
Tamr Unify Python Client Documentation

Release 0.4

Tamr

Aug 09, 2019

Contents

1	Example	3
2	User Guide	5
2.1	FAQ	5
2.2	Installation	6
2.3	Quickstart	7
2.4	Secure Credentials	8
2.5	Workflows	10
2.6	Advanced Usage	11
3	Contributor Guide	15
3.1	Contributor Guide	15
4	Developer Interface	17
4.1	Developer Interface	17
	Index	29

Version: 0.4 | [View on Github](#)

CHAPTER 1

Example

```
from tamr_unify_client import Client
from tamr_unify_client.auth import UsernamePasswordAuth
import os

# grab credentials from environment variables
username = os.environ['UNIFY_USERNAME']
password = os.environ['UNIFY_PASSWORD']
auth = UsernamePasswordAuth(username, password)

host = 'localhost' # replace with your Tamr Unify host
unify = Client(auth, host=host)

# programmatically interact with Tamr Unify!
# e.g. refresh your project's Unified Dataset
project = unify.projects.by_resource_id('3')
ud = project.unified_dataset()
op = ud.refresh()
assert op.succeeded()
```


2.1 FAQ

2.1.1 What version of the Python Client should I use?

If you are starting a new project or your existing project does not yet use the Python Client, we encourage you to use the **latest stable version** of the Python Client.

If you are already using the Python Client, you have 3 options:

1. **“I like my project’s code the way it is.”**

Keep using the version you are on.

2. **“I want some new features released in versions with the same major version that I’m currently using.”**

Upgrade to the latest stable version *with the same major version* as what you currently use.

3. **“I want all new features and I’m willing to modify my code to get those features!”**

Upgrade to the latest stable version *even* if it has a different major version from what you currently use.

Note that you do not need to reason about the Unify API version nor the the Unify version.

How does this the Python Client accomplish this?

The short answer is that the Python Client just cares about features, and will try everything it knows to implement those features correctly, independent of the API version.

We’ll illustrate with an example.

Let’s say you want to get a dataset by name in your Python code.

1. If no such feature exists, you can file a Feature Request. Note that the Python Client is limited by what the Unify API enables. So you should check if the Unify API docs to see if the feature you want is even possible.

2. If this feature already exists, you can try it out!

E.g. `unify.datasets.by_name(some_dataset_name)`

2.a It works!

2.b If it fails with an HTTP error, it could be for 2 reasons:

2.a.i It might be impossible to support that feature in the Python Client because your Unify API version does not have the necessary endpoints to support it.

2.a.ii Your Unify API version *does* support this feature with some endpoints, but the Python Client know how to correctly implement this feature for this version of the API. In this case, you should submit a Feature Request.

2.c If it fails with any other error, you should submit a Bug Report.

Note: To see how to submit Bug Reports / Feature Requests, see [Bug Reports / Feature Requests](#).

To check what endpoints your version of the Unify API supports, see docs.tamr.com/reference (be sure to select the correct version in the top left!).

2.1.2 How do I call custom endpoints, e.g. endpoints outside the Unify API?

To call a custom endpoint *within* the Unify API, use the `client.request()` method, and provide an endpoint described by a path relative to `base_path`. For example, if `base_path` is `/api/versioned/v1/` (the default), and you want to get `/api/versioned/v1/projects/1`, you only need to provide `projects/1` (the relative ID provided by the project) as the endpoint, and the Client will resolve that into `/api/versioned/v1/projects/1`.

There are various APIs outside the `/api/versioned/v1/` prefix that are often useful or necessary to call - e.g. `/api/service/health`, or other un-versioned / unsupported APIs. To call a custom endpoint *outside* the Unify API, use the `client.request()` method, and provide an endpoint described by an *absolute* path (a path starting with `/`). For example, to get `/api/service/health` (no matter what `base_path` is), call `client.request()` with `/api/service/health` as the endpoint. The Client will ignore `base_path` and send the request directly against the absolute path provided.

For additional detail, see [Custom HTTP requests and Unversioned API Access](#).

2.2 Installation

`tamr-unify-client` is compatible with Python 3.6 or newer.

Installation is as simple as:

```
pip install tamr-unify-client
```

Note: We recommend you use a virtual environment for your project and install the Python Client into that virtual environment.

You can create a virtual environment with Python 3 via:

```
python3 -m venv my-venv
```

For more, see [The Hitchhiker's Guide to Python](#) .

2.2.1 Offline installs

First, download `tamr-unify-client` and its dependencies on a machine with online access to PyPI:

```
pip download tamr-unify-client -d tamr-unify-client-requirements
zip -r tamr-unify-client-requirements.zip tamr-unify-client-requirements
```

Then, ship the `.zip` file to the target machine where you want `tamr-unify-client` installed. You can do this via email, cloud drives, `scp` or any other mechanism.

Finally, install `tamr-unify-client` from the saved dependencies:

```
unzip tamr-unify-client-requirements.zip
pip install --no-index --find-links=tamr-unify-client-requirements tamr-unify-client
```

If you are not using a virtual environment, you may need to specify the `--user` flag if you get permissions errors:

```
pip install --user --no-index --find-links=tamr-unify-client-requirements tamr-unify-
↪client
```

2.3 Quickstart

2.3.1 Client configuration

Start by importing the Python Client and authentication provider:

```
from tamr_unify_client import Client
from tamr_unify_client.auth import UsernamePasswordAuth
```

Next, create an authentication provider and use that to create an authenticated client:

```
import os

username = os.environ['UNIFY_USERNAME']
password = os.environ['UNIFY_PASSWORD']

auth = UsernamePasswordAuth(username, password)
unify = Client(auth)
```

Warning: For security, it's best to read your credentials in from environment variables or secure files instead of hardcoding them directly into your code.

For more, see [User Guide > Secure Credentials](#) .

By default, the client tries to find the Unify instance on `localhost`. To point to a different host, set the `host` argument when instantiating the `Client`.

For example, to connect to 10.20.0.1:

```
unify = Client(auth, host='10.20.0.1')
```

2.3.2 Top-level collections

The Python Client exposes 2 top-level collections: Projects and Datasets.

You can access these collections through the client and loop over their members with simple `for`-loops.

E.g.:

```
for project in unify.projects:
    print(project.name)

for dataset in unify.datasets:
    print(dataset.name)
```

2.3.3 Fetch a specific resource

If you know the identifier for a specific resource, you can ask for it directly via the `by_resource_id` methods exposed by collections.

E.g. To fetch the project with ID '1':

```
project = unify.projects.by_resource_id('1')
```

2.3.4 Resource relationships

Related resources (like a project and its unified dataset) can be accessed through specific methods.

E.g. To access the Unified Dataset for a particular project:

```
ud = project.unified_dataset()
```

2.3.5 Kick-off Unify Operations

Some methods on Model objects can kick-off long-running Unify operations.

Here, kick-off a “Unified Dataset refresh” operation:

```
operation = project.unified_dataset().refresh()
assert op.succeeded()
```

By default, the API Clients expose a synchronous interface for Unify operations.

2.4 Secure Credentials

This section discusses ways to pass credentials securely to `UsernamePasswordAuth`. Specifically, you **should not** hardcode your password(s) in your source code. Instead, you should use environment variables or secure files to store your credentials and simple Python code to read your credentials.

2.4.1 Environment variables

You can use `os.environ` to read in your credentials from environment variables:

```
# my_script.py
import os

from tamr_unify_client.auth import UsernamePasswordAuth

username = os.environ['UNIFY_USERNAME'] # replace with your username environment_
↳variable name
password = os.environ['UNIFY_PASSWORD'] # replace with your password environment_
↳variable name

auth = UsernamePasswordAuth(username, password)
```

You can pass in the environment variables from the terminal by including them before your command:

```
UNIFY_USERNAME="my Unify username" UNIFY_PASSWORD="my Unify password" python my_
↳script.py
```

You can also create an `.sh` file to store your environment variables and simply source that file before running your script.

2.4.2 Config files

You can also store your credentials in a secure credentials file:

```
# credentials.yaml
---
username: "my unify username"
password: "my unify password"
```

Then pip install `pyyaml` read the credentials in your Python code:

```
# my_script.py
from tamr_unify_client.auth import UsernamePasswordAuth
import yaml

creds = yaml.load("path/to/credentials.yaml") # replace with your credentials.yaml_
↳path

auth = UsernamePasswordAuth(creds.username, creds.password)
```

As in this example, we recommend you use YAML as your format since YAML has support for comments and is more human-readable than JSON.

Important: You **should not** check these credentials files into your version control system (e.g. `git`). Do not share this file with anyone who should not have access to the password stored in it.

2.5 Workflows

2.5.1 Continuous Categorization

```
from tamr_unify_client import Client
from tamr_unify_client.auth import UsernamePasswordAuth
import os

username = os.environ['UNIFY_USERNAME']
password = os.environ['UNIFY_PASSWORD']
auth = UsernamePasswordAuth(username, password)

host = 'localhost' # replace with your host
unify = Client(auth)

project_id = "1" # replace with your project ID
project = unify.projects.by_resource_id(project_id)
project = project.as_categorization()

unified_dataset = project.unified_dataset()
op = unified_dataset.refresh()
assert op.succeeded()

model = project.model()
op = model.train()
assert op.succeeded()

op = model.predict()
assert op.succeeded()
```

2.5.2 Continuous Mastering

```
from tamr_unify_client import Client
from tamr_unify_client.auth import UsernamePasswordAuth
import os

username = os.environ['UNIFY_USERNAME']
password = os.environ['UNIFY_PASSWORD']
auth = UsernamePasswordAuth(username, password)

host = 'localhost' # replace with your host
unify = Client(auth)

project_id = "1" # replace with your project ID
project = unify.projects.by_resource_id(project_id)
project = project.as_mastering()

unified_dataset = project.unified_dataset()
op = unified_dataset.refresh()
assert op.succeeded()

op = project.pairs().refresh()
assert op.succeeded()
```

(continues on next page)

(continued from previous page)

```

model = project.pair_matching_model()
op = model.train()
assert op.succeeded()

op = model.predict()
assert op.succeeded()

op = project.record_clusters().refresh()
assert op.succeeded()

op = project.published_clusters().refresh()
assert op.succeeded()

```

2.6 Advanced Usage

2.6.1 Asynchronous Operations

You can opt-in to an asynchronous interface via the `asynchronous` keyword argument for methods that kick-off Unify operations.

E.g.:

```

operation = project.unified_dataset().refresh(asynchronous=True)
# do asynchronous stuff while operation is running
operation.wait() # hangs until operation finishes
assert op.succeeded()

```

2.6.2 Logging API calls

It can be useful (e.g. for debugging) to log the API calls made on your behalf by the Python Client.

You can set up HTTP-API-call logging on any client via standard Python logging mechanisms

```

from tamr_unify_client import Client
from unify_api_v1.auth import UsernamePasswordAuth
import logging

auth = UsernamePasswordAuth("username", "password")
unify = Client(auth)

# Reload the `logging` library since other libraries (like `requests`) already
# configure logging differently. See: https://stackoverflow.com/a/53553516/1490091
import imp
imp.reload(logging)

logging.basicConfig(
    level=logging.INFO, format="%(message)s", filename=log_path, filemode="w"
)
unify.logger = logging.getLogger(name)

```

By default, when logging is set up, the client will log `{method} {url} : {response_status}` for each API call.

You can customize this by passing in a value for `log_entry`:

```
def log_entry(method, url, response):
    # custom logging function
    # use the method, url, and response to construct the logged `str`
    # e.g. for logging out machine-readable JSON:
    import json
    return json.dumps({
        "request": f"{method} {url}",
        "status": response.status_code,
        "json": response.json(),
    })

# after configuring `unify.logger`
unify.log_entry = log_entry
```

2.6.3 Custom HTTP requests and Unversioned API Access

We encourage you to use the high-level, object-oriented interface offered by the Python Client. If you aren't sure whether you need to send low-level HTTP requests, you probably don't.

But sometimes it's useful to directly send HTTP requests to Unify; for example, Unify has many APIs that are not covered by the higher-level interface (most of which are neither versioned nor supported). You can still call these endpoints using the Python Client, but you'll need to work with raw `Response` objects.

Custom endpoint

The client exposes a `request` method with the same interface as `requests.request`:

```
# import Python Client library and configure your client

unify = Client(auth)
# do stuff with the `unify` client

# now I NEED to send a request to a specific endpoint
response = unify.request('GET', 'relative/path/to/resource')
```

This will send a request relative to the `base_path` registered with the client. If you provide an absolute path to the resource, the `base_path` will be ignored when composing the request:

```
# import Python Client library and configure your client

unify = Client(auth)

# request a resource outside the configured base_path
response = unify.request('GET', '/absolute/path/to/resource')
```

You can also use the `get`, `post`, `put`, `delete` convenience methods:

```
# e.g. `get` convenience method
response = unify.get('relative/path/to/resource')
```

Custom Host / Port / Base API path

If you need to repeatedly send requests to another port or base API path (i.e. not `api/versioned/v1/`), you can simply instantiate a different client.

Then just call `request` as described above:

```
# import Python Client library and configure your client
unify = api.Client(auth)
# do stuff with the `unify` client

# now I NEED to send requests to a different host/port/base API path etc..
# NOTE: in this example, we reuse `auth` from the first client, but we could
# have made a new Authentication provider if this client needs it.
custom_client = api.Client(
    auth,
    host="10.10.0.1",
    port=9090,
    base_path="api/some_service/",
)
response = custom_client.get('relative/path/to/resource')
```

Note that any component of the `base_path` after the final slash will be ignored; see the documentation on [urljoin](#) for details.

One-off authenticated request

All of the Python Client Authentication providers adhere to the `requests.auth.BaseAuth` interface.

This means that you can pass in an Authentication provider directly to the `requests` library:

```
from tamr_unify_client.auth import UsernamePasswordAuth
import os
import requests

username = os.environ['UNIFY_USERNAME']
password = os.environ['UNIFY_PASSWORD']
auth = UsernamePasswordAuth(username, password)

response = requests.request('GET', 'some/specific/endpoint', auth=auth)
```


3.1 Contributor Guide

3.1.1 Code of Conduct

See [CODE_OF_CONDUCT.md](#)

3.1.2 Bug Reports / Feature Requests

Please leave bug reports and feature requests as [Github issues](#) .

Be sure to check through existing issues (open and closed) to confirm that the bug hasn't been reported before.

Duplicate bug reports are a huge drain on the time of other contributors, and should be avoided as much as possible.

3.1.3 Pull Requests

For larger, new features:

[Open an RFC issue](#) . Discuss the feature with project maintainers to be sure that your change fits with the project vision and that you won't be wasting effort going in the wrong direction.

Once you get the green light from maintainers, you can proceed with the PR.

Contributions / PRs should follow the [Forking Workflow](#) :

1. Fork it: <https://github.com/{}your-github-username{/unify-client-python/fork>
2. Create your feature branch:

```
git checkout -b my-new-feature
```

3. Commit your changes:

```
git commit -am 'Add some feature'
```

4. Push to the branch:

```
git push origin my-new-feature
```

5. Create a new Pull Request
-

We optimize for PR readability, so please squash commits before and during the PR review process if you think it will help reviewers and onlookers navigate your changes.

Don't be afraid to push `-f` on your PRs when it helps our eyes read your code.

3.1.4 Installation

1. Clone your fork and cd into the project:

```
git clone https://github.com/<your-github-username>/unify-client-python
cd unify-client-python
```

2. Create a virtualenv:

```
python3 -m venv .venv
source .venv/bin/activate
```

Note: See [User Guide > Installation](#) for compatible Python versions.

Caution: If you place your virtualenv within the project codebase, you must name it `.venv` for `flake8` to know how to exclude it from linting.

3. Install dev dependencies:

```
pip install -e .[dev]
```

3.1.5 Tests

To run all tests:

```
pytest .
```

To run specific tests, see [these pytest docs](#).

3.1.6 Editor config

Atom :

- `python-black`
- `linter-flake8` (be sure to activate your virtualenv BEFORE opening Atom)

4.1 Developer Interface

4.1.1 Authentication

class `tamr_unify_client.auth.UsernamePasswordAuth` (*username, password*)

Provides username/password authentication for Unify. Specifically, sets the *Authorization* HTTP header with Unify's custom *BasicCreds* format.

Parameters

- **username** (*str*) –
- **password** (*str*) –

Usage:

```
>>> from tamr_unify_client.auth import UsernamePasswordAuth
>>> auth = UsernamePasswordAuth('my username', 'my password')
>>> import tamr_unify_client as api
>>> unify = api.Client(auth)
```

4.1.2 Client

class `tamr_unify_client.Client` (*auth, host='localhost', protocol='http', port=9100, base_path='api/versioned/v1/', session=None*)

Python Client for Unify API. Each client is specific to a specific origin (protocol, host, port).

Parameters

- **auth** (`requests.auth.AuthBase`) – Unify-compatible Authentication provider.
Recommended: use one of the classes described in *Authentication*
- **host** (*str*) – Host address of remote Unify instance (e.g. *10.0.10.0*). Default: *'localhost'*

- **protocol** (*str*) – Either 'http' or 'https'. Default: 'http'
- **port** (*int*) – Unify instance main port. Default: 9100
- **base_path** (*str*) – Base API path. Requests made by this client will be relative to this path. Default: 'api/versioned/v1'
- **session** (*requests.Session*) – Session to use for API calls. Default: A new default *requests.Session()*.

Usage:

```
>>> import tamr_unify_client as api
>>> from tamr_unify_client.auth import UsernamePasswordAuth
>>> auth = UsernamePasswordAuth('my username', 'my password')
>>> local = api.Client(auth) # on http://localhost:9100
>>> remote = api.Client(auth, protocol='https', host='10.0.10.0') # on https://
↪ /10.0.10.0:9100
```

origin

HTTP origin i.e. <protocol>://<host>[:<port>]. For additional information, see [MDN web docs](#).

Type *str*

request (*method, endpoint, **kwargs*)

Sends an authenticated request to the server. The URL for the request will be "<origin>/<base_path>/<endpoint>".

Parameters

- **method** (*str*) – The HTTP method for the request to be sent.
- **endpoint** (*str*) – API endpoint to call (relative to the Base API path for this client).

Returns HTTP response

Return type *requests.Response*

get (*endpoint, **kwargs*)

Calls *request()* with the "GET" method.

post (*endpoint, **kwargs*)

Calls *request()* with the "POST" method.

put (*endpoint, **kwargs*)

Calls *request()* with the "PUT" method.

delete (*endpoint, **kwargs*)

Calls *request()* with the "DELETE" method.

projects

Collection of all projects on this Unify instance.

Returns Collection of all projects.

Return type *ProjectCollection*

datasets

Collection of all datasets on this Unify instance.

Returns Collection of all datasets.

Return type *DatasetCollection*

4.1.3 Dataset

```

class tamr_unify_client.models.dataset.resource.Dataset (client, data, alias=None)
    A Unify dataset.

    name
        Type str

    external_id
        Type str

    description
        Type str

    version
        Type str

    tags
        Type list[str]

    update_records (records)
        Send a batch of record creations/updates/deletions to this dataset.

        Parameters records (list[dict]) – Each record should be formatted as specified in the
            Public Docs for Dataset updates.

    refresh (**options)
        Brings dataset up-to-date if needed, taking whatever actions are required.

        Parameters **options – Options passed to underlying Operation . See
            apply_options() .

    records ()
        Stream this dataset’s records as Python dictionaries.

        Returns Stream of records.

        Return type Python generator yielding dict

    status () → tamr_unify_client.models.dataset_status.DatasetStatus
        Retrieve this dataset’s streamability status.

        Returns Dataset streamability status.

        Return type DatasetStatus

    relative_id
        Type str

    resource_id
        Type str

```

4.1.4 Dataset Status

```

class tamr_unify_client.models.dataset_status.DatasetStatus (client, data, alias=None)
    Streamability status of a Unify dataset.

```

dataset_name

The name of the associated dataset.

Type `str`

relative_dataset_id

The relative dataset ID of the associated dataset.

Type `str`

is_streamable

Whether the associated dataset is available to be streamed.

Type `bool`

relative_id

Type `str`

resource_id

Type `str`

4.1.5 Datasets

class `tamr_unify_client.models.dataset.collection.DatasetCollection` (*client*,
api_path='datasets')

Collection of *Dataset* s.

Parameters

- **client** (*Client*) – Client for API call delegation.
- **api_path** (*str*) – API path used to access this collection. E.g. "projects/1/inputDatasets". Default: "datasets".

by_resource_id (*resource_id*)

Retrieve a dataset by resource ID.

Parameters **resource_id** (*str*) – The resource ID. E.g. "1"

Returns The specified dataset.

Return type *Dataset*

by_relative_id (*relative_id*)

Retrieve a dataset by relative ID.

Parameters **relative_id** (*str*) – The resource ID. E.g. "datasets/1"

Returns The specified dataset.

Return type *Dataset*

by_external_id (*external_id*)

Retrieve a dataset by external ID.

Parameters **external_id** (*str*) – The external ID.

Returns The specified dataset, if found.

Return type *Dataset*

Raises

- **KeyError** – If no dataset with the specified *external_id* is found

- **LookupError** – If multiple datasets with the specified `external_id` are found

stream()

Stream datasets in this collection. Implicitly called when iterating over this collection.

Returns Stream of datasets.

Return type Python generator yielding *Dataset*

Usage:

```
>>> for dataset in collection.stream(): # explicit
>>>     do_stuff(dataset)
>>> for dataset in collection: # implicit
>>>     do_stuff(dataset)
```

by_name (*dataset_name*)

Lookup a specific dataset in this collection by exact-match on name.

Parameters **dataset_name** (*str*) – Name of the desired dataset.

Returns Dataset with matching name in this collection.

Return type *Dataset*

Raises **KeyError** – If no dataset with specified name was found.

4.1.6 Machine Learning Models

class `tamr_unify_client.models.machine_learning_model.MachineLearningModel` (*client*, *data*, *alias=None*)

A Unify Machine Learning model.

train (***options*)

Learn from verified labels.

Parameters ****options** – Options passed to underlying *Operation* . See *apply_options()* .

predict (***options*)

Suggest labels for unverified records.

Parameters ****options** – Options passed to underlying *Operation* . See *apply_options()* .

relative_id

Type *str*

resource_id

Type *str*

4.1.7 Operations

class `tamr_unify_client.models.operation.Operation` (*client*, *data*, *alias=None*)

A long-running operation performed by Unify. Operations appear on the “Jobs” page of the Unify UI.

By design, client-side operations represent server-side operations *at a particular point in time* (namely, when the operation was fetched from the server). In other words: Operations *will not* pick up on server-side changes automatically. To get an up-to-date representation, refetch the operation e.g. `op = op.poll()`.

apply_options (*asynchronous=False, **options*)

Applies operation options to this operation.

NOTE: This function **should not** be called directly. Rather, options should be passed in through a higher-level function e.g. `refresh()`.

Synchronous mode: Automatically waits for operation to resolve before returning the operation.

asynchronous mode: Immediately return the 'PENDING' operation. It is up to the user to coordinate this operation with their code via `wait()` and/or `poll()`.

Parameters

- **asynchronous** (*bool*) – Whether or not to run in asynchronous mode. Default: `False`.
- ****options** – When running in synchronous mode, these options are passed to the underlying `wait()` call.

Returns Operation with options applied.

Return type *Operation*

type

Type `str`

description

Type `str`

state

Server-side state of this operation.

Operation state can be unresolved (i.e. `state` is one of: 'PENDING', 'RUNNING'), or resolved (i.e. `state` is one of: 'CANCELED', 'SUCCEEDED', 'FAILED'). Unless opting into asynchronous mode, all exposed operations should be resolved.

Note: you only need to manually pick up server-side changes when opting into asynchronous mode when kicking off this operation.

Usage:

```
>>> op.state # operation is currently 'PENDING'
'PENDING'
>>> op.wait() # continually polls until operation resolves
>>> op.state # incorrect usage; operation object state never changes.
'PENDING'
>>> op = op.poll() # correct usage; use value returned by Operation.poll_
↳ or Operation.wait
>>> op.state
'SUCCEEDED'
```

poll()

Poll this operation for server-side updates.

Does not update the calling `Operation` object. Instead, returns a new `Operation`.

Returns Updated representation of this operation.

Return type `Operation`

wait (*poll_interval_seconds=3, timeout_seconds=None*)
Continuously polls for this operation's server-side state.

Parameters

- **poll_interval_seconds** (*int*) – Time interval (in seconds) between subsequent polls.
- **timeout_seconds** (*int*) – Time (in seconds) to wait for operation to resolve.

Raises `TimeoutError` – If operation takes longer than *timeout_seconds* to resolve.

Returns Resolved operation.

Return type `Operation`

succeeded ()
Convenience method for checking if operation was successful.

Returns True if operation's state is 'SUCCEEDED', False otherwise.

Return type `bool`

relative_id

Type `str`

resource_id

Type `str`

4.1.8 Project

class `tamr_unify_client.models.project.resource.Project` (*client, data, alias=None*)
A Unify project.

name

Type `str`

external_id

Type `str`

description

Type `str`

type

One of: "SCHEMA_MAPPING" "SCHEMA_MAPPING_RECOMMENDATIONS"
"CATEGORIZATION" "DEDUP"

Type `str`

unified_dataset ()
Unified dataset for this project.

Returns Unified dataset for this project.

Return type `Dataset`

as_categorization ()
Convert this project to a `CategorizationProject`

Returns This project.

Return type *CategorizationProject*

Raises **TypeError** – If the *type* of this project is not "CATEGORIZATION"

as_mastering()

Convert this project to a *MasteringProject*

Returns This project.

Return type *MasteringProject*

Raises **TypeError** – If the *type* of this project is not "DEDUP"

relative_id

Type *str*

resource_id

Type *str*

class `tamr_unify_client.models.project.categorization.CategorizationProject` (*client*,
data,
alias=None)

A Categorization project in Unify.

model()

Machine learning model for this Categorization project. Learns from verified labels and predicts categorization labels for unlabeled records.

Returns The machine learning model for categorization.

Return type *MachineLearningModel*

as_categorization()

Convert this project to a *CategorizationProject*

Returns This project.

Return type *CategorizationProject*

Raises **TypeError** – If the *type* of this project is not "CATEGORIZATION"

as_mastering()

Convert this project to a *MasteringProject*

Returns This project.

Return type *MasteringProject*

Raises **TypeError** – If the *type* of this project is not "DEDUP"

description

Type *str*

external_id

Type *str*

name

Type *str*

relative_id

Type `str`

resource_id

Type `str`

type

One of: "SCHEMA_MAPPING" "SCHEMA_MAPPING_RECOMMENDATIONS"
"CATEGORIZATION" "DEDUP"

Type `str`

unified_dataset()
Unified dataset for this project.

Returns Unified dataset for this project.

Return type `Dataset`

class `tamr_unify_client.models.project.mastering.MasteringProject` (*client, data, alias=None*)

A Mastering project in Unify.

pairs()
Record pairs generated by Unify’s binning model. Pairs are displayed on the “Pairs” page in the Unify UI.
Call `refresh()` from this dataset to regenerate pairs according to the latest binning model.

Returns The record pairs represented as a dataset.

Return type `Dataset`

pair_matching_model()
Machine learning model for pair-matching for this Mastering project. Learns from verified labels and predicts categorization labels for unlabeled pairs.

Calling `predict()` from this dataset will produce new (unpublished) clusters. These clusters are displayed on the “Clusters” page in the Unify UI.

Returns The machine learning model for pair-matching.

Return type `MachineLearningModel`

high_impact_pairs()
High-impact pairs as a dataset. Unify labels pairs as “high-impact” if labeling these pairs would help it learn most quickly (i.e. “Active learning”).

High-impact pairs are displayed with a lightning bolt icon on the “Pairs” page in the Unify UI.

Call `refresh()` from this dataset to produce new high-impact pairs according to the latest pair-matching model.

Returns The high-impact pairs represented as a dataset.

Return type `Dataset`

record_clusters()
Record Clusters as a dataset. Unify clusters labeled pairs using pairs model. These clusters populate the cluster review page and get transient cluster ids, rather than published cluster ids (i.e., “Permanent Ids”).

Call `refresh()` from this dataset to generate clusters based on to the latest pair-matching model.

Returns The record clusters represented as a dataset.

Return type *Dataset*

published_clusters ()

Published record clusters generated by Unify's pair-matching model.

Call *refresh* () from this dataset to republish clusters according to the latest clustering.

Returns The published clusters represented as a dataset.

Return type *Dataset*

as_categorization ()

Convert this project to a *CategorizationProject*

Returns This project.

Return type *CategorizationProject*

Raises **TypeError** – If the *type* of this project is not "CATEGORIZATION"

as_mastering ()

Convert this project to a *MasteringProject*

Returns This project.

Return type *MasteringProject*

Raises **TypeError** – If the *type* of this project is not "DEDUP"

description

Type *str*

external_id

Type *str*

name

Type *str*

relative_id

Type *str*

resource_id

Type *str*

type

One of: "SCHEMA_MAPPING" "SCHEMA_MAPPING_RECOMMENDATIONS"
"CATEGORIZATION" "DEDUP"

Type *str*

unified_dataset ()

Unified dataset for this project.

Returns Unified dataset for this project.

Return type *Dataset*

4.1.9 Projects

class `tamr_unify_client.models.project.collection.ProjectCollection` (*client*,
api_path='projects')

Collection of *Project*s.

Parameters

- **client** (*Client*) – Client for API call delegation.
- **api_path** (*str*) – API path used to access this collection. Default: "projects".

by_resource_id (*resource_id*)

Retrieve a project by resource ID.

Parameters **resource_id** (*str*) – The resource ID. E.g. "1"

Returns The specified project.

Return type *Project*

by_relative_id (*relative_id*)

Retrieve a project by relative ID.

Parameters **relative_id** (*str*) – The resource ID. E.g. "projects/1"

Returns The specified project.

Return type *Project*

by_external_id (*external_id*)

Retrieve a project by external ID.

Parameters **external_id** (*str*) – The external ID.

Returns The specified project, if found.

Return type *Project*

Raises

- **KeyError** – If no project with the specified `external_id` is found
- **LookupError** – If multiple projects with the specified `external_id` are found

stream ()

Stream projects in this collection. Implicitly called when iterating over this collection.

Returns Stream of projects.

Return type Python generator yielding *Project*

Usage:

```
>>> for project in collection.stream(): # explicit
>>>     do_stuff(project)
>>> for project in collection: # implicit
>>>     do_stuff(project)
```


A

Client (class in *tamr_unify_client*), 17

apply_options() (*tamr_unify_client.models.operation.Operation* method), 22

as_categorization() (*tamr_unify_client.models.project.categorization.CategorizationProject* method), 24

as_categorization() (*tamr_unify_client.models.project.mastering.MasteringProject* method), 26

as_categorization() (*tamr_unify_client.models.project.resource.Project* method), 23

as_mastering() (*tamr_unify_client.models.project.categorization.CategorizationProject* method), 24

as_mastering() (*tamr_unify_client.models.project.mastering.MasteringProject* method), 26

as_mastering() (*tamr_unify_client.models.project.resource.Project* method), 24

B

by_external_id() (*tamr_unify_client.models.dataset.collection.DatasetCollection* method), 20

by_external_id() (*tamr_unify_client.models.project.collection.ProjectCollection* method), 27

by_name() (*tamr_unify_client.models.dataset.collection.DatasetCollection* method), 21

by_relative_id() (*tamr_unify_client.models.dataset.collection.DatasetCollection* method), 20

by_relative_id() (*tamr_unify_client.models.project.collection.ProjectCollection* method), 27

by_resource_id() (*tamr_unify_client.models.dataset.collection.DatasetCollection* method), 20

by_resource_id() (*tamr_unify_client.models.project.collection.ProjectCollection* method), 27

C

CategorizationProject (class in *tamr_unify_client.models.project.categorization*), get() (*tamr_unify_client.Client* method), 18
24

Client (class in *tamr_unify_client*), 17

D

Dataset (class in *tamr_unify_client.models.dataset.resource*), 19

dataset_name (*tamr_unify_client.models.dataset_status.DatasetStatus* attribute), 19

DatasetCollection (class in *tamr_unify_client.models.dataset.collection*), 20

Datasets (*tamr_unify_client.Client* attribute), 18

DatasetStatus (class in *tamr_unify_client.models.dataset_status*), 19

description (*tamr_unify_client.Client* method), 18

description (*tamr_unify_client.models.dataset.resource.Dataset* attribute), 19

description (*tamr_unify_client.models.operation.Operation* attribute), 22

description (*tamr_unify_client.models.project.categorization.CategorizationProject* attribute), 24

description (*tamr_unify_client.models.project.mastering.MasteringProject* attribute), 26

description (*tamr_unify_client.models.project.resource.Project* attribute), 23

E

external_id (*tamr_unify_client.models.dataset.resource.Dataset* attribute), 19

external_id (*tamr_unify_client.models.project.categorization.CategorizationProject* attribute), 24

external_id (*tamr_unify_client.models.project.mastering.MasteringProject* attribute), 26

external_id (*tamr_unify_client.models.project.resource.Project* attribute), 23

G

get() (*tamr_unify_client.Client* method), 18

H

high_impact_pairs() (tamr_unify_client.models.project.mastering.MasteringProject method), 25

I

is_streamable (tamr_unify_client.models.dataset_status.DatasetStatus attribute), 20

M

MachineLearningModel (class in relative_dataset_id tamr_unify_client.models.machine_learning_model), 21

MasteringProject (class in relative_id tamr_unify_client.models.project.mastering), 25

model() (tamr_unify_client.models.project.categorization.CategorizationProject method), 24

N

name (tamr_unify_client.models.dataset.resource.Dataset attribute), 19

name (tamr_unify_client.models.project.categorization.CategorizationProject attribute), 24

name (tamr_unify_client.models.project.mastering.MasteringProject attribute), 26

name (tamr_unify_client.models.project.resource.Project attribute), 23

O

Operation (class in tamr_unify_client.models.operation), 21

origin (tamr_unify_client.Client attribute), 18

P

pair_matching_model() (tamr_unify_client.models.project.mastering.MasteringProject method), 25

pairs() (tamr_unify_client.models.project.mastering.MasteringProject method), 25

poll() (tamr_unify_client.models.operation.Operation method), 22

post() (tamr_unify_client.Client method), 18

predict() (tamr_unify_client.models.machine_learning_model.MachineLearningModel method), 21

Project (class in tamr_unify_client.models.project.resource), 23

ProjectCollection (class in tamr_unify_client.models.project.collection), 27

projects (tamr_unify_client.Client attribute), 18

published_clusters() (tamr_unify_client.models.project.mastering.MasteringProject method), 26

put() (tamr_unify_client.Client method), 18

R

record_clusters() (tamr_unify_client.models.project.mastering.MasteringProject method), 25

resource_id (tamr_unify_client.models.dataset.resource.Dataset attribute), 19

refresh() (tamr_unify_client.models.dataset.resource.Dataset method), 19

relative_dataset_id (tamr_unify_client.models.dataset_status.DatasetStatus attribute), 20

relative_id (tamr_unify_client.models.dataset.resource.Dataset attribute), 19

relative_id (tamr_unify_client.models.dataset_status.DatasetStatus attribute), 20

relative_id (tamr_unify_client.models.machine_learning_model.MachineLearningModel attribute), 21

relative_id (tamr_unify_client.models.operation.Operation attribute), 23

relative_id (tamr_unify_client.models.project.categorization.CategorizationProject attribute), 24

relative_id (tamr_unify_client.models.project.mastering.MasteringProject attribute), 26

relative_id (tamr_unify_client.models.project.resource.Project attribute), 24

request() (tamr_unify_client.Client method), 18

resource_id (tamr_unify_client.models.dataset.resource.Dataset attribute), 19

resource_id (tamr_unify_client.models.dataset_status.DatasetStatus attribute), 20

resource_id (tamr_unify_client.models.machine_learning_model.MachineLearningModel attribute), 21

resource_id (tamr_unify_client.models.operation.Operation attribute), 23

resource_id (tamr_unify_client.models.project.categorization.CategorizationProject attribute), 25

resource_id (tamr_unify_client.models.project.mastering.MasteringProject attribute), 26

resource_id (tamr_unify_client.models.project.resource.Project attribute), 24

S

state (tamr_unify_client.models.operation.Operation attribute), 22

status() (tamr_unify_client.models.dataset.resource.Dataset method), 19

stream() (tamr_unify_client.models.dataset.collection.DatasetCollection method), 21

stream() (tamr_unify_client.models.project.collection.ProjectCollection method), 27

stream_id (tamr_unify_client.models.operation.Operation attribute), 23

T

tags (*tamr_unify_client.models.dataset.resource.Dataset* attribute), 19

train() (*tamr_unify_client.models.machine_learning_model.MachineLearningModel* method), 21

type (*tamr_unify_client.models.operation.Operation* attribute), 22

type (*tamr_unify_client.models.project.categorization.CategorizationProject* attribute), 25

type (*tamr_unify_client.models.project.mastering.MasteringProject* attribute), 26

type (*tamr_unify_client.models.project.resource.Project* attribute), 23

U

unified_dataset() (*tamr_unify_client.models.project.categorization.CategorizationProject* method), 25

unified_dataset() (*tamr_unify_client.models.project.mastering.MasteringProject* method), 26

unified_dataset() (*tamr_unify_client.models.project.resource.Project* method), 23

update_records() (*tamr_unify_client.models.dataset.resource.Dataset* method), 19

UsernamePasswordAuth (class in *tamr_unify_client.auth*), 17

V

version (*tamr_unify_client.models.dataset.resource.Dataset* attribute), 19

W

wait() (*tamr_unify_client.models.operation.Operation* method), 23