Tamr Unify Python Client Documentation

Release 0.2

Tamr

Aug 09, 2019

Contents

Version: 0.2 | 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 interace 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()
```

CHAPTER 2

User Guide

2.1 Installation

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 Quickstart

2.2.1 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
```

(continues on next page)

(continued from previous page)

```
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 instead of hardcoding them directly into your code.

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')
```

2.2.2 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)
```

2.2.3 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_resoure_id('1')
```

2.2.4 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()
```

2.2.5 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.

2.3 Workflows

2.3.1 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()
```

2.3.2 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)
```

(continues on next page)

(continued from previous page)

```
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()

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

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

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

2.4 Advanced Usage

2.4.1 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()
```

2.4.2 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
```

(continues on next page)

(continued from previous page)

```
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
```

2.4.3 Custom HTTP requests

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

But sometimes it's useful to directly send HTTP requests to Unify.

Specific 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', 'some/specific/endpoint')
```

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

```
# e.g. `get` convenience method
response = unify.get('some/specific/endpoint')
```

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('some/specific/endpoint')
```

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)
```

2.5 FAQ

2.5.1 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.

CHAPTER 3

Contributor Guide

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 Contributing

For bug fixes, documentation changes, and small features:

1. Fork it: https://github.com/{[}my-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

For larger new features: Do everything as above, but first also make contact with the project maintainers to be sure your change fits with the project direction and you won't be wasting effort going in the wrong direction.

CHAPTER 4

Developer Interface

4.1 Developer Interface

4.1.1 Client

class tamr_unify_client.**Client**(*auth*, *host='localhost'*, *protocol='http'*, *port=9100*, *base_path='api/versioned/v1'*)

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"`

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

4.1.2 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*

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.

---

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

name

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`

relative_id

Type str

resource_id

Type str

4.1.3 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*

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)
```

---

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

name

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

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*

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

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*

4.1.4 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

4.1.5 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.status # operation is currently 'PENDING'
'PENDING'
>>> op.wait() # continually polls until operation resolves
>>> op.status # incorrect usage; operation object status never changes.
'PENDING'
>>> op = op.poll() # correct usage; use value returned by Operation.poll␣
↪or Operation.wait
>>> op.status
'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

4.1.6 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)
```

Index

W