mwdblib

Release 4.5.0

CERT Polska

Jul 21, 2023

API DOCUMENTATION

If you want to learn about mwdblib usage, check out 8. Automating things using REST API and mwdblib in mwdb-core User Guide.

CHAPTER

ONE

CORE INTERFACE (MWDB)

`class mwdblib.MWDB`(*api: Optional[*mwdblib.api.api.APIClient*] = None*, ***api_options: Any*)

Main object used for communication with MWDB REST API

Parameters

- **`api_url`** – MWDB API URL (that ends with '/api/').
- **`api_key`** – MWDB API key
- **`username`** – MWDB account username
- **`password`** – MWDB account password
- **`autologin`** – Login automatically using credentials stored in configuration or provided in arguments (default: True)
- **`verify_ssl`** – Verify SSL certificate correctness (default: True)
- **`obey_ratelimiter`** – If `False`, HTTP 429 errors will cause an exception like all other error codes. If `True`, library will transparently handle them by sleeping for a specified duration. Default is `True`.
- **`retry_on_downtime`** – If `True`, requests will be automatically retried after `downtime_timeout` seconds on HTTP 502/504 and ConnectionError. Default is `False`.
- **`max_downtime_retries`** – Number of retries caused by temporary downtime
- **`downtime_timeout`** – How long we need to wait between retries (in seconds). Default is 10.
- **`retry_idempotent`** – Retry idempotent POST requests (default). The only thing that is really non-idempotent in current API is *`MWDBObject.add_comment()`*, so it's not a big deal. You can turn it off if possible doubled comments are problematic in your MWDB instance. Default is `True`.
- **`use_keyring`** – If `True`, APIClient uses keyring to fetch stored credentials. If not, they're fetched from plaintext configuration. Default is `True`.
- **`emit_warnings`** – If `True`, warnings are emitted by APIClient. Default is `True`.
- **`config_path`** – Path to the configuration file (default is *~/.mwdb*). If None, configuration file will not be used by APIClient
- **`api`** (*`mwdblib.APIClient`*, optional) – Custom *`APIClient`* to be used for communication with MWDB

New in version 2.6.0: API request will sleep for a dozen of seconds when rate limit has been exceeded.

New in version 3.2.0: You can enable `retry_on_downtime` to automatically retry requests in case of HTTP 502/504 or ConnectionError.

Changed in version 4.0.0: *MWDB* by default uses credentials and api_url set by *mwdb login*. If you don't want to automatically fetch them from configuration, pass *config_path=None* to the constructor

New in version 4.0.0: Added `use_keyring`, `emit_warnings` and `config_path` options. `username` and `password` can be passed directly to the constructor.

New in version 4.4.0: Added `autologin` option.

Usage example:

```
from mwdblib import MWDB

mwdb = MWDB()
mwdb.login("example", "<password>")

file = mwdb.query_file("3629344675705286607dd0f680c66c19f7e310a1")
```

count(*query: Optional[str] = None*) → int

Returns number of objects matching provided query in Lucene syntax. If you already know type of objects you want to count, use specialized variants:

- *count_files()*
- *count_configs()*
- *count_blobs()*

Usage example:

```
from mwdblib import MWDB

mwdb = MWDB()

# Count samples tagged as evil and with size less than 100kB
result = mwdb.count_files("tag:evil AND file.size:[0 TO 100000]")
```

Parameters **query** (`str, optional`) – Query in Lucene syntax

Return type int

Raises requests.exceptions.HTTPError

count_blobs(*query: Optional[str] = None*) → int

Returns number of blobs matching provided query in Lucene syntax.

Parameters **query** (`str, optional`) – Query in Lucene syntax

Return type int

Raises requests.exceptions.HTTPError

count_configs(*query: Optional[str] = None*) → int

Returns number of configs matching provided query in Lucene syntax.

Parameters **query** (`str, optional`) – Query in Lucene syntax

Return type int

Raises requests.exceptions.HTTPError

count_files(*query: Optional[str] = None*) → int

Returns number of files matching provided query in Lucene syntax.

Parameters **query** (*str, optional*) – Query in Lucene syntax

Return type int

Raises requests.exceptions.HTTPError

listen_for_blobs(*last_object: Optional[Union[*mwdblib.blob.MWDBBlob*, str]] = None*, *blocking: bool = True*, *interval: int = 15*, *query: Optional[str] = None*) → Iterator[*mwdblib.blob.MWDBBlob*]

Listens for recent blobs and yields newly added.

See also:

More details can be found here: `listen_for_objects()`

New in version 3.2.0: Added listen_for_* methods

New in version 3.4.0: Added query parameter

New in version 3.4.0: The listen_for_* methods will now try to prevent you from iterating over the whole database by throwing an exception if they detect that there is something wrong with the pivot object

Parameters

- **last_object** (MWDBBlob *or str*) – MWDBBlob instance or object hash
- **blocking** (*bool, optional*) – Enable blocking mode (default)
- **interval** (*int, optional*) – Interval for periodic queries in blocking mode (default is 15 seconds)
- **query** (*str, optional*) – Lucene query to be used for listening for only specific blobs

Return type Iterator[*MWDBBlob*]

listen_for_configs(*last_object: Optional[Union[*mwdblib.config.MWDBConfig*, str]] = None*, *blocking: bool = True*, *interval: int = 15*, *query: Optional[str] = None*) → Iterator[*mwdblib.config.MWDBConfig*]

Listens for recent configs and yields newly added.

See also:

More details can be found here: `listen_for_objects()`

New in version 3.2.0: Added listen_for_* methods

New in version 3.4.0: Added query parameter

New in version 3.4.0: The listen_for_* methods will now try to prevent you from iterating over the whole database by throwing an exception if they detect that there is something wrong with the pivot object

Parameters

- **last_object** (MWDBConfig *or str*) – MWDBConfig instance or object hash
- **blocking** (*bool, optional*) – Enable blocking mode (default)
- **interval** (*int, optional*) – Interval for periodic queries in blocking mode (default is 15 seconds)
- **query** (*str, optional*) – Lucene query to be used for listening for only specific configs

Return type Iterator[*MWDBConfig*]

listen_for_files(*last_object: Optional[Union[*mwdblib.file.MWDBFile*, str]] = None*, *blocking: bool = True*, *interval: int = 15*, *query: Optional[str] = None*) → Iterator[*mwdblib.file.MWDBFile*]

Listens for recent files and yields newly added.

See also:

More details can be found here: `listen_for_objects()`

New in version 3.2.0: Added listen_for_* methods

New in version 3.4.0: Added query parameter

New in version 3.4.0: The listen_for_* methods will now try to prevent you from iterating over the whole database by throwing an exception if they detect that there is something wrong with the pivot object

Parameters

- **last_object** (`MWDBFile or str`) – MWDBFile instance or object hash
- **blocking** (*bool, optional*) – Enable blocking mode (default)
- **interval** (*int, optional*) – Interval for periodic queries in blocking mode (default is 15 seconds)
- **query** (*str, optional*) – Lucene query to be used for listening for only specific files

Return type Iterator[*MWDBFile*]

listen_for_objects(*last_object: Optional[Union[*mwdblib.object.MWDBObject*, str]] = None*, *blocking: bool = True*, *interval: int = 15*, *query: Optional[str] = None*) → Iterator[*mwdblib.object.MWDBObject*]

Listens for recent objects and yields newly added.

In blocking mode (default) if last_object is provided: the method fetches the latest objects until the provided object is reached and yields new objects from the oldest one. Otherwise, the method periodically asks for recent objects until a new object appears. The default request interval is 15 seconds.

In a non-blocking mode: a generator stops if there are no more objects to fetch.

last_object argument accepts both identifier and MWDBObject instance. If the object identifier is provided: method firstly checks whether the object exists in repository and has the correct type.

If you already know type of object you are looking for, use specialized variants:

- `listen_for_files()`
- `listen_for_configs()`
- `listen_for_blobs()`

Warning: Make sure that last_object is valid in MWDB instance. If you provide MWDBObject that doesn't exist, mwdblib will iterate over all objects and you can quickly hit your rate limit. Library is trying to protect you from that as much as possible by checking type and object existence, but it's still possible to do something unusual.

Additionally, if using the `query` parameter and passing the `last_object` pivot, make sure that the passed object actually matches the query criteria. Otherwise the mechanism that catches faulty pivots will signal that there's something wrong and raise an exception.

New in version 3.2.0: Added listen_for_* methods

New in version 3.4.0: Added query parameter

New in version 3.4.0: The listen_for_* methods will now try to prevent you from iterating over the whole database by throwing an exception if they detect that there is something wrong with the pivot object

Parameters

- **`last_object`** (`MWDBObject` *or* `str`) – MWDBObject instance or object hash
- **`blocking`** (*`bool`*, *`optional`*) – Enable blocking mode (default)
- **`interval`** (*`int`*, *`optional`*) – Interval for periodic queries in blocking mode (default is 15 seconds)
- **`query`** (*`str`*, *`optional`*) – Lucene query to be used for listening for only specific objects

Return type Iterator[*`MWDBObject`*]

`login`(*username: Optional[str] = None*, *password: Optional[str] = None*) → None

Performs user authentication using provided username and password.

If credentials are not set, asks interactively for credentials.

Warning: Keep in mind that password-authenticated sessions are short-lived, so password needs to be stored in *`APIClient`* object. Consider generating a new API key in your MWDB profile.

New in version 2.4.0: MWDB tries to reauthenticate on first Unauthorized exception

New in version 2.5.0: username and password arguments are optional. If one of the credentials is not provided via arguments, user will be asked for it.

New in version 2.6.0: *`MWDB.login()`* will warn if login is called after setting up API key

Changed in version 4.0.0: *`MWDB.login()`* no longer warns about password-authenticated sessions or credentials that are already set up.

Parameters

- **`username`** (*`str`*) – Username
- **`password`** (*`str`*) – Password

Raises requests.exceptions.HTTPError

`logout`() → None

Performs session logout and removes previously set API key.

`property options:` *`APIClientOptions`*

Returns object with current configuration of MWDB client

New in version 4.0.0: Added MWDB.options property.

`query`(*hash: str*, *raise_not_found: bool = True*) → Optional[*mwdblib.object.MWDBObject*]

Queries for object using provided hash. If you already know type of object you are looking for, use specialized variants:

- *`query_file()`*
- *`query_config()`*
- *`query_blob()`*

New in version 2.4.0: Added raise_not_found optional argument

Changed in version 3.0.0: Fallback to *`query_file()`* if other hash than SHA256 was provided

Parameters

- **`hash`** (*`str`*) – Object hash (identifier, MD5, SHA-1, SHA-2)
- **`raise_not_found`** (*`bool, optional`*) – If True (default), method raises HTTPError when object is not found

Return type *`MWDBObject`* or None (if raise_not_found=False)

Raises requests.exceptions.HTTPError

`query_blob`(*hash: str*, *raise_not_found: bool = True*) → Optional[*mwdblib.blob.MWDBBlob*]

Queries for blob object using provided hash

Parameters

- **`hash`** (*`str`*) – Object hash (SHA-256 identifier)
- **`raise_not_found`** (*`bool`*) – If True (default), method raises HTTPError when object is not found

Return type *`MWDBBlob`* or None (if raise_not_found=False)

Raises requests.exceptions.HTTPError

`query_config`(*hash: str*, *raise_not_found: bool = True*) → Optional[*mwdblib.config.MWDBConfig*]

Queries for configuration object using provided hash

Parameters

- **`hash`** (*`str`*) – Object hash (SHA-256 identifier)
- **`raise_not_found`** (*`bool`*) – If True (default), method raises HTTPError when object is not found

Return type *`MWDBConfig`* or None (if raise_not_found=False)

Raises requests.exceptions.HTTPError

`query_file`(*hash: str*, *raise_not_found: bool = True*) → Optional[*mwdblib.file.MWDBFile*]

Queries for file using provided hash

Parameters

- **`hash`** (*`str`*) – Object hash (identifier, MD5, SHA-1, SHA-2)
- **`raise_not_found`** (*`bool`*) – If True (default), method raises HTTPError when object is not found

Return type *`MWDBFile`* or None (if raise_not_found=False)

Raises requests.exceptions.HTTPError

`recent_blobs`(*chunk_size: Optional[int] = None*) → Iterator[*mwdblib.blob.MWDBBlob*]

Retrieves recently uploaded blob objects

Parameters **`chunk_size`** (*`int`*) – Number of blobs returned per API request

Return type Iterator[*`MWDBBlob`*]

Raises requests.exceptions.HTTPError

`recent_configs`(*chunk_size: Optional[int] = None*) → Iterator[*mwdblib.config.MWDBConfig*]

Retrieves recently uploaded configuration objects

Parameters **`chunk_size`** (*`int`*) – Number of configs returned per API request

Return type Iterator[*`MWDBConfig`*]

Raises requests.exceptions.HTTPError

recent_files(*chunk_size: Optional[int] = None*) → Iterator[*mwdblib.file.MWDBFile*]
Retrieves recently uploaded files

Parameters **chunk_size** (*int*) – Number of files returned per API request

Return type Iterator[*MWDBFile*]

Raises requests.exceptions.HTTPError

recent_objects(*chunk_size: Optional[int] = None*) → Iterator[*mwdblib.object.MWDBObject*]
Retrieves recently uploaded objects If you already know type of object you are looking for, use specialized variants:

- *recent_files()*
- *recent_configs()*
- *recent_blobs()*

Usage example:

```
from mwdblib import MWDB
from itertools import islice

mwdb = MWDB()
mwdb.login("admin", "password123")

# recent_files is generator, do not execute list(recent_files)!
files = islice(mwdb.recent_files(), 25)
print([(f.name, f.tags) for f in files])
```

Parameters **chunk_size** (*int*) – Number of objects returned per API request

Return type Iterator[*MWDBObject*]

Raises requests.exceptions.HTTPError

search(*query: str*, *chunk_size: Optional[int] = None*) → Iterator[*mwdblib.object.MWDBObject*]
Advanced search for objects using Lucene syntax. If you already know type of objects you are looking for, use specialized variants:

- *search_files()*
- *search_configs()*
- *search_blobs()*

Usage example:

```
from mwdblib import MWDB

# Search for samples tagged as evil and with size less than 100kB
results = mwdb.search_files("tag:evil AND file.size:[0 TO 100000]")
```

Parameters

- **query** (*str*) – Search query
- **chunk_size** (*int*) – Number of objects returned per API request

Return type Iterator[*MWDBObject*]

Raises requests.exceptions.HTTPError

search_blobs(*query: str*, *chunk_size: Optional[int] = None*) → Iterator[*mwdblib.blob.MWDBBlob*]

Advanced search for blob objects using Lucene syntax.

Parameters

- **query** (*str*) – Search query
- **chunk_size** (*int*) – Number of blobs returned per API request

Return type Iterator[*MWDBBlob*]

Raises requests.exceptions.HTTPError

search_configs(*query: str*, *chunk_size: Optional[int] = None*) → Iterator[*mwdblib.config.MWDBConfig*]

Advanced search for configuration objects using Lucene syntax.

Parameters

- **query** (*str*) – Search query
- **chunk_size** (*int*) – Number of configs returned per API request

Return type Iterator[*MWDBConfig*]

Raises requests.exceptions.HTTPError

search_files(*query: str*, *chunk_size: Optional[int] = None*) → Iterator[*mwdblib.file.MWDBFile*]

Advanced search for files using Lucene syntax.

Parameters

- **query** (*str*) – Search query
- **chunk_size** (*int*) – Number of files returned per API request

Return type Iterator[*MWDBFile*]

Raises requests.exceptions.HTTPError

upload_blob(*name: str*, *type: str*, *content: str*, *parent: Optional[Union[*mwdblib.object.MWDBObject*, str]] = None*, *metakeys: Optional[Dict[str, Union[str, List[str]]]] = None*, *attributes: Optional[Dict[str, Union[Any, List[Any]]]] = None*, *karton_id: Optional[str] = None*, *karton_arguments: Optional[Dict[str, str]] = None*, *tags: Optional[List[str]] = None*, *share_with: Optional[str] = None*, *private: bool = False*, *public: bool = False*) → *mwdblib.blob.MWDBBlob*

Upload blob object

Parameters

- **name** (*str*) – Blob name (see also *MWDBBlob.blob_name*)
- **type** (*str*) – Blob type (see also *MWDBBlob.blob_type*)
- **content** (*str*) – Blob content (see also *MWDBBlob.content*)
- **parent** (*MWDBObject* or str, optional) – Parent object or parent identifier
- **metakeys** (*dict, optional*) – Dictionary with string attributes (to be used for MWDB Core older than 2.6.0)
- **attributes** (*dict, optional*) – Dictionary with attributes to be set after upload. If you want to set many values with the same key: use list as value. Attributes support object values that are JSON-serializable.

- **karton_id** (*str, optional*) – Karton analysis identifier to be attached to the uploaded file
- **karton_arguments** (*dict, optional*) – Karton analysis arguments. Reserved for future.
- **tags** (*list, optional*) – Dictionary with tags to be set after upload.
- **share_with** (*str, optional*) – Group name you want to share object with
- **private** (*bool, optional*) – True if sample should be uploaded as private
- **public** (*bool, optional*) – True if sample should be visible for everyone

Return type *MWDBBlob*

New in version 4.0.0: Added `attributes` and `tags` arguments. They are supported by MWDB Core >= 2.6.0, use `metakeys` if your MWDB Core version is older.

New in version 4.1.0: Added `karton_id` and `karton_arguments` parameters. Use `karton_id` instead of `metakeys={"karton": "<id>"}` if you use MWDB Core >= 2.3.0

upload_config(*family: str*, *cfg: Dict[str, Any]*, *config_type: str = 'static'*, *parent: Optional[Union[*mwdblib.object.MWDBObject*, str]] = None*, *metakeys: Optional[Dict[str, Union[str, List[str]]]] = None*, *attributes: Optional[Dict[str, Union[Any, List[Any]]]] = None*, *karton_id: Optional[str] = None*, *karton_arguments: Optional[Dict[str, str]] = None*, *tags: Optional[List[str]] = None*, *share_with: Optional[str] = None*, *private: bool = False*, *public: bool = False*) → *mwdblib.config.MWDBConfig*

Upload configuration object

Parameters

- **family** (*str*) – Malware family name (see also *MWDBConfig.family*)
- **cfg** (*dict*) – Dict object with configuration (see also *MWDBConfig.cfg*)
- **config_type** (*str, optional*) – Configuration type (default: static, see also *MWDBConfig.type*)
- **parent** (*MWDBObject* or str, optional) – Parent object or parent identifier
- **metakeys** (*dict, optional*) – Dictionary with string attributes (to be used for MWDB Core older than 2.6.0)
- **attributes** (*dict, optional*) – Dictionary with attributes to be set after upload. If you want to set many values with the same key: use list as value. Attributes support object values that are JSON-serializable.
- **karton_id** (*str, optional*) – Karton analysis identifier to be attached to the uploaded file
- **karton_arguments** (*dict, optional*) – Karton analysis arguments. Reserved for future.
- **tags** (*list, optional*) – Dictionary with tags to be set after upload.
- **share_with** (*str, optional*) – Group name you want to share object with
- **private** (*bool, optional*) – True if sample should be uploaded as private
- **public** (*bool, optional*) – True if sample should be visible for everyone

Return type *MWDBConfig*

New in version 4.0.0: Added `attributes` and `tags` arguments. They are supported by MWDB Core >= 2.6.0, use `metakeys` if your MWDB Core version is older.

New in version 4.1.0: Added `karton_id` and `karton_arguments` parameters. Use `karton_id` instead of `metakeys={"karton": "<id>"}` if you use MWDB Core >= 2.3.0

```
mwdb.upload_config(
    "evil",
    {
        "botnet": "mal0123",
        "version": 2019,
        "urls": [
            "http://example.com",
            "http://example.com/2"
        ]
    }
    parent="3629344675705286607dd0f680c66c19f7e310a1",
    public=True)
```

upload_file(*name: str*, *content: Union[bytes, BinaryIO]*, *parent: Optional[Union[mwdblib.object.MWDBObject, str]] = None*, *metakeys: Optional[Dict[str, Union[str, List[str]]]] = None*, *attributes: Optional[Dict[str, Union[Any, List[Any]]]] = None*, *karton_id: Optional[str] = None*, *karton_arguments: Optional[Dict[str, str]] = None*, *tags: Optional[List[str]] = None*, *share_with: Optional[str] = None*, *private: bool = False*, *public: bool = False*) → *mwdblib.file.MWDBFile*

Upload file object

Parameters

- **name** (*str*) – Original file name (see also *MWDBFile.file_name*)
- **content** (*bytes or BinaryIO*) – File contents
- **parent** (*MWDBObject* or str, optional) – Parent object or parent identifier
- **metakeys** (*dict, optional*) – Dictionary with string attributes (to be used for MWDB Core older than 2.6.0)
- **attributes** (*dict, optional*) – Dictionary with attributes to be set after upload. If you want to set many values with the same key: use list as value. Attributes support object values that are JSON-serializable.
- **karton_id** (*str, optional*) – Karton analysis identifier to be attached to the uploaded file
- **karton_arguments** (*dict, optional*) – Karton analysis arguments. Reserved for future.
- **tags** (*list, optional*) – Dictionary with tags to be set after upload.
- **share_with** (*str, optional*) – Group name you want to share object with
- **private** (*bool, optional*) – True if sample should be uploaded as private
- **public** (*bool, optional*) – True if sample should be visible for everyone

Return type *MWDBFile*

New in version 4.0.0: Added `attributes` and `tags` arguments. They are supported by MWDB Core >= 2.6.0, use `metakeys` if your MWDB Core version is older.

New in version 4.1.0: Added `karton_id` and `karton_arguments` parameters. Use `karton_id` instead of `metakeys={"karton": "<id>"}` if you use MWDB Core >= 2.3.0

Usage example:

```
mwdb.upload_file(
    "malware.exe",
    open("malware.exe", "rb").read(),
    parent="3629344675705286607dd0f680c66c19f7e310a1",
    public=True)
```

class `mwdblib.`**`APIClient`**(*_auth_token: Optional[str] = None*, *autologin: bool = True*, ***api_options: Any*)

Client for MWDB REST API that performs authentication and low-level API request/response handling.

If you want to send arbitrary request to MWDB API, use `get()`, `post()`, `put()` and `delete()` methods using `MWDB.api` property.

```
mwdb = MWDB()
...
# Deletes object with given sha256
mwdb.api.delete(f'object/{sha256}')
```

property logged_user: Optional[str]

Username of logged-in user or the owner of used API key. Returns None if no credentials are provided

`login`(*username: str*, *password: str*) → None

Performs authentication using provided credentials

Parameters

- **username** – Account username
- **password** – Account password

`logout`() → None

Removes authorization token from APIClient instance

`request`(*method: str*, *url: str*, *noauth: bool = False*, *raw: bool = False*, **args: Any*, ***kwargs: Any*) → Any

Sends request to MWDB API. This method can be used for accessing features that are not directly supported by mwdblib library.

Other keyword arguments are the same as in requests library.

See also:

Use functions specific for HTTP methods instead of passing `method` argument on your own:

- `APIClient.get()`
- `APIClient.post()`
- `APIClient.put()`
- `APIClient.delete()`

Parameters

- **method** – HTTP method
- **url** – Relative url of API endpoint
- **noauth** –

Don't check if user is authenticated before sending request (default: False)

- `raw` – Return raw response bytes instead of parsed JSON (default: False)

`static requires`(*required_version: str*, *always_check_version: bool = False*) → Callable

Method decorator that provides server version requirement and fallback to older implementation if available.

To optimize requests sent by CLI: first method is called always if server version is not already available. If it fails with EndpointNotFoundError, server version is fetched and used to determine if fallback is available.

If your method fails on something different than missing endpoint, you can check version always by enabling `always_check_version` flag.

`property server_metadata: dict`

Information about MWDB Core server from `/api/server` endpoint.

`property server_version: str`

MWDB Core server version

`set_api_key`(*api_key: str*) → None

Sets API key to be used for authorization

Parameters `api_key` – API key to set

`supports_version`(*required_version: str*) → bool

Checks if server version is higher or equal than provided.

New in version 4.1.0.

`class mwdblib.APIClientOptions`(*config_path: Optional[pathlib.Path] = PosixPath('/home/docs/.mwdb')*, ***api_options: Any*)

Options bag that contains configuration for APIClient.

Field values are loaded using the following precedence:

- built-in defaults accessible via class properties e.g. `APIClientOptions.api_url`
- values from `~/.mwdb` configuration file
- values passed as an arguments to the `APIClientOptions` constructor

Configuration may depend on `api_url` value, so remember to set it if you want to talk with specific MWDB Core instance.

`clear_stored_credentials`(*config_writeback: bool = True*) → bool

Clears stored credentials in configuration for current user.

Used by `mwdb logout` CLI command.

`store_credentials`(*username: Optional[str]*, *password: Optional[str]*, *api_key: Optional[str]*) → bool

Stores current credentials in configuration for current user.

Used by `mwdb login` CLI command.

CHAPTER

TWO

OBJECT TYPES

class mwdblib.**MWDBObject**(*api: mwdblib.api.api.APIClient*, *data: Dict[str, Any]*)

Represents abstract, generic MWDB object.

Should never be instantiated directly.

If you really need to get synthetic instance - use internal `create()` static method.

add_attribute(*key: str*, *value: Any*) → None

Adds attribute.

Key can't be 'karton'. If you want to assign an analysis, use `assign_analysis()` instead or use `add_metakey()` method.

New in version 4.0.0.

Parameters

- **key** (*str*) – Attribute key
- **value** (*Any (JSON-like object)*) – Attribute value

add_child(*child: Union[mwdblib.object.MWDBObject, str]*) → None

Adds reference to child with current object as parent

Parameters **child** (*MWDBObject or str*) – Object or object identifier (sha256)

add_comment(*comment: str*) → None

Adds comment

Parameters **comment** (*str*) – Comment string

add_metakey(*key: str*, *value: str*) → None

Adds metakey attribute (string only)

Deprecated since version 4.0.0: Use `add_attribute` instead

Parameters

- **key** (*str*) – Attribute key
- **value** (*str*) – Attribute value

add_tag(*tag: str*) → None

Tags object using specified tag

Parameters **tag** (*str*) – Tag string

property analyses: List[MWDBKartonAnalysis]

Returns list of Karton analyses related with this object

Requires MWDB Core >= 2.3.0.

New in version 4.0.0.

assign_analysis(*analysis_id: str*) → MWDBKartonAnalysis

Assigns object to existing Karton analysis

Requires MWDB Core >= 2.3.0.

Parameters **analysis_id** – Karton analysis UUID

New in version 4.0.0.

property attributes: Dict[str, List[Any]]

Returns dict object with attributes.

Supports JSON-like values in MWDB Core >= 2.6.0.

New in version 4.0.0.

Returns Dict object containing attributes

property children: List[*mwdblib.object.MWDBObject*]

Returns list of child objects

Returns List of child objects

property comments: List[*MWDBComment*]

Returns list of comments

Returns List of comment objects

Example - print all comments of last object commented as "malware":

```
comments = next(mwdb.search_files('comment:"*malware*"')).comments
for comment in comments:
    print("{} {}".format(comment.author, comment.comment))
```

property content: bytes

Returns stringified contents of object

New in version 3.0.0: Added `MWDBObject.content` property

static create(*api:* mwdblib.api.api.APIClient, *data: Dict[str, Any]*) → *mwdblib.object.MWDBObject*

Creates specialized MWDBObject subclass instance based on specified `data`

flush() → None

Flushes local object state in case of pending updates. All object-specific properties will be lazy-loaded using API

get_attributes_detailed() → List[Dict[str, Any]]

Returns list with attributes including identifiers needed to remove specific attribute.

Requires MWDB Core >= 2.6.0.

New in version 4.1.0.

Returns List containing detailed attributes

property id: str

Object identifier (sha256)

property metakeys: Dict[str, List[str]]

Returns dict object with metakeys.

JSON-like values are coerced to strings for backwards compatibility.

Deprecated since version 4.0.0: Use `attributes` instead

Returns Dict object containing metakey attributes

property object_type: str
Object type ('file', 'static_config' or 'text_blob')

property parents: List[*mwdblib.object.MWDBObject*]
Returns list of parent objects

Returns List of parent objects

reanalyze(*arguments: Optional[Dict[str, Any]] = None*) → MWDBKartonAnalysis
Submits new Karton analysis for given object.

Requires MWDB Core >= 2.3.0.

Parameters **arguments** –

Optional, additional arguments for analysis. Reserved for future functionality.

New in version 4.0.0.

remove() → None
Remove specific object from mwdb

The object should be treated as invalidated after using this method .

remove_attribute(*attribute_id: int*) → None
Remove specific attribute from object

See also:

IDs of attributes can be found here: *get_attributes_detailed()*

Parameters **attribute_id** (*int*) – Attribute id

remove_comment(*comment_id: int*) → None
Remove specific comment from object

Parameters **comment_id** (*int*) – Comment id

remove_tag(*tag: str*) → None
Untags object using specified tag

Parameters **tag** (*str*) – Tag string

property sha256: str
Object identifier (sha256)

share_with(*group: str*) → None
Share object with specified group

New in version 3.0.0: Added *MWDBObject.share_with()* method

Parameters **group** (*str*) – Group name

property shares: List[*MWDBShare*]
Returns list of shares

Returns List of share objects

property tags: List[str]
Returns list of tags

Returns List of tags

property upload_time: datetime.datetime
Returns timestamp of first object upload

Returns datetime object with object upload timestamp

class mwdblib.MWDBFile(*api: mwdblib.api.api.APIClient*, *data: MWDBElementData*)

property config: Optional[*MWDBConfig*]
Returns latest config related with this object

Return type *MWDBConfig* or None

Returns Latest configuration if found

property content: bytes
Returns file contents, calling *MWDBFile.download()* if contents were not loaded yet

download() → bytes
Downloads file contents

Returns File contents

Return type bytes

Example - download first file with size less than 1000 bytes and VBS extension

```
dropper = next(
    mwdb.search_files('file.size:[0 TO 1000] AND file.name:"*.vbs"')
)

with open(dropper.file_name, "wb") as f:
    f.write(dropper.download())

print("Downloaded {}".format(dropper.file_name))
```

property file_name: str
Sample original name

property file_size: int
Sample size in bytes

property file_type: str
Sample type

property name: str
Alias for *file_name* property

property size: int
Alias for *file_size* property

property type: str
Alias for *file_type* property

class mwdblib.MWDBConfig(*api: mwdblib.api.api.APIClient*, *data: Dict[str, Any]*)

property cfg: Dict[str, Any]
Raw dict object with configuration

See also:

config_dict

property config: Dict[str, Any]
dict object with configuration. In-blob keys are mapped to MWDBBlob objects.

property config_dict: Dict[str, Any]
Raw dict object with configuration (in-blob keys are not mapped to *MWDBBlob* objects)

property content: bytes
Returns raw dict object as JSON bytes

Return type bytes

property family: str
Configuration family

property type: str
Configuration type ('static' or 'dynamic')

class mwdblib.MWDBBlob(*api:* mwdblib.api.api.APIClient, *data: Dict[str, Any]*)

property blob_name: str
Blob name

property blob_size: int
Blob size in bytes

property blob_type: str
Blob semantic type

property config: Optional[*MWDBConfig*]
Returns latest config related with this object

Returns Latest configuration if found

property content: bytes
Contains blob content

Changed in version 3.0.0: Returned type is guaranteed to be utf8-encoded bytes

property last_seen: datetime.datetime

Returns datetime object when blob was last seen in MWDB

property name: str
Alias for *blob_name* property

property size: int
Alias for *blob_size* property

property type: str
Alias for *blob_type* property

CHAPTER

THREE

SECONDARY OBJECTS

class mwdblib.comment.**MWDBComment**(*api:* APIClient, *data: MWDBElementData*, *parent:* MWDBObject)

Represents comment for MWDB object

property author: str

Comment author

property comment: str

Comment text

delete() → None

Deletes this comment

property id: int

Comment identifier

property timestamp: datetime.datetime

Comment timestamp

class mwdblib.share.**MWDBShare**(*api:* APIClient, *data: MWDBElementData*, *parent:* mwdblib.object.MWDBObject)

Represents share entry in MWDB object

property group: str

Returns a group name that object is shared with

Returns Group name

property reason: *mwdblib.share.MWDBShareReason*

Returns why object was shared

property timestamp: datetime.datetime

Returns timestamp of share

Returns datetime object with object share timestamp

class mwdblib.share.**MWDBShareReason**(*api:* APIClient, *share_data: MWDBElementData*)

Represents the reason why object was shared with specified group

property what: *mwdblib.object.MWDBObject*

Returns what was shared

property who: str

Returns who caused action returned by *why* property.

Returns User login

property why: str

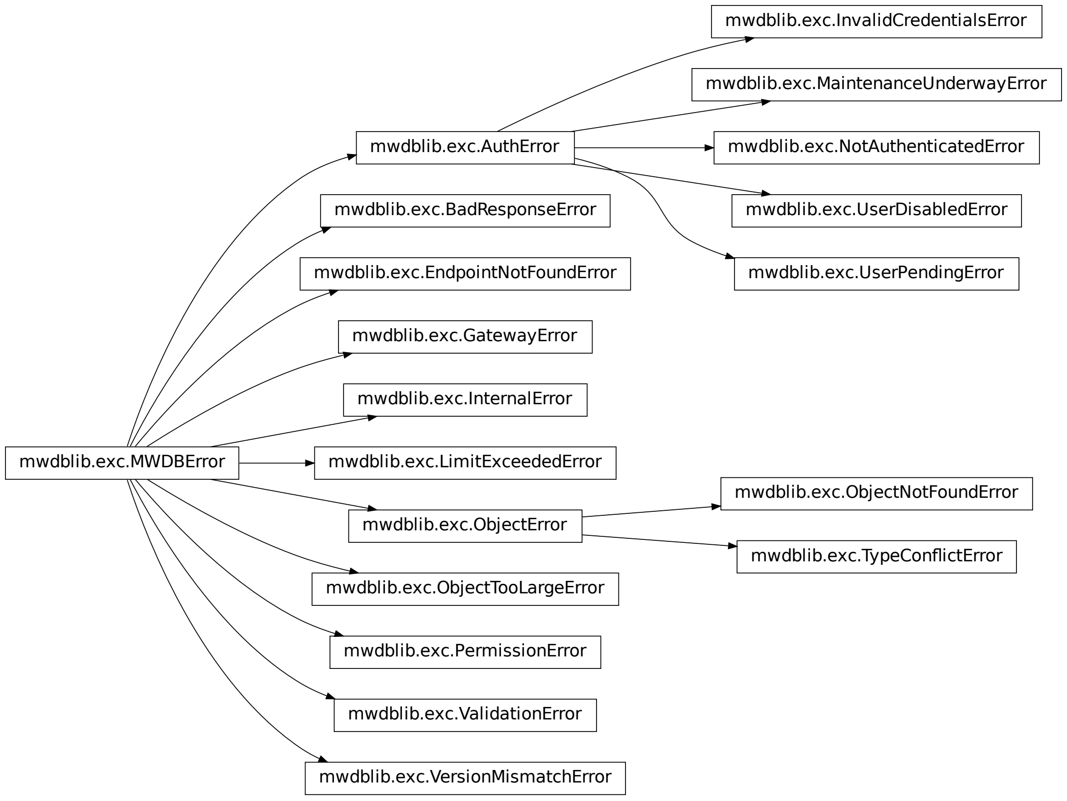
Returns why it was shared

Returns One of actions: ‘queried’, ‘shared’, ‘added’, ‘migrated’

CHAPTER

FOUR

EXCEPTION OBJECTS


mwdblib.exc.InvalidCredentialsError
mwdblib.exc.MaintenanceUnderwayError
mwdblib.exc.AuthError
mwdblib.exc.NotAuthenticatedError
mwdblib.exc.BadResponseError
mwdblib.exc.UserDisabledError
mwdblib.exc.EndpointNotFoundError
mwdblib.exc.UserPendingError
mwdblib.exc.GatewayError
mwdblib.exc.InternalError
mwdblib.exc.MWDBError
mwdblib.exc.LimitExceededError
mwdblib.exc.ObjectNotFoundError
mwdblib.exc.ObjectError
mwdblib.exc.TypeConflictError
mwdblib.exc.ObjectTooLargeError
mwdblib.exc.PermissionError
mwdblib.exc.ValidationError
mwdblib.exc.VersionMismatchError

class mwdblib.exc.**MWDBError**(*message: Optional[str] = None*, *http_error: Optional[requests.exceptions.HTTPError] = None*)

Generic class for MWDB exceptions

Parameters

- **message** (*str*) – Error message
- **http_error** (`requests.exceptions.HTTPError`) – Original HTTP error

class `mwdblib.exc.AuthError`(*message: Optional[str] = None*, *http_error: Optional[requests.exceptions.HTTPError] = None*)

Authentication error, raised on HTTP 401: Unauthorized.

class `mwdblib.exc.ValidationError`(*message: Optional[str] = None*, *http_error: Optional[requests.exceptions.HTTPError] = None*)

Validation error, raised on HTTP 400: Bad Request. Check the message to find more information about this error.

Most possible causes are:

- Search query syntax is incorrect
- Metakey has wrong format
- User/group name has wrong format
- Unexpected None's are provided as an argument

class `mwdblib.exc.ObjectError`(*message: Optional[str] = None*, *http_error: Optional[requests.exceptions.HTTPError] = None*)

Object error, raised when specified object cannot be accessed or uploaded.

class `mwdblib.exc.PermissionError`(*message: Optional[str] = None*, *http_error: Optional[requests.exceptions.HTTPError] = None*)

Permission error, raised when permissions are unsufficient (HTTP 403: Forbidden).

class `mwdblib.exc.LimitExceededError`(*message: Optional[str] = None*, *http_error: Optional[requests.exceptions.HTTPError] = None*)

Rate limit exceeded error. MWDB will try to throttle requests unless *obey_ratelimiter* flag is set.

class `mwdblib.exc.InternalError`(*message: Optional[str] = None*, *http_error: Optional[requests.exceptions.HTTPError] = None*)

Internal error. Something really bad occurred on the server side.

class `mwdblib.exc.NotAuthenticatedError`(*message: Optional[str] = None*, *http_error: Optional[requests.exceptions.HTTPError] = None*)

Authentication is required for specified request but credentials are not set. Use `MWDB.login()` or set API key.

class `mwdblib.exc.InvalidCredentialsError`(*message: Optional[str] = None*, *http_error: Optional[requests.exceptions.HTTPError] = None*)

Provided wrong password, API key has wrong format or was revoked.

class `mwdblib.exc.UserPendingError`(*message: Optional[str] = None*, *http_error: Optional[requests.exceptions.HTTPError] = None*)

User has just been registered and is waiting for acceptance.

class `mwdblib.exc.UserDisabledError`(*message: Optional[str] = None*, *http_error: Optional[requests.exceptions.HTTPError] = None*)

User is banned. Contact your administrator for more information.

class `mwdblib.exc.MaintenanceUnderwayError`(*message: Optional[str] = None*, *http_error: Optional[requests.exceptions.HTTPError] = None*)

MWDB has been turned into maintenance mode. Try again later.

class `mwdblib.exc.ObjectNotFoundError`(*message: Optional[str] = None*, *http_error: Optional[requests.exceptions.HTTPError] = None*)

Object is not found, because it doesn't exist or you are not permitted to access it.

class `mwdblib.exc.TypeConflictError`(*message: Optional[str] = None*, *http_error: Optional[requests.exceptions.HTTPError] = None*)

Object you want to upload exists yet and has different type. Use `MWDB.query()` to find it.

If you don't have access (`ObjectNotFoundError` is raised), try to upload it as config or blob.

Double check whether the data you want to upload are meaningful (not an empty file or single string).

`class mwdblib.exc.`**`BadResponseError`**(*message: Optional[str] = None*, *http_error: Optional[requests.exceptions.HTTPError] = None*)

Can't decode JSON response from server. Probably APIClient.api_url points to the MWDB web app instead of MWDB REST API.

`class mwdblib.exc.`**`GatewayError`**(*message: Optional[str] = None*, *http_error: Optional[requests.exceptions.HTTPError] = None*)

Bad Gateway or Gateway Timeout. It is serious but usually temporary, can be caused by new version deploy or lack of resources.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX