn as string

hex_to_int (*hex_string*, *size=8*)
Decode hex to signed integer of defined size

Parameters

- **hex_string** (*str*) – hex value as string

- **size** (*int*) – size in bits [8, 16, 32]

Returns integer

hex_to_ip(*hex_string*)
Decode a 4 or 16 octect hex string to an IPv4 or IPv6 string

Parameters **hex_string** (*str*) – Hex representation of an IPv4 or IPv6 address

Returns IP Address as a string

hex_to_ipv4_address(*hex_string*)
Decode a hex string to an IPv4 Address as a string

Parameters **hex_string** (*str*) – Hex representation of an IPv4 address

Returns IPv4 Address as a string

hex_to_ipv6_address(*hex_string*)
Decode a hex string to an IPv6 address as a string

Parameters **hex_string** (*str*) – Hex representation of an IPv6 address

Returns IPv6 Address as a string

hex_to_optcode(*hex_string*)
Encode Option Code in hex (1-octet)

Parameters **optcode** (*str/int*) – Option Code

Returns hex encoding as string

hex_to_string(*hex_string*)
Decode a string of hex values to a text string

Parameters **hex_string** (*str*) – Hex representation of a string

Returns text string (str)

hex_to_suboptions(*hex_string*, *encapsulated=False*)
Extract the sub-options from the hex data

hex_to_uint(*hex_string*, *size=8*)
Encode integer of specified size as unsigned int in hex Uses 2's compliment if supplied with negative number

Parameters

- **i** (*int*) – integer value to encode
- **size** (*int*) – size in bits [8, 16, 32]

Returns hex encoding as string

output_decoded_options(*decoded_opts=[]*, *output='pprint'*)
Simple output for decode_dhcp_options() data

Parameters

- **decoded_opts** (*list*) – List of dict
- **output** (*str*) – specify format [pprint, csv, yaml]

tests()
Run through encoding methods and output example results

validate_ip(*ip*)
Validate input data is a valid IP address (Supports both IPv4 and IPv6)

Parameters **ip** (*str*) – ip address as a string

Returns Return True for valid and False otherwise

Return type bool

7.4 Usage and Examples for dhcputils (decode)

The DHCP Utils provides classes to assist with the encoding and decoding of DHCP options in to/from hexidecimal. The primary use case is for sub-option encoding for custom option spaces, in particular, Option 43 encoding.

7.4.1 Decoding Class

The *bloxone.dhcp_decode()* class provides a set of *decode* methods to attempt to decode a hexideciaml string representation of DHCP sub-options.

To decode specific data-types a set of *hex_to_type* methods are provided. These take a hexideciaml string representation of the data and apply the required decoding methodology and return the decoded data.

Since and encoded set of DHCP sub-options does not include the data type accurate decoding is problematic. Two mechanisms are provided to try and address this:

```
- It is possible to provide a known definition for the sub-options and
use the specified data types.

- Allow the class to make a best-efforts *guess* of the data_type.
```

In both cases a string decoding is also applied and provided back. A flag, *guess* is used to inform whether or not the data_type was *guessed* or based on a known definition.

The `DHCP_OPTION_DEFS` class can be utilised to extract sub-option definitions from a YAML configuration and maintain a vendor database of option definitions.

7.4.2 Usage

The core aim of the class is to provide a simpler interface to decode DHCP sub options and return this as a list of dictionary definitions detailing the decoding.

To ensure flexibility the decoding is broken down in to two methods: `decode_dhcp_option()` and `decode_data()`.

`decode_data()` takes the hexideciaml string and a data_type and will apply the appropriate hex_to_type() decoding method. For example:

```
import bloxone
de = bloxone.dhcp_decode()
de.decode_data('0a0a0a0a', data_type='ipv4_address')
# Output: '10.10.10.10'

# Note this can be shortened to
# de.decode_data('0a0a0a0a', data_type='ip')
```

`decode_dhcp_option()` also takes a hexidecimal string, however, in this case it will be enterpreted as an encoded set of sub-options (based on option 43 encoding). As mentioned due to the fact that this does not contain data type information, you can either provide a dictionary containing an option definition (of the same format used by the

dhcp_encode() class), or let the function attempt to make a guess. In both cases the string decoding will be included in the response.

The method returns a list of decoded sub-options with a dictionary per sub option. This will be of the format:

```
 [ {'name': 'Undefined',
    'code': 3,
    'data': '10.10.10.10,11.11.11.11',
'data_length': 8,
    'data_str': '\n\n\n\n\x0b\x0b\x0b\x0b',
    'guess': True,
    'type': 'array_of_ip'} ]
```

For simple output an output_decoded_options() method is provided.

Examples:

```
import bloxone
de = bloxone.dhcp_decode()

# Hex string to attempt to decode
h = '011941414242444443434544442d6161626263636464656566602040a0a0a0a' +
        '03080a0a0a0a0b0b0b0b040101050116'

opt_list = de.decode_dhcp_option(h)
de.output_decoded_options(opt_list)
```

This will produce the output:

```
[{'code': 1,
'data': 'AABBDDCCEEDD-aabbccddeeff',
'data_length': 25,
'data_str': 'AABBDDCCEEDD-aabbccddeeff',
'guess': True,
'name': 'Undefined',
'type': 'string'},
{'code': 2,
'data': '10.10.10.10',
'data_length': 4,
'data_str': '\n\n\n\n',
'guess': True,
'name': 'Undefined',
'type': 'ip'},
{'code': 3,
'data': '10.10.10.10,11.11.11.11',
'data_length': 8,
'data_str': '\n\n\n\n\x0b\x0b\x0b\x0b',
'guess': True,
'name': 'Undefined',
'type': 'array_of_ip'},
{'code': 4,
'data': 1,
'data_length': 1,
'data_str': '\x01',
'guess': True,
'name': 'Undefined',
'type': 'int8'},
{'code': 5,
'data': 22,
```

(continues on next page)

(continued from previous page)

```
'data_length': 1,
'data_str': '\x16',
'guess': True,
'name': 'Undefined',
'type': 'int8'}]
```

Example providing sub-option definitions using the same hex string:

```
# Set up the sub-option definitions
sub1 = { 'name': 'Test1', 'code': 1, 'type': 'string', 'data': ''}
sub2 = { 'name': 'Test2', 'code': 2, 'type': 'ipv4_address',
                'data': '', 'array': False }
sub3 = { 'name': 'Test3', 'code': 3, 'type': 'ipv4_address',
                'data': '', 'array': True }
sub4 = { 'name': 'Test4', 'code': 4, 'type': 'boolean' }
sub5 = { 'name': 'Test5', 'code': 5, 'type': 'int8' }

# Create list of option definitions
options = [ sub1, sub2, sub3, sub4, sub5 ]

opt_list = de.decode_dhcp_option(h, sub_opt_defs=options)
de.output_decoded_options(opt_list)
```

Here you can see that the name and data-types are defined from sub-option definitions:

```
[{'code': 1,
'data': 'AABBDDCCEEDD-aabbccddeeff',
'data_length': 25,
'data_str': 'AABBDDCCEEDD-aabbccddeeff',
'guess': False,
'name': 'Test1',
'type': 'string'},
{'code': 2,
'data': '10.10.10.10',
'data_length': 4,
'data_str': '\n\n\n\n',
'guess': False,
'name': 'Test2',
'type': 'ipv4_address'},
{'code': 3,
'data': '10.10.10.10,11.11.11.11',
'data_length': 8,
'data_str': '\n\n\n\n\x0b\x0b\x0b\x0b',
'guess': False,
'name': 'Test3',
'type': 'array_of_ip'},
{'code': 4,
'data': 'true',
'data_length': 1,
'data_str': '\x01',
'guess': False,
'name': 'Test4',
'type': 'boolean'},
{'code': 5,
'data': 22,
'data_length': 1,
'data_str': '\x16',
```

(continues on next page)

(continued from previous page)

```
'guess': False,
'name': 'Test5',
'type': 'int8'}]
```

As mentioned the `DHCP_OPTION_DEFS` class can be utilised to access vendor DHCP Option definitions from a YAML configuration an example script and example vendor configuration file can be found as part of the dhcp_option_encoding project on GitHub.

A simple example using showing this is shown below:

```
h = '010c4d532d55432d436c69656e7402056874747073031965707 36c' +
        '796e6330312e657073696c6f6e68712e6c6f63616c040334343305' +
        '252f4365727450726f762f4365727450726f766973696f6e696e67' +
        '536572766963652e737663'
v = bloxone.DHCP_OPTION_DEFS('vendor_dict.yaml')
sub_options = v.sub_options('MS-UC-Client')

opt_list = de.decode_dhcp_option(h, sub_opt_defs=sub_options)
de.output_decoded_options(opt_list)
```

As with the `dhcp_encode()` class you can get a list of supported decoding data types using the *opt_types** attribute:

```
>>> import bloxone
>>> de = bloxone.dhcp_decode()
>>> de.opt_types
['string', 'ip', 'array_of_ip', 'ipv4_address', 'ipv6_address', 'boolean',
'int8', 'uint8', 'int16', 'uint16', 'int32', 'uint32', 'fqdn', 'binary',
'empty']
```

For decoding purposes the generic *ip* and *array_of_ip* types are exposed, the respective methods support both IPv4 and IPv6.

Each of the supported data-types has a specific method of the format *hex_to_type()*. These can be directly access and typically support data both in its native format and as a string:

```
de.hex_to_string('48656c6c6f20776f726c64')
# 'Hello world'
de.hex_to_ip('c0a80165')
# '192.168.1.101'
de.hex_to_fqdn('0377777708696e666f626c6f7803636f6d00')
# 'www.infoblox.com.'
de.hex_to_int8('16')
# 22

etc
```

A `tests()` method is also provided that will show example encodings/decodings for each data-type and option encodings.

7.5 DHCP Option Data Class

class bloxone.**DHCP_OPTION_DEFS**(*cfg='vendor_dict.yaml'*)

Class to load and handle DHCP Option Defs

count()

Get numbner of defined vendors

Returns int

dump_vendor_def(*vendor*)
Returns the vendor definition as a dict

Parameters **vendor** (*str*) – Vendor Identifier

Returns dict containing vendor definition

included(*vendor*)
Check whether this vendor is configured

Parameters **vendor** (*str*) – Vendor Identifier

Returns bool

keys()

Returns list of top level keys

option_def(*vendor*)
Returns option definition as dict

Parameters **vendor** (*str*) – Vendor Identifier

Returns Dict containing both parent and sub option definitions

parent_opt_def(*vendor*)
Returns parent-option definition as dict

Parameters **vendor** (*str*) – Vendor Identifier

Returns dict containing parent option definition

sub_options(*vendor*)
Returns list of sub-option definitions

Parameters **vendor** (*str*) – Vendor Identifier

Returns list of dict

vendor_description(*vendor*)
Get description of vendor

Parameters **vendor** (*str*) – Vendor Identifier

vendor_keys(*vendor*)
Returns vendor top level keys

Parameters **vendor** (*str*) – Vendor Identifier

Returns list of keys defined for a vendor

vendor_prefix(*vendor*)
Get the prefix is present as a string

Parameters **vendor** (*str*) – Vendor Identifier

Returns string containing defined prefix or '' if none

vendors()

Returns list of defined vendors

version()

Returns str containing config file version or 'Version not defined'

7.6 DHCP Options Data Class Usage

To simplify the definition and encoding/decoding of DHCP Options with the *bloxone.dhcp_encode()* class, a data class to handle Vendor definitions is included.

The *bloxone.DHCP_OPTION_DEFS()* class provides a simple interface to read the vendor definition from a YAML file that can contain one or more vendor definitions.

7.6.1 YAML File Format

The base file definition allows for a file version number and a list of vendors, a sample showing all of the options is shown below:

```
---
# DHCP Vendor Option Definitions
version: 0.0.1

vendors:

    Sample-Vendor:
        vci: sample-vci
        description: My Vendor Class
        prefix: "<prefix str if required>"
        option-def:
            parent-option:
                name: option name
                code: 43
                type: binary
                array: False
            sub-options:
                - name: Sub Opt 1
                  code: 1
                  type: string
                  data: Encode this string
                  array: False
                  data-only: False
                - name: Sub Opt 2
                  code: 5
                  type: ipv4_address
                  data: 10.10.10.10,20.20.20.20
                  array: True
                  data-only: False
```

The format allows the complete definition of a vendor, with the core element being the *option-def* that defines, in particular, the list of sub-options for encoding.

The definition can include a prefix to prepend to the encoding, and data-only flags to handle both option 43 style encodings and option 125 style encodings.

Example Definitions:

```
---
# DHCP Vendor Option Definitions
version: 0.0.1

vendors:
```

(continues on next page)

(continued from previous page)

```
MS-UC-Client:
    vci: MS-UC-Client
    description: Microsoft Lync Client
    option-def:
        parent-option:
            name: option 43
            code: 43
            type: binary
            array: False
        sub-options:
            - name: UC Identifier
              code: 1
              type: string
              data: MS-UC-Client
              array: False
            - name: URL Scheme
              code: 2
              type: string
              data: https
              array: False
            - name: Web Server FQDN
              code: 3
              type: string
              data: epslync01.epsilonhq.local
              array: False
            - name: Web Server Port
              code: 4
              type: string
              data: 443
              array: False
            - name: Certificate Web Service
              code: 5
              type: string
              data: /CertProv/CertProvisioningService.svc
              array: False


####### CISCO
# Option 43 sub-option 241

Cisco AP:
    vci: Cisco AP
    description: Cisco Aironet Series APs
    option-def:
        parent-option:
            name: option 43
            code: 43
            type: binary
            array: False
        sub-options:
            - name: Controller IP
              code: 241
              type: ipv4_address
              data: 10.150.1.15,10.150.50.15
              array: True
```

(continues on next page)

(continued from previous page)

```
    ####### MITEL

    Mitel:
        vci: Mitel
        description: Mitel Phone (prepend 00000403)
        prefix: "00000403"
        option-def:
            parent-option:
                name: option 125
                code: 125
                type: binary
                array: False
            sub-options:
                - code: 125
                    type: string
                    data: id:ipphone.mitel.com;call_srv=X;vlan=X;dscp=46;l2p=X;sw_
↪tftp=X
                    data-only: True
```

The *bloxone.DHCP_OPTION_DEFS()* class allows you to specify the YAML file to read, and enables you to read the elements quickly and easily. As an example, the file above can be used to encode the vendors sub-options:

```
import bloxone
de = bloxone.dhcp_encode()
vendors = bloxone.DHCP_OPTION_DEFS('vendor_dict.yaml')
if vendors.included('MS-US-Client'):
    print(de.encode_dhcp_option(vendors.sub_options('MS-US-Client')))

'''
Vendor: MS-UC-Client, Encoding: 010c4d532d55432d436c69656e7402056874
747073031965707 36c796e6330312e657073696c6f6e68712e6c6f63616c04033434
3305252f4365727450726f762f4365727450726f766973696f6e696e675365727669
63652e737663
'''

if vendors.included('Mitel'):
    print(de.encode_dhcp_option(vendors.sub_options('Mitel')))
'''
Vendor: Mitel, Encoding: 0000040369643a697070686f6e652e6d6974656c2e636f
6d3b63616c6c5f7372763d583b766c616e3d583b647363703d34363b6c32703d583b737
75f746674703d58
'''

# Process all vendors:
for vendor in vendors.vendors():
    print(vendor)
    print(de.encode_dhcp_option(vendors.sub_options('Mitel')))
```

An example script using both classes to perform encodings from a CLI can be found on github https://github.com/ccmarris/dhcp_option_encoding

CHAPTER 8

Additional Utilities

Description: Simple utility functions for data type validation, domain handling, and data normalisationa specifically with the aim of supporting queries to TIDE and Dossier.

Requirements: Python3 with re, ipaddress, requests

Author: Chris Marrison

Date Last Updated: 20210714

Todo:

bloxone.utils.**buildregex**()
Pre-compile 'standard' regexes as used by data_type and validate_XXX functions

Parameters **none** –

Returns Compiled regex for hostnames url_regex (re): Compiled regex for URLs

Return type host_regex (re)

bloxone.utils.**convert_url_to_host**(*url*)
Break down URL and return host element

Parameters **url** (*str*) – Validated URL

Returns hostname or ip

Return type host (str)

bloxone.utils.**count_labels**(*fqdn*)
Count number of labels in an FQDN

Parameters **fqdn** (*str*) – Hostname as fqdn

Returns number of labels

Return type count (int)

bloxone.utils.**data_type**(*qdata*, *host_regex*, *url_regex*)
Validate and determine data type (host, ip or url)

Parameters

- **qdata** (*str*) – data to determine type/validity
- **host_regex/url_regex** (*re*) – pre-compiled regexes

Returns data type of qdata ("ip", "host", or "url")

Return type dtype (str)

bloxone.utils.**db_query**(*db_cursor*, *table*, *query_type*, *query_data*, **flags*)
Perform db query and return appropriate rows

Parameters

- **db_cursor** (*obj*) – db.cursor object
- **table** (*str*) – database table name
- **query_type** (*str*) – data type for query
- **query_data** (*str*) – search string

Returns (All matching db rows

Return type rows (list)

bloxone.utils.**get_domain**(*fqdn*, *no_of_labels=2*)
Take FQDN and return n label domain or fqdn if no. of labels is 2 or less

Parameters

- **fqdn** (*str*) – Hostname as fqdn
- **no_of_labels** (*int*) – Number of labels to return default = 2

Returns N label domain name or fqdn

Return type domain (str)

bloxone.utils.**get_table**(*cursor*)
Determine db table and return

Parameters **cursor** (*obj*) – db.cursor object

Returns Table name as string

Return type table (str)

bloxone.utils.**normalise**(*item*, *itype=None*, *port=False*, *www=False*)
Take ip, host or url item process and return normalise data.

Parameters

- **item** (*str*) – item to normalise
- **itype** (*str*) – One of ["host", "url", "ip"]
- **port** (*bool*) – stip port number e.g. :8080
- **www** (*bool*) – strip www. from hostname

Returns Normalised item or "invalid"

Return type normalised (str)

bloxone.utils.**opendb**(*dbfile*)
Open sqlite db and return cursor()

Parameters **dbfile** (*str*) – path to file

Returns Returns a db.cursor()

Return type cursor (obj)

bloxone.utils.**reverse_labels**(*domain*)
Reserve order of domain labels (or any dot separated data, e.g. IP)

Parameters **domain** (*str*) – domain.labels

Returns labels.domain

Return type rdomain (str)

bloxone.utils.**strip_host**(*fqdn*)
Take FQDN and strip first label or fqdn if no. of labels is 2 or less

Parameters **fqdn** (*str*) – Hostname as fqdn

Returns stripped domain down to two labels

Return type domain (str)

bloxone.utils.**validate_fqdn**(*hostname*, *regex*)
Validate input data is a legitmate fqdn

Parameters

- **hostname** (*str*) – fqdn as a string
- **regex** (*obj*) – Compiled regex for hostnames

Returns Return True for valid and False otherwise

Return type bool

bloxone.utils.**validate_ip**(*ip*)
Validate input data is a valid IP address

Parameters **ip** (*str*) – ip address as a string

Returns Return True for valid and False otherwise

Return type bool

bloxone.utils.**validate_url**(*url*, *regex*)
Validate input data is a valid URL

Parameters

- **url** (*str*) – string to verify as URL
- **regex** (*re*) – pre-compiled regex obj

Returns Return True for valid and False otherwise

Return type bool

CHAPTER 9

Source Documentation

https://python-bloxone.readthedocs.io/en/latest/

exception bloxone.bloxone.**APIKeyFormatError**
Exception for API key format mismatch

exception bloxone.bloxone.**IniFileKeyError**
Exception for missing key in ini file

exception bloxone.bloxone.**IniFileSectionError**
Exception for missing section in ini file

class bloxone.bloxone.**b1**(*cfg_file='config.ini'*)
Parent Class to simplify access to the BloxOne APIs for subclasses Can also be used to genericallly access the API

Raises

- *IniFileSectionError*
- *IniFileKeyError*
- *APIKeyFormatError*
- FileNotFoundError

create(*url*, *body=''*)
Generic create object wrapper

Parameters

- **url** (*str*) – Full URL
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

delete(*url*, *id=''*, *body=''*)
Generic delete object wrapper

Parameters

- **url** (*str*) – Full URL
- **id** (*str*) – Object id to delete
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

get(*url*, *id=''*, *action=''*, ***params*)
Generic get object wrapper

Parameters

- **url** (*str*) – Full URL
- **id** (*str*) – Optional Object ID
- **action** (*str*) – Optional object action, e.g. "nextavailableip"

Returns Requests response object

Return type response object

post(*url*, *id=''*, *action=''*, *body=''*, ***params*)
Generic Post object wrapper

Parameters

- **url** (*str*) – Full URL
- **id** (*str*) – Optional Object ID
- **action** (*str*) – Optional object action, e.g. "nextavailableip"

Returns Requests response object

Return type response object

replace(*url*, *id=''*, *body=''*)
Generic create object wrapper

Parameters

- **url** (*str*) – Full URL
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

update(*url*, *id=''*, *body=''*)
Generic create object wrapper

Parameters

- **url** (*str*) – Full URL
- **body** (*str*) – JSON formatted data payload

Returns Requests response object

Return type response object

bloxone.bloxone.**read_b1_ini**(*ini_filename*)
Open and parse ini file

Parameters **ini_filename** (*str*) – name of inifile

Returns Dictionary of BloxOne configuration elements

Return type config (dict)

Raises

- *IniFileSectionError*
- *IniFileKeyError*
- *APIKeyFormatError*
- FileNotFoundError

bloxone.bloxone.**verify_api_key**(*apikey*)
Verify format of API Key

Parameters **apikey** (*str*) – api key

Returns True is apikey passes format validation

Return type bool

CHAPTER 10

ChangeLog

20211104 v0.8.4 App control for OPHs
2021102 v0.8.3 Added OPH status reporting methods
20210805 v0.8.2 Improved error protection
20210805 v0.8.1 Added requirements.txt for yaml
20210804 v0.8.0 Added output csv and yaml output
20210804 v0.7.9 Minor bug fixes to dhcputils.py
20210803 v0.7.9 Updates to b1ddi to support 'actions'
20210730 v0.7.8 Minor bugs in new dhcp_decode class
20210730 v0.7.8 dhcp_decode class
20210726 v0.7.6 Documentation Updated for dhcp option classes
20210723 v0.7.6 Prefix handling in dhcputils, minor bug fixes
20210718 v0.7.5 dhcp_utils.py with near complete coding class
20210713 v0.7.4 Methods and docs for b1diagnostics class
20210709 v0.7.3 Added get_ophid() method to b1oph class
20210621 v0.7.2 Fixed issue with get_zone_child() method in b1ddi
20210618 v0.7.1 Framwork for b1diagnostics and b1notifications classes
20210308 v0.7.0 Added api_key format verification, raising exception
20210308 v0.6.9 Added exception raising for ini file not found
20210221 v0.6.8 Created 'public' generic methods for get, create, i
replace, update:wq
20210215 v0.6.7 Added New Platform classes for new API elements
b1anycast, b1authn, b1bootstrap, b1oph, b1sw, b1ztp
Deprecating b1platform class (inherits b1oph for
compatibility
20210212 v0.6.6 Added b1cdc class
20201105 v0.6.5 Minor bug fixes
20201105 v0.6.2 Added get_all_auditlog() method
20201105 v0.6.1 Dossier & threat enrichment methods added

20201102 v0.6.0 Fixed the add_tag and delete_tag methods
20201022 v0.5.9 Added auditlog method to b1platform
20200907 v0.5.8 Fixed a regex warning in utils.buildregex()
20200904 v0.5.7 Added get_option_ids helper method.
20200821 v0.5.5 Changes to project_urls for packaging
20200821 v0.5.4 Fixed fact that documentation wasn't included in
    package.
20200818 v0.53 Streamlined get_id using _filter
20200818 v0.5.1 Initial Classes for Threat Defence DFP
20200818 v0.5.0 Initial Classes for Threat Defence (EP, Cloud, LAD)
20200817 v0.4.1 Added tag manipulation to b1ddi
20200810 v0.4.0 Minor changes and improved documentatin
20200810 v0.3.9 Fixed bug in b1td.get method
20200807 v0.3.8 Created b1td class for TIDE API b1td.py
20200714 v0.3.0 Added specific add/delete tags for on_prem_hosts
20200714 v0.2.4 Added create and update methods to b1platform class
20200713 v0.2.1 Renamed patch to update as originally intended
20200713 v0.2.0 Removed original methods for get_object
20200713 v0.1.5 Added get_object_by_key and create initial documentation
20200711 v0.1.2 Added generic wrappers for DDI for create and delete
    Added get_id method to get object id from key/value pair
20200710 v0.1.1 Generic Wrapper and restructuring
20200708 v0.0.5 Read only examples for several b1 objects
        Commit before restructuring to a more generic
    wrappers and useful functions
20200701 v0.0.2 Subclass for b1ddi api methods added to b1
20200629 v0.0.1 Initial Class commit for b1 class
    Base class attributes and ini file
    handling.

CHAPTER 11

Authors and acknowledgment

Author: Chris Marrison Email: chris@infoblox.com

Thanks to Geoff Horne for his input in the early stages of this project, and for letting me undertake it.

Thanks to Krishna Vasudeva for testing and accelerating the addition of some of the classes with her contributions.

Thanks to John Neerdael, for some helper method ideas for b1ddi.

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index

E

F

G

H

I

K

M

N

O

P

Q

R

S

T

U

V