PyMISP Documentation

Release main

Raphaël Vinot

Nov 30, 2021

CONTENTS

Contents:

IMPORTANT NOTE: This library will require **at least** python 3.8 starting the 1st of January 2022. If you have legacy versions of python, please use the latest PyMISP version that will be released in December 2021, and consider updating your system(s). Anything released within the last 2 years will do, starting with Ubuntu 20.04.

CHAPTER

ONE

PYMISP - PYTHON LIBRARY TO ACCESS MISP

PyMISP is a Python library to access MISP platforms via their REST API.

PyMISP allows you to fetch events, add or update events/attributes, add or update samples or search for attributes.

1.1 Install from pip

It is strongly recommended to use a virtual environment

If you want to know more about virtual environments, python has you covered

Only basic dependencies:

```
pip3 install pymisp
```

And there are a few optional dependencies:

- fileobjects: to create PE/ELF/Mach-o objects. **Important**: it will install pydeep, which require the system package `libfuzzy-dev`
- openioc: to import files in OpenIOC format (not really maintained).
- virustotal: to query VirusTotal and generate the appropriate objects
- docs: to generate te documentation
- pdfexport: to generate PDF reports out of MISP events
- url: to generate URL objects out of URLs with Pyfaup
- email: to generate MISP Email objects
- brotli: to use the brotli when interacting with a MISP instance

Example:

```
pip3 install pymisp[virustotal,email]
```

1.2 Install the latest version from repo from development purposes

Note: poetry is required; e.g., "pip3 install poetry"

```
git clone https://github.com/MISP/PyMISP.git && cd PyMISP
git submodule update --init
poetry install -E fileobjects -E openioc -E virustotal -E docs -E pdfexport
```

1.2.1 Running the tests

```
poetry run nosetests-3.4 --with-coverage --cover-package=pymisp,tests --cover-tests␣
↪tests/test_*.py
```

If you have a MISP instance to test against, you can also run the live ones:

Note: You need to update the key in `tests/testlive_comprehensive.py` to the automation key of your admin account.

```
poetry run nosetests-3.4 --with-coverage --cover-package=pymisp,tests --cover-tests␣
↪tests/testlive_comprehensive.py
```

1.3 Samples and how to use PyMISP

Various examples and samples scripts are in the examples/ directory.

In the examples directory, you will need to change the keys.py.sample to enter your MISP url and API key.

```
cd examples
cp keys.py.sample keys.py
vim keys.py
```

The API key of MISP is available in the Automation section of the MISP web interface.

To test if your URL and API keys are correct, you can test with examples/last.py to fetch the events published in the last x amount of time (supported time indicators: days (d), hours (h) and minutes (m)). last.py

```
cd examples
python3 last.py -l 10h # 10 hours
python3 last.py -l 5d  #  5 days
python3 last.py -l 45m # 45 minutes
```

1.4 Debugging

You have two options here:

1. Pass `debug=True` to `PyMISP` and it will enable logging.DEBUG to stderr on the whole module
2. Use the python logging module directly:

```
import logging
logger = logging.getLogger('pymisp')

# Configure it as you wish, for example, enable DEBUG mode:
logger.setLevel(logging.DEBUG)
```

Or if you want to write the debug output to a file instead of stderr:

```
import pymisp
import logging

logger = logging.getLogger('pymisp')
logging.basicConfig(level=logging.DEBUG, filename="debug.log", filemode='w',␣
↪format=pymisp.FORMAT)
```

1.5 Test cases

1. The content of `mispevent.py` is tested on every commit
2. The test cases that require a running MISP instance can be run the following way:

```
# From poetry

nosetests-3.4 -s --with-coverage --cover-package=pymisp,tests --cover-tests tests/
↪testlive_comprehensive.py:TestComprehensive.[test_name]
```

1.6 Documentation

The documentation is available here.

1.6.1 Jupyter notebook

A series of Jupyter notebooks for PyMISP tutorial are available in the repository.

1.7 Everything is a Mutable Mapping

... or at least everything that can be imported/exported from/to a json blob

`AbstractMISP` is the master class, and inherits from `collections.MutableMapping` which means the class can be represented as a python dictionary.

The abstraction assumes every property that should not be seen in the dictionary is prepended with a `_`, or its name is added to the private list `__not_jsonable` (accessible through `update_not_jsonable` and `set_not_jsonable`.

This master class has helpers that make it easy to load, and export to, and from, a json string.

`MISPEvent`, `MISPAttribute`, `MISPObjectReference`, `MISPObjectAttribute`, and `MISPObject` are subclasses of AbstractMISP, which mean that they can be handled as python dictionaries.

1.8 MISP Objects

Creating a new MISP object generator should be done using a pre-defined template and inherit `AbstractMISPObjectGenerator`.

Your new MISPObject generator must generate attributes and add them as class properties using `add_attribute`.

When the object is sent to MISP, all the class properties will be exported to the JSON export.

CHAPTER

TWO

LICENSE

PyMISP is distributed under an open source license. A simplified 2-BSD license.

CHAPTER

THREE

PYMISP - CLASSES

3.1 PyMISP

class pymisp.**PyMISP**(*url*, *key*, *ssl=True*, *debug=False*, *proxies={}*, *cert=None*, *auth=None*, *tool=''*, *timeout=None*)

Python API for MISP

Parameters

- **url** (str) – URL of the MISP instance you want to connect to
- **key** (str) – API key of the user you want to use
- **ssl** (bool) – can be True or False (to check or to not check the validity of the certificate. Or a CA_BUNDLE in case of self signed or other certificate (the concatenation of all the crt of the chain)
- **debug** (bool) – Write all the debug information to stderr
- **proxies** (Mapping) – Proxy dict, as described here: http://docs.python-requests.org/en/master/user/advanced/#proxies
- **cert** (Optional[Tuple[str, tuple]]) – Client certificate, as described here: http://docs.python-requests.org/en/master/user/advanced/#client-side-certificates
- **auth** (Optional[AuthBase]) – The auth parameter is passed directly to requests, as described here: http://docs.python-requests.org/en/master/user/authentication/
- **tool** (str) – The software using PyMISP (string), used to set a unique user-agent
- **timeout** (Union[float, Tuple[float, float], None]) – Timeout, as described here: https://requests.readthedocs.io/en/master/user/advanced/#timeouts

accept_attribute_proposal(*proposal*)

Accept a proposal. You cannot modify an existing proposal, only accept/discard

Parameters **proposal** (Union[*MISPShadowAttribute*, int, str, UUID]) – attribute proposal to accept

Return type Dict

accept_event_delegation(*delegation*, *pythonify=False*)

Accept the delegation of an event

Parameters

- **delegation** (Union[*MISPEventDelegation*, int, str]) – event delegation to accept
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Dict

accept_user_registration(*registration*, *organisation=None*, *role=None*, *perm_sync=False*, *perm_publish=False*, *perm_admin=False*, *unsafe_fallback=False*)

Accept a user registration

Parameters

- **registration** (Union[*MISPInbox*, int, str, UUID]) – the registration to accept
- **organisation** (Union[*MISPOrganisation*, int, str, UUID, None]) – user organization
- **role** (Union[*MISPRole*, int, str, None]) – user role
- **perm_sync** (bool) – indicator for sync
- **perm_publish** (bool) – indicator for publish
- **perm_admin** (bool) – indicator for admin
- **unsafe_fallback** (bool) – indicator for unsafe fallback

add_attribute(*event*, *attribute*, *pythonify=False*)

Add an attribute to an existing MISP event

Parameters

- **event** (Union[*MISPEvent*, int, str, UUID]) – event to extend
- **attribute** (Iterable) – attribute or (MISP version 2.4.113+) list of attributes to add. If a list is passed, the pythonified response is a dict with the following structure: {'attributes': [MISPAttribute], 'errors': {errors by attributes}}
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPAttribute*, *MISPShadowAttribute*]

add_attribute_proposal(*event*, *attribute*, *pythonify=False*)

Propose a new attribute in an event

Parameters

- **event** (Union[*MISPEvent*, int, str, UUID]) – event to receive new attribute
- **attribute** (*MISPAttribute*) – attribute to propose
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPShadowAttribute*]

add_correlation_exclusion(*correlation_exclusion*, *pythonify=False*)

Add a new correlation exclusion

Parameters

- **correlation_exclusion** (MISPCorrelationExclusion) – correlation exclusion to add
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, MISPCorrelationExclusion]

add_event(*event*, *pythonify=False*, *metadata=False*)

Add a new event on a MISP instance

Parameters

- **event** (*MISPEvent*) – event to add

- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output
- **metadata** (bool) – Return just event metadata after successful creating

Return type Union[Dict, *MISPEvent*]

add_event_blocklist(*uuids*, *comment=None*, *event_info=None*, *event_orgc=None*)
Add a new event in the blocklist

Parameters

- **uuids** (Union[str, List[str]]) – UUIDs
- **comment** (Optional[str]) – comment
- **event_info** (Optional[str]) – event information
- **event_orgc** (Optional[str]) – event organization

Return type Dict

add_event_report(*event*, *event_report*, *pythonify=False*)
Add an event report to an existing MISP event

Parameters

- **event** (Union[*MISPEvent*, int, str, UUID]) – event to extend
- **event_report** (MISPEventReport) – event report to add.
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, MISPEventReport]

add_feed(*feed*, *pythonify=False*)
Add a new feed on a MISP instance

Parameters

- **feed** (*MISPFeed*) – feed to add
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPFeed*]

add_galaxy_cluster(*galaxy*, *galaxy_cluster*, *pythonify=False*)
Add a new galaxy cluster to a MISP Galaxy

Parameters

- **galaxy** (Union[MISPGalaxy, str, UUID]) – A MISPGalaxy (or UUID) where you wish to add the galaxy cluster
- **galaxy_cluster** (MISPGalaxyCluster) – A MISPGalaxyCluster you wish to add
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, MISPGalaxyCluster]

add_galaxy_cluster_relation(*galaxy_cluster_relation*)
Add a galaxy cluster relation, cluster relation must include cluster UUIDs in both directions

Parameters **galaxy_cluster_relation** (MISPGalaxyClusterRelation) – The MISPGalaxyClusterRelation to add

Return type Dict

add_object(*event*, *misp_object*, *pythonify=False*, *break_on_duplicate=False*)
Add a MISP Object to an existing MISP event

Parameters

- **event** (Union[*MISPEvent*, int, str, UUID]) – event to extend
- **misp_object** (*MISPObject*) – object to add
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output
- **break_on_duplicate** (bool) – if True, check and reject if this object's attributes match an existing object's attributes; may require much time

Return type Union[Dict, *MISPObject*]

add_object_reference(*misp_object_reference*, *pythonify=False*)
Add a reference to an object

Parameters

- **misp_object_reference** (*MISPObjectReference*) – object reference
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPObjectReference*]

add_org_to_sharing_group(*sharing_group*, *organisation*, *extend=False*)
Add an organisation to a sharing group.

Parameters

- **sharing_group** (Union[*MISPSharingGroup*, int, str, UUID]) – Sharing group's local instance ID, or Sharing group's global UUID
- **organisation** (Union[*MISPOrganisation*, int, str, UUID]) – Organisation's local instance ID, or Organisation's global UUID, or Organisation's name as known to the curent instance
- **extend** (bool) – Allow the organisation to extend the group

Return type Dict

add_organisation(*organisation*, *pythonify=False*)
Add an organisation

Parameters

- **organisation** (*MISPOrganisation*) – organization to add
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPOrganisation*]

add_organisation_blocklist(*uuids*, *comment=None*, *org_name=None*)
Add a new organisation in the blocklist

Parameters

- **uuids** (Union[str, List[str]]) – UUIDs
- **comment** (Optional[str]) – comment
- **org_name** (Optional[str]) – organization name

Return type Dict

add_server(*server*, *pythonify=False*)
Add a server to synchronise with. Note: You probably want to use PyMISP.get_sync_config and PyMISP.import_server instead

Parameters

- **server** (*MISPServer*) – sync server config
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPServer*]

add_server_to_sharing_group(*sharing_group*, *server*, *all_orgs=False*)
Add a server to a sharing group.

Parameters

- **sharing_group** (Union[*MISPSharingGroup*, int, str, UUID]) – Sharing group's local instance ID, or Sharing group's global UUID
- **server** (Union[*MISPServer*, int, str, UUID]) – Server's local instance ID, or URL of the Server, or Server's name as known to the curent instance
- **all_orgs** (bool) – Add all the organisations of the server to the group

Return type Dict

add_sharing_group(*sharing_group*, *pythonify=False*)
Add a new sharing group

Parameters

- **sharing_group** (*MISPSharingGroup*) – sharing group to add
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPSharingGroup*]

add_sighting(*sighting*, *attribute=None*, *pythonify=False*)
Add a new sighting (globally, or to a specific attribute)

Parameters

- **sighting** (*MISPSighting*) – sighting to add
- **attribute** (Union[*MISPAttribute*, int, str, UUID, None]) – specific attribute to modify with the sighting
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPSighting*]

add_tag(*tag*, *pythonify=False*)
Add a new tag on a MISP instance. The user calling this method needs the Tag Editor permission. It doesn't add a tag to an event, simply creates it on the MISP instance.

Parameters

- **tag** (*MISPTag*) – tag to add
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPTag*]

add_user(*user*, *pythonify=False*)
Add a new user

Parameters

- **user** (*MISPUser*) – user to add
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPUser*]

attribute_exists(*attribute*)
Fast check if attribute exists.

Parameters **attribute** (Union[*MISPAttribute*, int, str, UUID]) – Attribute to check

Return type bool

attribute_proposals(*event=None*, *pythonify=False*)
Get all the attribute proposals

Parameters

- **event** (Union[*MISPEvent*, int, str, UUID, None]) – event
- **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPShadowAttribute*]]

attributes(*pythonify=False*)
Get all the attributes from the MISP instance

Parameters **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPAttribute*]]

attributes_statistics(*context='type'*, *percentage=False*)
Get attribute statistics from the MISP instance

Parameters

- **context** (str) – "type" or "category"
- **percentage** (bool) – get percentages

Return type Dict

build_complex_query(*or_parameters=None*, *and_parameters=None*, *not_parameters=None*)
Build a complex search query. MISP expects a dictionary with AND, OR and NOT keys.

Return type Dict[str, List[~SearchType]]

cache_all_feeds()
Cache all the feeds

Return type Dict

cache_feed(*feed*)
Cache a specific feed by id

Parameters **feed** (Union[*MISPFeed*, int, str, UUID]) – feed to cache

Return type Dict

cache_freetext_feeds()
Cache all the freetext feeds

Return type Dict

cache_misp_feeds()
Cache all the MISP feeds

Return type Dict

change_sharing_group_on_entity(*misp_entity*, *sharing_group_id*, *pythonify=False*)
Change the sharing group of an event, an attribute, or an object

Parameters

- **misp_entity** (Union[*MISPEvent*, *MISPAttribute*, *MISPObject*]) – entity to change
- **sharing_group_id** – group to change
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPEvent*, *MISPObject*, *MISPAttribute*, *MISPShadowAttribute*]

change_user_password(*new_password*)
Change the password of the curent user

Parameters **new_password** (str) – password to set

Return type Dict

clean_correlation_exclusions()
Initiate correlation exclusions cleanup

communities(*pythonify=False*)
Get all the communities

Parameters **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[MISPCommunity]]

compare_feeds()
Generate the comparison matrix for all the MISP feeds

Return type Dict

contact_event_reporter(*event*, *message*)
Send a message to the reporter of an event

Parameters

- **event** (Union[*MISPEvent*, int, str, UUID]) – event with reporter to contact
- **message** (str) – message to send

Return type Dict

correlation_exclusions(*pythonify=False*)
Get all the correlation exclusions

Parameters **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[MISPCorrelationExclusion]]

db_schema_diagnostic()
Get the schema diagnostic

Return type Dict

delegate_event(*event=None*, *organisation=None*, *event_delegation=None*, *distribution=- 1*, *message=''*, *pythonify=False*)
Delegate an event. Either event and organisation OR event_delegation are required

Parameters

- **event** (Union[*MISPEvent*, int, str, UUID, None]) – event to delegate
- **organisation** (Union[*MISPOrganisation*, int, str, UUID, None]) – organization
- **event_delegation** (Optional[*MISPEventDelegation*]) – event delegation
- **distribution** (int) – distribution == -1 means recipient decides
- **message** (str) – message
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPEventDelegation*]

delete_attribute(*attribute*, *hard=False*)
Delete an attribute from a MISP instance

Parameters

- **attribute** (Union[*MISPAttribute*, int, str, UUID]) – attribute to delete
- **hard** (bool) – flag for hard delete

Return type Dict

delete_attribute_proposal(*attribute*)
Propose the deletion of an attribute

Parameters **attribute** (Union[*MISPAttribute*, int, str, UUID]) – attribute to delete

Return type Dict

delete_correlation_exclusion(*correlation_exclusion*)
Delete a correlation exclusion

Parameters **correlation_exclusion** (Union[MISPCorrelationExclusion, int, str, UUID]) – The MISPCorrelationExclusion you wish to delete from MISP

Return type Dict

delete_event(*event*)
Delete an event from a MISP instance"'

Parameters **event** (Union[*MISPEvent*, int, str, UUID]) – event to delete

Return type Dict

delete_event_blocklist(*event_blocklist*)
Delete a blocklisted event by id

Parameters **event_blocklist** (Union[*MISPEventBlocklist*, str, UUID]) – event block list to delete

Return type Dict

delete_event_report(*event_report*, *hard=False*)
Delete an event report from a MISP instance

Parameters

- **event_report** (Union[MISPEventReport, int, str, UUID]) – event report to delete
- **hard** (bool) – flag for hard delete

Return type Dict

delete_feed(*feed*)
Delete a feed from a MISP instance

Parameters **feed** (Union[*MISPFeed*, int, str, UUID]) – feed to delete

Return type Dict

delete_galaxy_cluster(*galaxy_cluster*, *hard=False*)
Deletes a galaxy cluster from MISP

Parameters

- **galaxy_cluster** (Union[MISPGalaxyCluster, int, str, UUID]) – The MISPGalaxyCluster you wish to delete from MISP
- **hard** – flag for hard delete

Return type Dict

delete_galaxy_cluster_relation(*galaxy_cluster_relation*)
Delete a galaxy cluster relation

Parameters **galaxy_cluster_relation** (Union[MISPGalaxyClusterRelation, int, str, UUID]) – The MISPGalaxyClusterRelation to delete

Return type Dict

delete_object(*misp_object*, *hard=False*)
Delete an object from a MISP instance

Parameters

- **misp_object** (Union[*MISPObject*, int, str, UUID]) – object to delete
- **hard** (bool) – flag for hard delete

Return type Dict

delete_object_reference(*object_reference*)
Delete a reference to an object

Parameters **object_reference** (Union[*MISPObjectReference*, int, str, UUID]) – object reference

Return type Dict

delete_organisation(*organisation*)
Delete an organisation by id

Parameters **organisation** (Union[*MISPOrganisation*, int, str, UUID]) – organization to delete

Return type Dict

delete_organisation_blocklist(*organisation_blocklist*)
Delete a blocklisted organisation by id

Parameters **organisation_blocklist** (Union[*MISPOrganisationBlocklist*, str, UUID]) – organization block list to delete

Return type Dict

delete_server(*server*)
Delete a sync server

Parameters **server** (Union[*MISPServer*, int, str, UUID]) – sync server config

Return type Dict

delete_sharing_group(*sharing_group*)
Delete a sharing group

Parameters **sharing_group** (Union[*MISPSharingGroup*, int, str, UUID]) – sharing group to delete

Return type Dict

delete_sighting(*sighting*)
Delete a sighting from a MISP instance

Parameters **sighting** (Union[*MISPSighting*, int, str, UUID]) – sighting to delete

Return type Dict

delete_tag(*tag*)
Delete a tag from a MISP instance

Parameters **tag** (Union[*MISPTag*, int, str, UUID]) – tag to delete

Return type Dict

delete_user(*user*)
Delete a user by id

Parameters **user** (Union[*MISPUser*, int, str, UUID]) – user to delete

Return type Dict

delete_user_setting(*user_setting*, *user=None*)
Delete a user setting

Parameters

- **user_setting** (str) – name of user setting
- **user** (Union[*MISPUser*, int, str, UUID, None]) – user

Return type Dict

property describe_types_local: Dict
Returns the content of describe types from the package

Return type Dict

property describe_types_remote: Dict
Returns the content of describe types from the remote instance

Return type Dict

direct_call(*url*, *data=None*, *params={}*, *kw_params={}*)
Very lightweight call that posts a data blob (python dictionary or json string) on the URL

Parameters

- **url** (str) – URL to post to
- **data** (Optional[Dict]) – data to post
- **params** (Mapping) – dict with parameters for request
- **kw_params** (Mapping) – dict with keyword parameters for request

Return type Any

disable_feed(*feed*, *pythonify=False*)
Disable a feed

Parameters

- **feed** (Union[*MISPFeed*, int, str, UUID]) – feed to disable
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPFeed*]

disable_feed_cache(*feed*, *pythonify=False*)
Disable the caching of a feed

Parameters

- **feed** (Union[*MISPFeed*, int, str, UUID]) – feed to disable caching
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPFeed*]

disable_noticelist(*noticelist*)
Disable a noticelist by id

Parameters **noticelist** (Union[*MISPNoticelist*, int, str, UUID]) – Noticelist to disable

Return type Dict

disable_tag(*tag*, *pythonify=False*)
Disable a tag

Parameters

- **tag** (*MISPTag*) – tag to disable
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPTag*]

disable_taxonomy(*taxonomy*)
Disable a taxonomy.

Parameters **taxonomy** (Union[*MISPTaxonomy*, int, str, UUID]) – taxonomy to disable

Return type Dict

disable_taxonomy_tags(*taxonomy*)
Disable all the tags of a taxonomy

Parameters **taxonomy** (Union[*MISPTaxonomy*, int, str, UUID]) – taxonomy with tags to disable

Return type Dict

disable_warninglist(*warninglist*)
Disable a warninglist

Parameters **warninglist** (Union[*MISPWarninglist*, int, str, UUID]) – warninglist to disable

Return type Dict

discard_attribute_proposal(*proposal*)
Discard a proposal. You cannot modify an existing proposal, only accept/discard

Parameters **proposal** (Union[*MISPShadowAttribute*, int, str, UUID]) – attribute proposal to discard

Return type Dict

discard_event_delegation(*delegation*, *pythonify=False*)
Discard the delegation of an event

Parameters

- **delegation** (Union[*MISPEventDelegation*, int, str]) – event delegation to discard
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Dict

discard_user_registration(*registration*)
Discard a user registration

Parameters **registration** (Union[*MISPInbox*, int, str, UUID]) – the registration to discard

enable_feed(*feed*, *pythonify=False*)
Enable a feed; fetching it will create event(s)

Parameters

- **feed** (Union[*MISPFeed*, int, str, UUID]) – feed to enable
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPFeed*]

enable_feed_cache(*feed*, *pythonify=False*)
Enable the caching of a feed

Parameters

- **feed** (Union[*MISPFeed*, int, str, UUID]) – feed to enable caching
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPFeed*]

enable_noticelist(*noticelist*)
Enable a noticelist by id

Parameters **noticelist** (Union[*MISPNoticelist*, int, str, UUID]) – Noticelist to enable

Return type Dict

enable_tag(*tag*, *pythonify=False*)
Enable a tag

Parameters

- **tag** (*MISPTag*) – tag to enable
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPTag*]

enable_taxonomy(*taxonomy*)
Enable a taxonomy

Parameters **taxonomy** (Union[*MISPTaxonomy*, int, str, UUID]) – taxonomy to enable

Return type Dict

enable_taxonomy_tags(*taxonomy*)
Enable all the tags of a taxonomy. NOTE: this is automatically done when you call enable_taxonomy

Parameters **taxonomy** (Union[*MISPTaxonomy*, int, str, UUID]) – taxonomy with tags to enable

Return type Dict

enable_warninglist(*warninglist*)
Enable a warninglist

Parameters **warninglist** (Union[*MISPWarninglist*, int, str, UUID]) – warninglist to enable

Return type Dict

event_blocklists(*pythonify=False*)
Get all the blocklisted events

Parameters **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPEventBlocklist*]]

event_delegations(*pythonify=False*)
Get all the event delegations

Parameters **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPEventDelegation*]]

event_exists(*event*)
Fast check if event exists.

Parameters **event** (Union[*MISPEvent*, int, str, UUID]) – Event to check

Return type bool

events(*pythonify=False*)
Get all the events from the MISP instance

Parameters **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPEvent*]]

feeds(*pythonify=False*)
Get the list of existing feeds

Parameters **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPFeed*]]

fetch_feed(*feed*)
Fetch one single feed by id

Parameters **feed** (Union[*MISPFeed*, int, str, UUID]) – feed to fetch

Return type Dict

fork_galaxy_cluster(*galaxy*, *galaxy_cluster*, *pythonify=False*)
Forks an existing galaxy cluster, creating a new one with matching attributes

Parameters

- **galaxy** (Union[MISPGalaxy, int, str, UUID]) – The galaxy (or galaxy ID) where the cluster you want to fork resides
- **galaxy_cluster** (MISPGalaxyCluster) – The galaxy cluster you wish to fork
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, MISPGalaxyCluster]

freetext(*event*, *string*, *adhereToWarninglists=False*, *distribution=None*, *returnMetaAttributes=False*, *pythonify=False*, ***kwargs*)
Pass a text to the freetext importer

Parameters

- **event** (Union[*MISPEvent*, int, str, UUID]) – event
- **string** (str) – query
- **adhereToWarninglists** (Union[bool, str]) – flag
- **distribution** (Optional[int]) – distribution == -1 means recipient decides
- **returnMetaAttributes** (bool) – flag
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output
- **kwargs** – kwargs passed to prepare_request

Return type Union[Dict, List[*MISPAttribute*]]

galaxies(*pythonify=False*)
Get all the galaxies

Parameters **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[MISPGalaxy]]

get_all_functions(*not_implemented=False*)
Get all methods available via the API, including ones that are not implemented.

get_attribute(*attribute*, *pythonify=False*)
Get an attribute from a MISP instance

Parameters

- **attribute** (Union[*MISPAttribute*, int, str, UUID]) – attribute to get
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPAttribute*]

get_attribute_proposal(*proposal*, *pythonify=False*)
Get an attribute proposal

Parameters

- **proposal** (Union[*MISPShadowAttribute*, int, str, UUID]) – proposal to get
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPShadowAttribute*]

get_community(*community*, *pythonify=False*)
Get a community by id from a MISP instance

Parameters

- **community** (Union[MISPCommunity, int, str, UUID]) – community to get
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, MISPCommunity]

get_correlation_exclusion(*correlation_exclusion*, *pythonify=False*)
Get a correlation exclusion by ID

Parameters

- **correlation_exclusion** (Union[MISPCorrelationExclusion, int, str, UUID]) – Correlation exclusion to get
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, MISPCorrelationExclusion]

get_event(*event*, *deleted=False*, *extended=False*, *pythonify=False*)
Get an event from a MISP instance. Includes collections like Attribute, EventReport, Feed, Galaxy, Object, Tag, etc. so the response size may be large.

Parameters

- **event** (Union[*MISPEvent*, int, str, UUID]) – event to get
- **deleted** (Union[int, list]) – whether to include soft-deleted attributes
- **extended** (int) – whether to get extended events
- **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, *MISPEvent*]

get_event_report(*event_report*, *pythonify=False*)
Get an event report from a MISP instance

Parameters

- **event_report** (Union[MISPEventReport, int, str, UUID]) – event report to get
- **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, MISPEventReport]

get_event_reports(*event_id*, *pythonify=False*)
Get event report from a MISP instance that are attached to an event ID

Parameters

- **event_id** (Union[int, str]) – event id to get the event reports for
- **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output.

Return type Union[Dict, List[MISPEventReport]]

get_feed(*feed*, *pythonify=False*)
Get a feed by id

Parameters

- **feed** (Union[*MISPFeed*, int, str, UUID]) – feed to get
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPFeed*]

get_galaxy(*galaxy*, *withCluster=False*, *pythonify=False*)
Get a galaxy by id

Parameters

- **galaxy** (Union[MISPGalaxy, int, str, UUID]) – galaxy to get
- **withCluster** (bool) – Include the clusters associated with the galaxy
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, MISPGalaxy]

get_galaxy_cluster(*galaxy_cluster*, *pythonify=False*)
Gets a specific galaxy cluster

Parameters

- **galaxy_cluster** (Union[MISPGalaxyCluster, int, str, UUID]) – The MISPGalaxyCluster you want to get
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, MISPGalaxyCluster]

get_noticelist(*noticelist*, *pythonify=False*)
Get a noticelist by id

Parameters

- **notistlist** – Noticelist to get
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPNoticelist*]

get_object(*misp_object*, *pythonify=False*)
Get an object from the remote MISP instance

Parameters

- **misp_object** (Union[*MISPObject*, int, str, UUID]) – object to get
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPObject*]

get_object_template(*object_template*, *pythonify=False*)
Gets the full object template

Parameters

- **object_template** (Union[*MISPObjectTemplate*, int, str, UUID]) – template or ID to get
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPObjectTemplate*]

get_organisation(*organisation*, *pythonify=False*)
Get an organisation by id

Parameters

- **organisation** (Union[*MISPOrganisation*, int, str, UUID]) – organization to get
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPOrganisation*]

get_raw_object_template(*uuid_or_name*)
Get a row template. It needs to be present on disk on the MISP instance you're connected to. The response of this method can be passed to MISPObject(<name>, misp_objects_template_custom=<response>)

Return type Dict

get_server_setting(*setting*)
Get a setting from the MISP instance

Parameters **setting** (str) – server setting name

Return type Dict

get_sharing_group(*sharing_group*, *pythonify=False*)
Get a sharing group

Parameters

- **sharing_group** (Union[*MISPSharingGroup*, int, str, UUID]) – sharing group to find
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPSharingGroup*]

get_sync_config(*pythonify=False*)
Get the sync server config. WARNING: This method only works if the user calling it is a sync user

Parameters **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPServer*]

get_tag(*tag*, *pythonify=False*)
Get a tag by id.

Parameters

- **tag** (Union[*MISPTag*, int, str, UUID]) – tag to get
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPTag*]

get_taxonomy(*taxonomy*, *pythonify=False*)
Get a taxonomy by id or namespace from a MISP instance

Parameters

- **taxonomy** (Union[*MISPTaxonomy*, int, str, UUID]) – taxonomy to get
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPTaxonomy*]

get_user(*user='me'*, *pythonify=False*, *expanded=False*)
Get a user by id

Parameters

- **user** (Union[*MISPUser*, int, str, UUID]) – user to get; *me* means the owner of the API key doing the query
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output
- **expanded** (bool) – Also returns a MISPRole and a MISPUserSetting

Return type Union[Dict, *MISPUser*, Tuple[*MISPUser*, *MISPRole*, List[*MISPUserSetting*]]]

get_user_setting(*user_setting*, *user=None*, *pythonify=False*)
Get a user setting

Parameters

- **user_setting** (str) – name of user setting
- **user** (Union[*MISPUser*, int, str, UUID, None]) – user
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPUserSetting*]

get_warninglist(*warninglist*, *pythonify=False*)
Get a warninglist by id

Parameters

- **warninglist** (Union[*MISPWarninglist*, int, str, UUID]) – warninglist to get
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPWarninglist*]

import_server(*server*, *pythonify=False*)
Import a sync server config received from get_sync_config

Parameters

- **server** (*MISPServer*) – sync server config
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPServer*]

load_default_feeds()
Load all the default feeds.

Return type Dict

property misp_instance_version: Dict
Returns the version of the instance.

Return type Dict

property misp_instance_version_master: Dict
Get the most recent version from github

Return type Dict

noticelists(*pythonify=False*)
Get all the noticelists

Parameters **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPNoticelist*]]

object_exists(*misp_object*)
Fast check if object exists.

Parameters **misp_object** (Union[*MISPObject*, int, str, UUID]) – Attribute to check

Return type bool

object_templates(*pythonify=False*)
Get all the object templates

Parameters **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPObjectTemplate*]]

organisation_blocklists(*pythonify=False*)
Get all the blocklisted organisations

Parameters **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPOrganisationBlocklist*]]

organisation_exists(*organisation*)
Fast check if organisation exists.

Parameters **organisation** (Union[*MISPOrganisation*, int, str, UUID]) – Organisation to check

Return type bool

organisations(*scope='local'*, *search=None*, *pythonify=False*)
Get all the organisations

Parameters

- **scope** – scope of organizations to get
- **search** (Optional[str]) – The search to make against the list of organisations
- **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPOrganisation*]]

publish(*event*, *alert=False*)
Publish the event with one single HTTP POST

Parameters

- **event** (Union[*MISPEvent*, int, str, UUID]) – event to publish
- **alert** (bool) – whether to send an email. The default is to not send a mail as it is assumed this method is called on update.

Return type Dict

publish_galaxy_cluster(*galaxy_cluster*)
Publishes a galaxy cluster

Parameters **galaxy_cluster** (Union[MISPGalaxyCluster, int, str, UUID]) – The galaxy cluster you wish to publish

Return type Dict

push_event_to_ZMQ(*event*)
Force push an event by id on ZMQ

Parameters **event** (Union[*MISPEvent*, int, str, UUID]) – the event to push

Return type Dict

property pymisp_version_main: Dict
Get the most recent version of PyMISP from github

Return type Dict

property pymisp_version_master: Dict
PyMISP version as defined in the main repository

Return type Dict

property recommended_pymisp_version: Dict
Returns the recommended API version from the server

Return type Dict

remote_acl(*debug_type='findMissingFunctionNames'*)
This should return an empty list, unless the ACL is outdated.

Parameters **debug_type** (str) – printAllFunctionNames, findMissingFunctionNames, or printRoleAccess

Return type Dict

remove_org_from_sharing_group(*sharing_group*, *organisation*)
Remove an organisation from a sharing group.

Parameters

- **sharing_group** (Union[*MISPSharingGroup*, int, str, UUID]) – Sharing group's local instance ID, or Sharing group's global UUID
- **organisation** (Union[*MISPOrganisation*, int, str, UUID]) – Organisation's local instance ID, or Organisation's global UUID, or Organisation's name as known to the curent instance

Return type Dict

remove_server_from_sharing_group(*sharing_group*, *server*)
Remove a server from a sharing group.

Parameters

- **sharing_group** (Union[*MISPSharingGroup*, int, str, UUID]) – Sharing group's local instance ID, or Sharing group's global UUID
- **server** (Union[*MISPServer*, int, str, UUID]) – Server's local instance ID, or URL of the Server, or Server's name as known to the curent instance

Return type Dict

request_community_access(*community*, *requestor_email_address=None*, *requestor_gpg_key=None*, *requestor_organisation_name=None*, *requestor_organisation_uuid=None*, *requestor_organisation_description=None*, *message=None*, *sync=False*, *anonymise_requestor_server=False*, *mock=False*)
Request the access to a community

Parameters

- **community** (Union[MISPCommunity, int, str, UUID]) – community to request access
- **requestor_email_address** (Optional[str]) – requestor email
- **requestor_gpg_key** (Optional[str]) – requestor key
- **requestor_organisation_name** (Optional[str]) – requestor org name
- **requestor_organisation_uuid** (Optional[str]) – requestor org ID
- **requestor_organisation_description** (Optional[str]) – requestor org desc
- **message** (Optional[str]) – requestor message
- **sync** (bool) – synchronize flag
- **anonymise_requestor_server** (bool) – anonymise flag
- **mock** (bool) – mock flag

Return type Dict

restart_workers()
Restart all the workers

Return type Dict

roles(*pythonify=False*)
Get the existing roles

Parameters **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPRole*]]

search(*controller='events'*, *return_format='json'*, *limit=None*, *page=None*, *value=None*, *type_attribute=None*, *category=None*, *org=None*, *tags=None*, *quick_filter=None*, *quickFilter=None*, *date_from=None*, *date_to=None*, *eventid=None*, *with_attachments=None*, *withAttachments=None*, *metadata=None*, *uuid=None*, *publish_timestamp=None*, *last=None*, *timestamp=None*, *published=None*, *enforce_warninglist=None*, *enforceWarninglist=None*, *to_ids=None*, *deleted=None*, *include_event_uuid=None*, *includeEventUuid=None*, *include_event_tags=None*, *includeEventTags=None*, *event_timestamp=None*, *sg_reference_only=None*, *eventinfo=None*, *searchall=None*, *requested_attributes=None*, *include_context=None*, *includeContext=None*, *headerless=None*, *include_sightings=None*, *includeSightings=None*, *include_correlations=None*, *includeCorrelations=None*, *include_decay_score=None*, *includeDecayScore=None*, *object_name=None*, *exclude_decayed=None*, *pythonify=False*, ***kwargs*)
Search in the MISP instance

Parameters

- **controller** (str) – Controller to search on, it can be *events*, *objects*, *attributes*. The response will either be a list of events, objects, or attributes.
- **return_format** (str) – Set the return format of the search (Currently supported: json, xml, openioc, suricata, snort - more formats are being moved to restSearch with the goal being that all searches happen through this API). Can be passed as the first parameter after restSearch or via the JSON payload.
- **limit** (Optional[int]) – Limit the number of results returned, depending on the scope (for example 10 attributes or 10 full events).
- **page** (Optional[int]) – If a limit is set, sets the page to be returned. page 3, limit 100 will return records 201->300).
- **value** (Optional[~SearchParameterTypes]) – Search for the given value in the attributes' value field.
- **type_attribute** (Optional[~SearchParameterTypes]) – The attribute type, any valid MISP attribute type is accepted.
- **category** (Optional[~SearchParameterTypes]) – The attribute category, any valid MISP attribute category is accepted.
- **org** (Optional[~SearchParameterTypes]) – Search by the creator organisation by supplying the organisation identifier.
- **tags** (Optional[~SearchParameterTypes]) – Tags to search or to exclude. You can pass a list, or the output of *build_complex_query*
- **quick_filter** (Optional[str]) – The string passed to this field will ignore all of the other arguments. MISP will return an xml / json (depending on the header sent) of all

events that have a sub-string match on value in the event info, event orgc, or any of the attribute value1 / value2 fields, or in the attribute comment.

- **date_from** (Union[date, int, str, float, None]) – Events with the date set to a date after the one specified. This filter will use the date of the event.
- **date_to** (Union[date, int, str, float, None]) – Events with the date set to a date before the one specified. This filter will use the date of the event.
- **eventid** (Optional[~SearchType]) – The events that should be included / excluded from the search
- **with_attachments** (Optional[bool]) – If set, encodes the attachments / zipped malware samples as base64 in the data field within each attribute
- **metadata** (Optional[bool]) – Only the metadata (event, tags, relations) is returned, attributes and proposals are omitted.
- **uuid** (Optional[str]) – Restrict the results by uuid.
- **publish_timestamp** (Union[date, int, str, float, None, Tuple[Union[date, int, str, float, None], Union[date, int, str, float, None]]]) – Restrict the results by the last publish timestamp (newer than).
- **timestamp** (Union[date, int, str, float, None, Tuple[Union[date, int, str, float, None], Union[date, int, str, float, None]]]) – Restrict the results by the timestamp (last edit). Any event with a timestamp newer than the given timestamp will be returned. In case you are dealing with /attributes as scope, the attribute's timestamp will be used for the lookup. The input can be a timestamp or a short-hand time description (7d or 24h for example). You can also pass a list with two values to set a time range (for example ["14d", "7d"]).
- **published** (Optional[bool]) – Set whether published or unpublished events should be returned. Do not set the parameter if you want both.
- **enforce_warninglist** (Optional[bool]) – Remove any attributes from the result that would cause a hit on a warninglist entry.
- **to_ids** (Union[~ToIDSType, List[~ToIDSType], None]) – By default all attributes are returned that match the other filter parameters, regardless of their to_ids setting. To restrict the returned data set to to_ids only attributes set this parameter to 1. 0 for the ones with to_ids set to False.
- **deleted** (Optional[str]) – If this parameter is set to 1, it will only return soft-deleted attributes. ["0", "1"] will return the active ones as well as the soft-deleted ones.
- **include_event_uuid** (Optional[bool]) – Instead of just including the event ID, also include the event UUID in each of the attributes.
- **include_event_tags** (Optional[bool]) – Include the event level tags in each of the attributes.
- **event_timestamp** (Union[date, int, str, float, None]) – Only return attributes from events that have received a modification after the given timestamp.
- **sg_reference_only** (Optional[bool]) – If this flag is set, sharing group objects will not be included, instead only the sharing group ID is set.
- **eventinfo** (Optional[str]) – Filter on the event's info field.
- **searchall** (Optional[bool]) – Search for a full or a substring (delimited by % for substrings) in the event info, event tags, attribute tags, attribute values or attribute comment fields.

- **requested_attributes** (Optional[str]) – [CSV only] Select the fields that you wish to include in the CSV export. By setting event level fields additionally, includeContext is not required to get event metadata.
- **include_context** (Optional[bool]) – [Attribute only] Include the event data with each attribute. [CSV output] Add event level metadata in every line of the CSV.
- **headerless** (Optional[bool]) – [CSV Only] The CSV created when this setting is set to true will not contain the header row.
- **include_sightings** (Optional[bool]) – [JSON Only - Attribute] Include the sightings of the matching attributes.
- **include_decay_score** (Optional[bool]) – Include the decay score at attribute level.
- **include_correlations** (Optional[bool]) – [JSON Only - attribute] Include the correlations of the matching attributes.
- **object_name** (Optional[str]) – [objects controller only] Search for objects with that name
- **exclude_decayed** (Optional[bool]) – [attributes controller only] Exclude the decayed attributes from the response
- **pythonify** (Optional[bool]) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Deprecated:

Parameters

- **quickFilter** (Optional[str]) – synonym for quick_filter
- **withAttachments** (Optional[bool]) – synonym for with_attachments
- **last** (Union[date, int, str, float, None, Tuple[Union[date, int, str, float, None], Union[date, int, str, float, None]]]) – synonym for publish_timestamp
- **enforceWarninglist** (Optional[bool]) – synonym for enforce_warninglist
- **includeEventUuid** (Optional[bool]) – synonym for include_event_uuid
- **includeEventTags** (Optional[bool]) – synonym for include_event_tags
- **includeContext** (Optional[bool]) – synonym for include_context

Return type Union[Dict, str, List[Union[*MISPEvent*, *MISPAttribute*, *MISPObject*]]]

search_feeds(*value=None*, *pythonify=False*)
Search in the feeds cached on the servers

Return type Union[Dict, List[*MISPFeed*]]

search_galaxy_clusters(*galaxy*, *context='all'*, *searchall=None*, *pythonify=False*)
Searches the galaxy clusters within a specific galaxy

Parameters

- **galaxy** (Union[MISPGalaxy, int, str, UUID]) – The MISPGalaxy you wish to search in
- **context** (str) – The context of how you want to search within the **galaxy_**
- **searchall** (Optional[str]) – The search you want to make against the galaxy and context
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, List[MISPGalaxyCluster]]

search_index(*all=None*, *attribute=None*, *email=None*, *published=None*, *hasproposal=None*, *eventid=None*, *tags=None*, *date_from=None*, *date_to=None*, *eventinfo=None*, *threatlevel=None*, *distribution=None*, *analysis=None*, *org=None*, *timestamp=None*, *publish_timestamp=None*, *sharinggroup=None*, *minimal=None*, *pythonify=None*)

Search event metadata shown on the event index page. Using ! in front of a value means NOT, except for parameters date_from, date_to and timestamp which cannot be negated. Criteria are AND-ed together; values in lists are OR-ed together. Return matching events with metadata but no attributes or objects; also see minimal parameter.

Parameters

- **all** (Optional[str]) – Search for a full or a substring (delimited by % for substrings) in the event info, event tags, attribute tags, attribute values or attribute comment fields.
- **attribute** (Optional[str]) – Filter on attribute's value.
- **email** (Optional[str]) – Filter on user's email.
- **published** (Optional[bool]) – Set whether published or unpublished events should be returned. Do not set the parameter if you want both.
- **hasproposal** (Optional[bool]) – Filter for events containing proposal(s).
- **eventid** (Optional[~SearchType]) – The events that should be included / excluded from the search
- **tags** (Optional[~SearchParameterTypes]) – Tags to search or to exclude. You can pass a list, or the output of *build_complex_query*
- **date_from** (Union[date, int, str, float, None]) – Events with the date set to a date after the one specified. This filter will use the date of the event.
- **date_to** (Union[date, int, str, float, None]) – Events with the date set to a date before the one specified. This filter will use the date of the event.
- **eventinfo** (Optional[str]) – Filter on the event's info field.
- **threatlevel** (Optional[List[~SearchType]]) – Threat level(s) (1,2,3,4) | list
- **distribution** (Optional[List[~SearchType]]) – Distribution level(s) (0,1,2,3) | list
- **analysis** (Optional[List[~SearchType]]) – Analysis level(s) (0,1,2) | list
- **org** (Optional[~SearchParameterTypes]) – Search by the creator organisation by supplying the organisation identifier.
- **timestamp** (Union[date, int, str, float, None, Tuple[Union[date, int, str, float, None], Union[date, int, str, float, None]]]) – Restrict the results by the timestamp (last edit). Any event with a timestamp newer than the given timestamp will be returned. In case you are dealing with /attributes as scope, the attribute's timestamp will be used for the lookup.
- **publish_timestamp** (Union[date, int, str, float, None, Tuple[Union[date, int, str, float, None], Union[date, int, str, float, None]]]) – Filter on event's publish timestamp.
- **sharinggroup** (Optional[List[~SearchType]]) – Restrict by a sharing group | list
- **minimal** (Optional[bool]) – Return only event ID, UUID, timestamp, sighting_timestamp and published.

- **pythonify** (Optional[bool]) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPEvent*]]

search_logs(*limit=None*, *page=None*, *log_id=None*, *title=None*, *created=None*, *model=None*, *action=None*, *user_id=None*, *change=None*, *email=None*, *org=None*, *description=None*, *ip=None*, *pythonify=False*)

Search in logs

Note: to run substring queries simply append/prepend/encapsulate the search term with %

Parameters

- **limit** (Optional[int]) – Limit the number of results returned, depending on the scope (for example 10 attributes or 10 full events).
- **page** (Optional[int]) – If a limit is set, sets the page to be returned. page 3, limit 100 will return records 201->300).
- **log_id** (Optional[int]) – Log ID
- **title** (Optional[str]) – Log Title
- **created** (Union[date, int, str, float, None]) – Creation timestamp
- **model** (Optional[str]) – Model name that generated the log entry
- **action** (Optional[str]) – The thing that was done
- **user_id** (Optional[int]) – ID of the user doing the action
- **change** (Optional[str]) – Change that occured
- **email** (Optional[str]) – Email of the user
- **org** (Optional[str]) – Organisation of the User doing the action
- **description** (Optional[str]) – Description of the action
- **ip** (Optional[str]) – Origination IP of the User doing the action
- **pythonify** (Optional[bool]) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPLog*]]

search_sightings(*context=None*, *context_id=None*, *type_sighting=None*, *date_from=None*, *date_to=None*, *publish_timestamp=None*, *last=None*, *org=None*, *source=None*, *include_attribute=None*, *include_event_meta=None*, *pythonify=False*)

Search sightings

Parameters

- **context** (Optional[str]) – The context of the search. Can be either “attribute”, “event”, or nothing (will then match on events and attributes).
- **context_id** (Optional[~SearchType]) – Only relevant if context is either “attribute” or “event”. Then it is the relevant ID.
- **type_sighting** (Optional[str]) – Type of sighting
- **date_from** (Union[date, int, str, float, None]) – Events with the date set to a date after the one specified. This filter will use the date of the event.
- **date_to** (Union[date, int, str, float, None]) – Events with the date set to a date before the one specified. This filter will use the date of the event.

- **publish_timestamp** (Union[date, int, str, float, None, Tuple[Union[date, int, str, float, None], Union[date, int, str, float, None]]]) – Restrict the results by the last publish timestamp (newer than).
- **org** (Optional[~SearchType]) – Search by the creator organisation by supplying the organisation identifier.
- **source** (Optional[str]) – Source of the sighting
- **include_attribute** (Optional[bool]) – Include the attribute.
- **include_event_meta** (Optional[bool]) – Include the meta information of the event.

Deprecated:

Parameters **last** (Union[date, int, str, float, None, Tuple[Union[date, int, str, float, None], Union[date, int, str, float, None]]]) – synonym for publish_timestamp

Example

```
>>> misp.search_sightings(publish_timestamp='30d') # search sightings for the␣
↪last 30 days on the instance
[ ... ]
>>> misp.search_sightings(context='attribute', context_id=6, include_
↪attribute=True) # return list of sighting for attribute 6 along with the␣
↪attribute itself
[ ... ]
>>> misp.search_sightings(context='event', context_id=17, include_event_
↪meta=True, org=2) # return list of sighting for event 17 filtered with org id␣
↪2
```

Return type Union[Dict, List[Dict[str, Union[*MISPEvent*, *MISPAttribute*, *MISPSighting*]]]]

search_tags(*tagname*, *strict_tagname=False*, *pythonify=False*)

Search for tags by name.

Parameters

- **tag_name** – Name to search, use % for substrings matches.
- **strict_tagname** (bool) – only return tags matching exactly the tag name (so skipping synonyms and cluster's value)

Return type Union[Dict, List[*MISPTag*]]

server_pull(*server*, *event=None*)

Initialize a pull from a sync server, optionally limited to one event

Parameters

- **server** (Union[*MISPServer*, int, str, UUID]) – sync server config
- **event** (Union[*MISPEvent*, int, str, UUID, None]) – event

Return type Dict

server_push(*server*, *event=None*)

Initialize a push to a sync server, optionally limited to one event

Parameters

- **server** (Union[*MISPServer*, int, str, UUID]) – sync server config

- **event** (Union[*MISPEvent*, int, str, UUID, None]) – event

Return type Dict

server_settings()
Get all the settings from the server

Return type Dict

servers(*pythonify=False*)
Get the existing servers the MISP instance can synchronise with

Parameters **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPServer*]]

set_default_role(*role*)
Set a default role for the new user accounts

Parameters **role** (Union[*MISPRole*, int, str, UUID]) – the default role to set

Return type Dict

set_server_setting(*setting*, *value*, *force=False*)
Set a setting on the MISP instance

Parameters

- **setting** (str) – server setting name
- **value** (Union[str, int]) – value to set
- **force** (bool) – override value test

Return type Dict

set_user_setting(*user_setting*, *value*, *user=None*, *pythonify=False*)
Set a user setting

Parameters

- **user_setting** (str) – name of user setting
- **value** (Union[str, dict]) – value to set
- **user** (Union[*MISPUser*, int, str, UUID, None]) – user
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPUserSetting*]

sharing_group_exists(*sharing_group*)
Fast check if sharing group exists.

Parameters **sharing_group** (Union[*MISPSharingGroup*, int, str, UUID]) – Sharing group to check

Return type bool

sharing_groups(*pythonify=False*)
Get the existing sharing groups

Parameters **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPSharingGroup*]]

sightings(*misp_entity=None*, *org=None*, *pythonify=False*)
Get the list of sightings related to a MISPEvent or a MISPAttribute (depending on type of misp_entity)

Parameters

- **misp_entity** (Optional[*AbstractMISP*]) – MISP entity
- **org** (Union[*MISPOrganisation*, int, str, UUID, None]) – MISP organization
- **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPSighting*]]

tag(*misp_entity*, *tag*, *local=False*)
Tag an event or an attribute.

Parameters

- **misp_entity** (Union[*AbstractMISP*, str, dict]) – a MISPEvent, a MISP Attribute, or a UUID
- **tag** (Union[*MISPTag*, str]) – tag to add
- **local** (bool) – whether to tag locally

Return type Dict

tags(*pythonify=False*, ***kw_params*)
Get the list of existing tags.

Parameters **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPTag*]]

tags_statistics(*percentage=False*, *name_sort=False*)
Get tag statistics from the MISP instance

Parameters

- **percentage** (bool) – get percentages
- **name_sort** (bool) – sort by name

Return type Dict

taxonomies(*pythonify=False*)
Get all the taxonomies

Parameters **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPTaxonomy*]]

test_server(*server*)
Test if a sync link is working as expected

Parameters **server** (Union[*MISPServer*, int, str, UUID]) – sync server config

Return type Dict

toggle_global_pythonify()
Toggle the pythonify variable for the class

Return type None

toggle_warninglist(*warninglist_id=None*, *warninglist_name=None*, *force_enable=False*)
Toggle (enable/disable) the status of a warninglist by id

Parameters

- **warninglist_id** (Union[str, int, List[int], None]) – ID of the WarningList
- **warninglist_name** (Union[str, List[str], None]) – name of the WarningList
- **force_enable** (bool) – Force the warning list in the enabled state (does nothing if already enabled)

Return type Dict

untag(*misp_entity*, *tag*)
Untag an event or an attribute

Parameters

- **misp_entity** (Union[*AbstractMISP*, str, dict]) – misp_entity can be a UUID
- **tag** (Union[*MISPTag*, str]) – tag to remove

Return type Dict

update_attribute(*attribute*, *attribute_id=None*, *pythonify=False*)
Update an attribute on a MISP instance

Parameters

- **attribute** (*MISPAttribute*) – attribute to update
- **attribute_id** (Optional[int]) – attribute ID to update
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPAttribute*, *MISPShadowAttribute*]

update_attribute_proposal(*initial_attribute*, *attribute*, *pythonify=False*)
Propose a change for an attribute

Parameters

- **initial_attribute** (Union[*MISPAttribute*, int, str, UUID]) – attribute to change
- **attribute** (*MISPAttribute*) – attribute to propose
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPShadowAttribute*]

update_event(*event*, *event_id=None*, *pythonify=False*, *metadata=False*)
Update an event on a MISP instance"'

Parameters

- **event** (*MISPEvent*) – event to update
- **event_id** (Optional[int]) – ID of event to update
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output
- **metadata** (bool) – Return just event metadata after successful update

Return type Union[Dict, *MISPEvent*]

update_event_blocklist(*event_blocklist*, *event_blocklist_id=None*, *pythonify=False*)
Update an event in the blocklist

Parameters

- **event_blocklist** (*MISPEventBlocklist*) – event block list
- **event_blocklist_id** (Union[int, str, UUID, None]) – event block lisd id
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPEventBlocklist*]

update_event_report(*event_report*, *event_report_id=None*, *pythonify=False*)
Update an event report on a MISP instance

Parameters

- **event_report** (MISPEventReport) – event report to update
- **event_report_id** (Optional[int]) – event report ID to update
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, MISPEventReport]

update_feed(*feed*, *feed_id=None*, *pythonify=False*)
Update a feed on a MISP instance

Parameters

- **feed** (*MISPFeed*) – feed to update
- **feed_id** (Optional[int]) – feed id
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPFeed*]

update_galaxies()
Update all the galaxies.

Return type Dict

update_galaxy_cluster(*galaxy_cluster*, *pythonify=False*)
Update a custom galaxy cluster.

;param galaxy_cluster: The MISPGalaxyCluster you wish to update :type pythonify: bool :param pythonify: Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, MISPGalaxyCluster]

update_galaxy_cluster_relation(*galaxy_cluster_relation*)
Update a galaxy cluster relation

Parameters **galaxy_cluster_relation** (MISPGalaxyClusterRelation) – The MISPGalaxyClusterRelation to update

Return type Dict

update_misp()
Trigger a server update

Return type Dict

update_noticelists()
Update all the noticelists.

Return type Dict

update_object(*misp_object*, *object_id=None*, *pythonify=False*)
Update an object on a MISP instance

Parameters

- **misp_object** (*MISPObject*) – object to update
- **object_id** (Optional[int]) – ID of object to update
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPObject*]

update_object_templates()
Trigger an update of the object templates

Return type Dict

update_organisation(*organisation*, *organisation_id=None*, *pythonify=False*)
Update an organisation

Parameters

- **organisation** (*MISPOrganisation*) – organization to update
- **organisation_id** (Optional[int]) – id to update
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPOrganisation*]

update_organisation_blocklist(*organisation_blocklist*, *organisation_blocklist_id=None*, *pythonify=False*)
Update an organisation in the blocklist

Parameters

- **organisation_blocklist** (*MISPOrganisationBlocklist*) – organization block list
- **organisation_blocklist_id** (Union[int, str, UUID, None]) – organization block lisd id
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPOrganisationBlocklist*]

update_server(*server*, *server_id=None*, *pythonify=False*)
Update a server to synchronise with

Parameters

- **server** (*MISPServer*) – sync server config
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPServer*]

update_sharing_group(*sharing_group*, *sharing_group_id=None*, *pythonify=False*)
Update sharing group parameters

Parameters **sharing_group** (Union[*MISPSharingGroup*, dict]) – MISP Sharing Group

:param sharing_group_id Sharing group ID :type pythonify: bool :param pythonify: Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPSharingGroup*]

update_tag(*tag*, *tag_id=None*, *pythonify=False*)
Edit only the provided parameters of a tag

Parameters

- **tag** (*MISPTag*) – tag to update
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Aram tag_id tag ID to update

Return type Union[Dict, *MISPTag*]

update_taxonomies()
Update all the taxonomies.

Return type Dict

update_user(*user*, *user_id=None*, *pythonify=False*)
Update a user on a MISP instance

Parameters

- **user** (*MISPUser*) – user to update
- **user_id** (Optional[int]) – id to update
- **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type Union[Dict, *MISPUser*]

update_warninglists()
Update all the warninglists.

Return type Dict

upload_stix(*path=None*, *data=None*, *version='2'*)
Upload a STIX file to MISP.

Parameters

- **path** (Union[str, Path, BytesIO, StringIO, None]) – Path to the STIX on the disk (can be a path-like object, or a pseudofile)
- **data** (Union[str, bytes, None]) – stix object
- **version** (str) – Can be 1 or 2

user_registrations(*pythonify=False*)
Get all the user registrations

Parameters **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPInbox*]]

user_settings(*pythonify=False*)
Get all the user settings

Parameters **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPUserSetting*]]

users(*search=None*, *organisation=None*, *pythonify=False*)
Get all the users, or a filtered set of users

Parameters

- **search** (Optional[str]) – The search to make against the list of users
- **organisation** (Optional[int]) – The ID of an organisation to filter against
- **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPUser*]]

users_statistics(*context='data'*)
Get user statistics from the MISP instance

Parameters **context** (str) – one of 'data', 'orgs', 'users', 'tags', 'attributehistogram', 'sightings', 'galaxyMatrix'

Return type Dict

values_in_warninglist(*value*)
Check if IOC values are in warninglist

Parameters **value** (Iterable) – iterator with values to check

Return type Dict

property version: Dict
Returns the version of PyMISP you're currently using

Return type Dict

warninglists(*pythonify=False*)
Get all the warninglists.

Parameters **pythonify** (bool) – Returns a list of PyMISP Objects instead of the plain json output. Warning: it might use a lot of RAM

Return type Union[Dict, List[*MISPWarninglist*]]

3.2 MISPAbstract

class pymisp.**AbstractMISP**(***kwargs*)

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

Return type None

from_json(*json_string*)
Load a JSON string

Return type None

jsonable()
This method is used by the JSON encoder

Return type Dict

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

3.3 MISPEncode

class pymisp.**MISPEncode**(**args*, ***kwargs*)

default(*obj*)
Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

3.4 MISPEvent

class pymisp.**MISPEvent**(*describe_types=None*, *strict_validation=False*, ***kwargs*)

add_attribute(*type*, *value*, ***kwargs*)
Add an attribute. type and value are required but you can pass all other parameters supported by MISPAttribute

Return type Union[*MISPAttribute*, List[*MISPAttribute*]]

add_attribute_tag(*tag*, *attribute_identifier*)
Add a tag to an existing attribute. Raise an Exception if the attribute doesn't exist.

Parameters

- **tag** (Union[*MISPTag*, str]) – Tag name as a string, MISPTag instance, or dictionary
- **attribute_identifier** (str) – can be an ID, UUID, or the value.

Return type List[*MISPAttribute*]

add_event_report(*name*, *content*, ***kwargs*)
Add an event report. name and value are requred but you can pass all other parameters supported by MISPEventReport

Return type MISPEventReport

add_galaxy(***kwargs*)
Add a MISP galaxy and sub-clusters into an event. Supports all other parameters supported by MISPGalaxy

Return type MISPGalaxy

add_object(*obj=None*, ***kwargs*)
Add an object to the Event, either by passing a MISPObject, or a dictionary

Return type *MISPObject*

add_proposal(*shadow_attribute=None*, ***kwargs*)
Alias for add_shadow_attribute

Return type *MISPShadowAttribute*

add_shadow_attribute(*shadow_attribute=None*, ***kwargs*)
Add a tag to the attribute (by name or a MISPTag object)

Return type *MISPShadowAttribute*

clear() → None. Remove all items from D.

delete_attribute(*attribute_id*)
Delete an attribute

Parameters **attribute_id** (str) – ID or UUID

delete_object(*object_id*)
Delete an object

Parameters **object_id** (str) – ID or UUID

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

get_attribute_tag(*attribute_identifier*)
Return the tags associated to an attribute or an object attribute.

Parameters **attribute_identifier** (str) – can be an ID, UUID, or the value.

Return type List[*MISPTag*]

get_object_by_id(*object_id*)
Get an object by ID

Parameters **object_id** (Union[str, int]) – the ID is the one set by the server when creating the new object

Return type *MISPObject*

get_object_by_uuid(*object_uuid*)
Get an object by UUID

Parameters **object_uuid** (str) – the UUID is set by the server when creating the new object

Return type *MISPObject*

get_objects_by_name(*object_name*)
Get objects by name

Parameters **object_name** (str) – name is set by the server when creating the new object

Return type List[*MISPObject*]

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

load(*json_event*, *validate=False*, *metadata_only=False*)
Load a JSON dump from a pseudo file or a JSON string

load_file(*event_path*, *validate=False*, *metadata_only=False*)
Load a JSON dump from a file on the disk

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

publish()
Mark the attribute as published

set_date(*d=None*, *ignore_invalid=False*)
Set a date for the event

Parameters

- **d** (Union[date, int, str, float, None]) – String, datetime, or date object
- **ignore_invalid** (bool) – if True, assigns current date if d is not an expected type

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

property tags: List[*pymisp.abstract.MISPTag*]
Returns a list of tags associated to this Event

Return type List[*MISPTag*]

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_feed(*valid_distributions=[0, 1, 2, 3, 4, 5]*, *with_meta=False*, *with_distribution=False*)
Generate a json output for MISP Feed.

Parameters

- **valid_distributions** (List[int]) – only makes sense if the distribution key is set; i.e., the event is exported from a MISP instance.
- **with_distribution** – exports distribution and Sharing Group info; otherwise all SharingGroup information is discarded (protecting privacy)

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

unpublish()
Mark the attribute as un-published (set publish flag to false)

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.5 MISPEventBlocklist

class pymisp.**MISPEventBlocklist**(***kwargs*)

clear() → None. Remove all items from D.

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.6 MISPEventDelegation

class pymisp.**MISPEventDelegation**(***kwargs*)

clear() → None. Remove all items from D.

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.7 MISPAttribute

class pymisp.**MISPAttribute**(*describe_types=None*, *strict=False*)

add_proposal(*shadow_attribute=None*, ***kwargs*)
Alias for add_shadow_attribute

Return type *MISPShadowAttribute*

add_shadow_attribute(*shadow_attribute=None*, ***kwargs*)
Add a shadow attribute to the attribute (by name or a MISPShadowAttribute object)

Return type *MISPShadowAttribute*

add_sighting(*sighting=None*, ***kwargs*)
Add a sighting to the attribute (by name or a MISPSighting object)

Return type *MISPSighting*

clear() → None. Remove all items from D.

delete()
Mark the attribute as deleted (soft delete)

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

hash_values(*algorithm='sha512'*)
Compute the hash of every value for fast lookups

Return type List[str]

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

property known_types: List[str]
Returns a list of all the known MISP attributes types

Return type List[str]

property malware_binary: Optional[_io.BytesIO]
Returns a BytesIO of the malware, if the attribute has one. Decrypts, unpacks and caches the binary on the first invocation, which may require some time for large attachments (~1s/MB).

Return type Optional[BytesIO]

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

property tags: List[*pymisp.abstract.MISPTag*]
Returns a list of tags associated to this Attribute

Return type List[*MISPTag*]

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.8 MISPObject

class pymisp.**MISPObject**(*name*, *strict=False*, *standalone=True*, *default_attributes_parameters={}*, ***kwargs*)

add_attribute(*object_relation*, *simple_value=None*, ***value*)
Add an attribute. object_relation is required and the value key is a dictionary with all the keys supported by MISPAttribute

Return type Optional[*MISPAttribute*]

add_attributes(*object_relation*, **attributes*)
Add multiple attributes with the same object_relation. Helper for object_relation when multiple is True in the template. It is the same as calling multiple times add_attribute with the same object_relation.

Return type List[Optional[*MISPAttribute*]]

add_reference(*referenced_uuid*, *relationship_type*, *comment=None*, ***kwargs*)
Add a link (uuid) to another object

Return type *MISPObjectReference*

clear() → None. Remove all items from D.

delete()
Mark the object as deleted (soft delete)

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

get_attributes_by_relation(*object_relation*)
Returns the list of attributes with the given object relation in the object

Return type List[*MISPAttribute*]

has_attributes_by_relation(*list_of_relations*)
True if all the relations in the list are defined in the object

Return type bool

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*, *strict=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*, *strict=False*)
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.9 MISPObjectAttribute

class pymisp.**MISPObjectAttribute**(*definition*)

add_proposal(*shadow_attribute=None*, ***kwargs*)
Alias for add_shadow_attribute

Return type *MISPShadowAttribute*

add_shadow_attribute(*shadow_attribute=None*, ***kwargs*)
Add a shadow attribute to the attribute (by name or a MISPShadowAttribute object)

Return type *MISPShadowAttribute*

add_sighting(*sighting=None*, ***kwargs*)
Add a sighting to the attribute (by name or a MISPSighting object)

Return type *MISPSighting*

clear() → None. Remove all items from D.

delete()
Mark the attribute as deleted (soft delete)

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(*object_relation*, *value*, ***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

hash_values(*algorithm='sha512'*)
Compute the hash of every value for fast lookups

Return type List[str]

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

property known_types: List[str]
Returns a list of all the known MISP attributes types

Return type List[str]

property malware_binary: Optional[_io.BytesIO]
Returns a BytesIO of the malware, if the attribute has one. Decrypts, unpacks and caches the binary on the first invocation, which may require some time for large attachments (~1s/MB).

Return type Optional[BytesIO]

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

property tags: List[*pymisp.abstract.MISPTag*]
Returns a list of tags associated to this Attribute

Return type List[*MISPTag*]

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.10 MISPObjectReference

class pymisp.**MISPObjectReference**

clear() → None. Remove all items from D.

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.11 MISPObjectTemplate

class pymisp.**MISPObjectTemplate**(***kwargs*)

clear() → None. Remove all items from D.

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.12 MISPTag

class pymisp.**MISPTag**(***kwargs*)

clear() → None. Remove all items from D.

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.13 MISPUser

class pymisp.**MISPUser**(***kwargs*)

clear() → None. Remove all items from D.

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.14 MISPUserSetting

class pymisp.**MISPUserSetting**(**kwargs)

clear() → None. Remove all items from D.

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(**kwargs)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.15 MISPOrganisation

class pymisp.**MISPOrganisation**

clear() → None. Remove all items from D.

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.16 MISPOrganisationBlocklist

class pymisp.**MISPOrganisationBlocklist**(***kwargs*)

clear() → None. Remove all items from D.

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.17 MISPFeed

class pymisp.**MISPFeed**(***kwargs*)

clear() → None. Remove all items from D.

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.18 MISPInbox

class pymisp.**MISPInbox**(***kwargs*)

clear() → None. Remove all items from D.

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], **F) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.19 MISPLog

class pymisp.**MISPLog**(**kwargs*)

clear() → None. Remove all items from D.

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(**kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.20 MISPNoticelist

class pymisp.**MISPNoticelist**(***kwargs*)

clear() → None. Remove all items from D.

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.21 MISPRole

class pymisp.**MISPRole**(***kwargs*)

clear() → None. Remove all items from D.

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.22 MISPServer

class pymisp.**MISPServer**(***kwargs*)

clear() → None. Remove all items from D.

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.23 MISPShadowAttribute

class pymisp.**MISPShadowAttribute**

clear() → None. Remove all items from D.

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.24 MISPSharingGroup

class pymisp.**MISPSharingGroup**

clear() → None. Remove all items from D.

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.25 MISPSighting

class pymisp.**MISPSighting**

clear() → None. Remove all items from D.

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Initialize the MISPSighting from a dictionary

Parameters

- **value** – Value of the attribute the sighting is related too. Pushing this object will update the sighting count of each attribute with this value on the instance.
- **uuid** – UUID of the attribute to update
- **id** – ID of the attriute to update
- **source** – Source of the sighting
- **type** – Type of the sighting
- **timestamp** – Timestamp associated to the sighting

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.26 MISPTaxonomy

class pymisp.**MISPTaxonomy**(***kwargs*)

clear() → None. Remove all items from D.

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

3.27 MISPWarninglist

class pymisp.**MISPWarninglist**(***kwargs*)

clear() → None. Remove all items from D.

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*)
Dump recursively any class of type MISPAbstract to a json string

Return type str

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

CHAPTER

FOUR

PYMISP - TOOLS

4.1 File Object

class pymisp.tools.**FileObject**(*filepath=None*, *pseudofile=None*, *filename=None*, ***kwargs*)

add_attribute(*object_relation*, *simple_value=None*, ***value*)
Add an attribute. object_relation is required and the value key is a dictionary with all the keys supported by MISPAttribute

Return type Optional[*MISPAttribute*]

add_attributes(*object_relation*, **attributes*)
Add multiple attributes with the same object_relation. Helper for object_relation when multiple is True in the template. It is the same as calling multiple times add_attribute with the same object_relation.

Return type List[Optional[*MISPAttribute*]]

add_reference(*referenced_uuid*, *relationship_type*, *comment=None*, ***kwargs*)
Add a link (uuid) to another object

Return type *MISPObjectReference*

clear() → None. Remove all items from D.

delete()
Mark the object as deleted (soft delete)

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

generate_attributes()
Contains the logic where all the values of the object are gathered

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

get_attributes_by_relation(*object_relation*)
Returns the list of attributes with the given object relation in the object

Return type List[*MISPAttribute*]

has_attributes_by_relation(*list_of_relations*)
True if all the relations in the list are defined in the object

Return type bool

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*, *strict=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*, *strict=False*)
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

4.2 ELF Object

class pymisp.tools.**ELFObject**(*parsed=None*, *filepath=None*, *pseudofile=None*, ***kwargs*)

add_attribute(*object_relation*, *simple_value=None*, ***value*)
Add an attribute. object_relation is required and the value key is a dictionary with all the keys supported by MISPAttribute

Return type Optional[*MISPAttribute*]

add_attributes(*object_relation*, **attributes*)
Add multiple attributes with the same object_relation. Helper for object_relation when multiple is True in the template. It is the same as calling multiple times add_attribute with the same object_relation.

Return type List[Optional[*MISPAttribute*]]

add_reference(*referenced_uuid*, *relationship_type*, *comment=None*, ***kwargs*)
Add a link (uuid) to another object

Return type *MISPObjectReference*

clear() → None. Remove all items from D.

delete()
Mark the object as deleted (soft delete)

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

generate_attributes()
Contains the logic where all the values of the object are gathered

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

get_attributes_by_relation(*object_relation*)
Returns the list of attributes with the given object relation in the object

Return type List[*MISPAttribute*]

has_attributes_by_relation(*list_of_relations*)
True if all the relations in the list are defined in the object

Return type bool

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*, *strict=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*, *strict=False*)
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

class pymisp.tools.**ELFSectionObject**(*section*, ***kwargs*)

add_attribute(*object_relation*, *simple_value=None*, ***value*)
Add an attribute. object_relation is required and the value key is a dictionary with all the keys supported by MISPAttribute

Return type Optional[*MISPAttribute*]

add_attributes(*object_relation*, **attributes*)
Add multiple attributes with the same object_relation. Helper for object_relation when multiple is True in the template. It is the same as calling multiple times add_attribute with the same object_relation.

Return type List[Optional[*MISPAttribute*]]

add_reference(*referenced_uuid*, *relationship_type*, *comment=None*, ***kwargs*)
Add a link (uuid) to another object

Return type *MISPObjectReference*

clear() → None. Remove all items from D.

delete()
Mark the object as deleted (soft delete)

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

generate_attributes()
Contains the logic where all the values of the object are gathered

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

get_attributes_by_relation(*object_relation*)
Returns the list of attributes with the given object relation in the object

Return type List[*MISPAttribute*]

has_attributes_by_relation(*list_of_relations*)
True if all the relations in the list are defined in the object

Return type bool

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*, *strict=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*, *strict=False*)
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

4.3 PE Object

class pymisp.tools.**PEObject**(*parsed=None*, *filepath=None*, *pseudofile=None*, ***kwargs*)

add_attribute(*object_relation*, *simple_value=None*, ***value*)
Add an attribute. object_relation is required and the value key is a dictionary with all the keys supported by MISPAttribute

Return type Optional[*MISPAttribute*]

add_attributes(*object_relation*, **attributes*)
Add multiple attributes with the same object_relation. Helper for object_relation when multiple is True in the template. It is the same as calling multiple times add_attribute with the same object_relation.

Return type List[Optional[*MISPAttribute*]]

add_reference(*referenced_uuid*, *relationship_type*, *comment=None*, ***kwargs*)
Add a link (uuid) to another object

Return type *MISPObjectReference*

clear() → None. Remove all items from D.

delete()
Mark the object as deleted (soft delete)

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

generate_attributes()
Contains the logic where all the values of the object are gathered

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

get_attributes_by_relation(*object_relation*)
Returns the list of attributes with the given object relation in the object

Return type List[*MISPAttribute*]

has_attributes_by_relation(*list_of_relations*)
True if all the relations in the list are defined in the object

Return type bool

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*, *strict=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*, *strict=False*)
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

class pymisp.tools.**PESectionObject**(*section*, ***kwargs*)

add_attribute(*object_relation*, *simple_value=None*, ***value*)
Add an attribute. object_relation is required and the value key is a dictionary with all the keys supported by MISPAttribute

Return type Optional[*MISPAttribute*]

add_attributes(*object_relation*, **attributes*)
Add multiple attributes with the same object_relation. Helper for object_relation when multiple is True in the template. It is the same as calling multiple times add_attribute with the same object_relation.

Return type List[Optional[*MISPAttribute*]]

add_reference(*referenced_uuid*, *relationship_type*, *comment=None*, ***kwargs*)
Add a link (uuid) to another object

Return type *MISPObjectReference*

clear() → None. Remove all items from D.

delete()
Mark the object as deleted (soft delete)

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

generate_attributes()
Contains the logic where all the values of the object are gathered

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

get_attributes_by_relation(*object_relation*)
Returns the list of attributes with the given object relation in the object

Return type List[*MISPAttribute*]

has_attributes_by_relation(*list_of_relations*)
True if all the relations in the list are defined in the object

Return type bool

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*, *strict=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*, *strict=False*)
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

4.4 Mach-O Object

class pymisp.tools.**MachOObject**(*parsed=None*, *filepath=None*, *pseudofile=None*, ***kwargs*)

add_attribute(*object_relation*, *simple_value=None*, ***value*)
Add an attribute. object_relation is required and the value key is a dictionary with all the keys supported by MISPAttribute

Return type Optional[*MISPAttribute*]

add_attributes(*object_relation*, **attributes*)
Add multiple attributes with the same object_relation. Helper for object_relation when multiple is True in the template. It is the same as calling multiple times add_attribute with the same object_relation.

Return type List[Optional[*MISPAttribute*]]

add_reference(*referenced_uuid*, *relationship_type*, *comment=None*, ***kwargs*)
Add a link (uuid) to another object

Return type *MISPObjectReference*

clear() → None. Remove all items from D.

delete()
Mark the object as deleted (soft delete)

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

generate_attributes()
Contains the logic where all the values of the object are gathered

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

get_attributes_by_relation(*object_relation*)
Returns the list of attributes with the given object relation in the object

Return type List[*MISPAttribute*]

has_attributes_by_relation(*list_of_relations*)
True if all the relations in the list are defined in the object

Return type bool

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*, *strict=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*, *strict=False*)
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

class pymisp.tools.**MachOSectionObject**(*section*, ***kwargs*)

add_attribute(*object_relation*, *simple_value=None*, ***value*)
Add an attribute. object_relation is required and the value key is a dictionary with all the keys supported by MISPAttribute

Return type Optional[*MISPAttribute*]

add_attributes(*object_relation*, **attributes*)
Add multiple attributes with the same object_relation. Helper for object_relation when multiple is True in the template. It is the same as calling multiple times add_attribute with the same object_relation.

Return type List[Optional[*MISPAttribute*]]

add_reference(*referenced_uuid*, *relationship_type*, *comment=None*, ***kwargs*)
Add a link (uuid) to another object

Return type *MISPObjectReference*

clear() → None. Remove all items from D.

delete()
Mark the object as deleted (soft delete)

property edited: bool
Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

Return type None

generate_attributes()
Contains the logic where all the values of the object are gathered

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

get_attributes_by_relation(*object_relation*)
Returns the list of attributes with the given object relation in the object

Return type List[*MISPAttribute*]

has_attributes_by_relation(*list_of_relations*)
True if all the relations in the list are defined in the object

Return type bool

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

jsonable()
This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*, *strict=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*, *strict=False*)
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

4.5 VT Report Object

class pymisp.tools.**VTReportObject**(*apikey*, *indicator*, *vt_proxies=None*, ***kwargs*)

VirusTotal Report

Apikey VirusTotal API key (private works, but only public features are supported right now)

Indicator IOC to search VirusTotal for

add_attribute(*object_relation*, *simple_value=None*, ***value*)

Add an attribute. object_relation is required and the value key is a dictionary with all the keys supported by MISPAttribute

Return type Optional[*MISPAttribute*]

add_attributes(*object_relation*, **attributes*)

Add multiple attributes with the same object_relation. Helper for object_relation when multiple is True in the template. It is the same as calling multiple times add_attribute with the same object_relation.

Return type List[Optional[*MISPAttribute*]]

add_reference(*referenced_uuid*, *relationship_type*, *comment=None*, ***kwargs*)

Add a link (uuid) to another object

Return type *MISPObjectReference*

clear() → None. Remove all items from D.

delete()

Mark the object as deleted (soft delete)

property edited: bool

Recursively check if an object has been edited and update the flag accordingly to the parent objects

Return type bool

from_dict(***kwargs*)

Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)

Load a JSON string

Return type None

generate_attributes()

Parse the VirusTotal report for relevant attributes

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

get_attributes_by_relation(*object_relation*)

Returns the list of attributes with the given object relation in the object

Return type List[*MISPAttribute*]

has_attributes_by_relation(*list_of_relations*)

True if all the relations in the list are defined in the object

Return type bool

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

jsonable()

This method is used by the JSON encoder

Return type Dict

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

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_not_jsonable(*args*)
Set __not_jsonable to a new list

Return type None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*json_format=False*, *strict=False*)
Dump the class to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

Return type Dict

to_json(*sort_keys=False*, *indent=None*, *strict=False*)
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

Return type None

values() → an object providing a view on D's values

4.6 STIX

pymisp.tools.stix.**load_stix**(*stix*, *distribution=3*, *threat_level_id=2*, *analysis=0*)
Returns a MISPEvent object from a STIX package

pymisp.tools.stix.**make_stix_package**(*misp_event*, *to_json=False*, *to_xml=False*)
Returns a STIXPackage from a MISPEvent.

Optionally can return the package in json or xml.

4.7 OpenIOC

tools.**load_openioc**()

tools.**load_openioc_file**()

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W