Tamr Unify Python Client Documentation

Release 0.9

Tamr

Aug 23, 2019

Contents

Version: 0.9 | View on Github

CHAPTER 1

Example

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

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

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

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

CHAPTER 2

User Guide

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 Tamr API version nor the the Tamr 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 Tamr API enables. So you should check if the Tamr 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. `tamr.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 Tamr API version does not have the necessary endpoints to support it.

2.a.ii Your Tamr 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 Tamr 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 Tamr API?

To call a custom endpoint *within* the Tamr 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 Tamr 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.

2.2.1 Stable releases

Installation is as simple as:

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

Or:

```
poetry add tamr-unify-client
```

Note: If you don't use poetry, 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 .

2.2.2 Latest (unstable)

Note: This project uses the new `pyproject.toml` file, not a `setup.py` file, so make sure you have the latest version of `pip` installed: `pip install -U pip`.

To install the bleeding edge:

```
git clone https://github.com/Datatamer/tamr-client
cd tamr-client
pip install .
```

2.2.3 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['TAMR_USERNAME']
password = os.environ['TAMR_PASSWORD']

auth = UsernamePasswordAuth(username, password)
tamr = 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 Tamr 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`:

```
tamr = 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 tamr.projects:
  print(project.name)

for dataset in tamr.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 = tamr.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 Tamr Operations

Some methods on Model objects can kick-off long-running Tamr 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 Tamr 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['TAMR_USERNAME'] # replace with your username environment␣
↪variable name
password = os.environ['TAMR_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:

```
TAMR_USERNAME="my Tamr username" TAMR_PASSWORD="my Tamr 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 tamr username"
password: "my tamr 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['TAMR_USERNAME']
password = os.environ['TAMR_PASSWORD']
auth = UsernamePasswordAuth(username, password)

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

project_id = "1" # replace with your project ID
project = tamr.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
```

(continues on next page)

(continued from previous page)

```
username = os.environ['TAMR_USERNAME']
password = os.environ['TAMR_PASSWORD']
auth = UsernamePasswordAuth(username, password)

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

project_id = "1" # replace with your project ID
project = tamr.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.record_clusters().refresh()
assert op.succeeded()

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

2.6 Creating and Modifying Resources

2.6.1 Creating resources

Resources, such as projects, dataset, and attribute configurations, can be created through their respective collections. Each `create` function takes in a dictionary that conforms to the Tamr Public Docs for creating that resource type:

```
spec = {
    "name": "project",
    "description": "Mastering Project",
    "type": "DEDUP"
    "unifiedDatasetName": "project_unified_dataset"
}
project = tamr.projects.create(spec)
```

2.6.2 Using specs

These dictionaries can also be created using spec classes.

Each `Resource` has a corresponding `ResourceSpec` which can be used to build an instance of that resource by specifying the value for each property.

The spec can then be converted to a dictionary that can be passed to `create`.

For instance, to create a project:

```
spec = (
    ProjectSpec.new()
    .with_name("Project")
    .with_type("DEDUP")
    .with_description("Mastering Project")
    .with_unified_dataset_name("Project_unified_dataset")
)
project = tamr.projects.create(spec.to_dict())
```

Calling `with_*` on a spec creates a new spec with the same properties besides the modified one. The original spec is unaltered, so it could be used multiple times:

```
base_spec = (
    ProjectSpec.new()
    .with_type("DEDUP")
    .with_description("Mastering Project")
)

specs = []
for name in project_names:
    spec = (
        base_spec.with_name(name)
        .with_unified_dataset_name(name + "_unified_dataset")
    )
    specs.append(spec)

projects = [tamr.projects.create(spec.to_dict()) for spec in specs]
```

2.6.3 Creating a dataset

Datasets can be created as described above, but the dataset's schema and records must then be handled separately.

To combine all of these steps into one, `DatasetCollection` has a convenience function `create_from_dataframe` that takes a Pandas DataFrame. This makes it easy to create a Tamr dataset from a CSV:

```
import pandas as pd

df = pd.read_csv("my_data.csv")
dataset = tamr.datasets.create_from_dataframe(df, "primary key name", "My Data")
```

This will create a dataset called "My Data" with the specified primary key, an attribute for each column of the `DataFrame`, and the `DataFrame`'s rows as records.

2.6.4 Modifying a resource

Certain resources can also be modified using specs.

After getting a spec corresponding to a resource and modifying some properties, the updated resource can be committed to Unify with the `put` function:

```
updated_dataset = (
    dataset.spec()
    .with_description("Modified description")
    .put()
)
```

Each spec class has many properties that can be changed, but refer to the Public Docs for which properties will actually be updated in Tamr. If an immutable property is changed in the update request, the new value will simply be ignored.

2.7 Geospatial Data

2.7.1 What geospatial data is supported?

In general, the Python Geo Interface is supported; see https://gist.github.com/sgillies/2217756

There are three layers of information, modeled after GeoJSON; see https://tools.ietf.org/html/rfc7946 :

- The outermost layer is a FeatureCollection
- Within a FeatureCollection are Features, each of which represents one "thing", like a building or a river. Each feature has:
  - type (string; required)
  - id (object; required)
  - geometry (Geometry, see below; optional)
  - bbox ("bounding box", 4 doubles; optional)
  - properties (map[string, object]; optional)
- Within a Feature is a Geometry, which represents a shape, like a point or a polygon. Each geometry has:
  - type (one of "Point", "MultiPoint", "LineString", "MultiLineString", "Polygon", "MultiPolygon"; required)
  - coordinates (doubles; exactly how these are structured depends on the type of the geometry)

Although the Python Geo Interface is non-prescriptive when it comes to the data types of the id and properties, Tamr has a more restricted set of supported types. See https://docs.tamr.com/reference#attribute-types

The `Dataset` class supports the `__geo_interface__` property. This will produce one `FeatureCollection` for the entire dataset.

There is a companion iterator `itergeofeatures()` that returns a generator that allows you to stream the records in the dataset as Geospatial features.

To produce a GeoJSON representation of a dataset:

```
dataset = client.datasets.by_name("my_dataset")
with open("my_dataset.json", "w") as f:
  json.dump(dataset.__geo_interface__, f)
```

By default, `itergeofeatures()` will use the first dataset attribute with geometry type to fill in the feature geometry. You can override this by specifying the geometry attribute to use in the `geo_attr` parameter to `itergeofeatures`.

`Dataset` can also be updated from a feature collection that supports the Python Geo Interface:

```
import geopandas
geodataframe = geopandas.GeoDataFrame(...)
dataset = client.dataset.by_name("my_dataset")
dataset.from_geo_features(geodataframe)
```

By default the features' geometries will be placed into the first dataset attribute with geometry type. You can override this by specifying the geometry attribute to use in the `geo_attr` parameter to `from_geo_features`.

2.7.2 Rules for converting from Tamr records to Geospatial Features

The record's primary key will be used as the feature's `id`. If the primary key is a single attribute, then the value of that attribute will be the value of `id`. If the primary key is composed of multiple attributes, then the value of the `id` will be an array with the values of the key attributes in order.

Tamr allows any number of geometry attributes per record; the Python Geo Interface is limited to one. When converting Tamr records to Python Geo Features, the first geometry attribute in the schema will be used as the geometry; all other geometry attributes will appear as properties with no type conversion. In the future, additional control over the handling of multiple geometries may be provided; the current set of capabilities is intended primarily to support the use case of working with FeatureCollections within Tamr, and FeatureCollection has only one geometry per feature.

An attribute is considered to have geometry type if it has type `RECORD` and contains an attribute named `point`, `multiPoint`, `lineString`, `multiLineString`, `polygon`, or `multiPolygon`.

If an attribute named `bbox` is available, it will be used as `bbox`. No conversion is done on the value of `bbox`. In the future, additional control over the handling of `bbox` attributes may be provided.

All other attributes will be placed in `properties`, with no type conversion. This includes all geometry attributes other than the first.

2.7.3 Rules for converting from Geospatial Features to Tamr records

The Feature's `id` will be converted into the primary key for the record. If the record uses a simple key, no value translation will be done. If the record uses a composite key, then the value of the Feature's `id` must be an array of values, one per attribute in the key.

If the Feature contains keys in `properties` that conflict with the record keys, `bbox`, or geometry, those keys are ignored (omitted).

If the Feature contains a `bbox`, it is copied to the record's `bbox`.

All other keys in the Feature's `properties` are propagated to the same-name attribute on the record, with no type conversion.

2.7.4 Streaming data access

The `Dataset` method `itergeofeatures()` returns a generator that allows you to stream the records in the dataset as Geospatial features:

```
my_dataset = client.datasets.by_name("my_dataset")
for feature in my_dataset.itergeofeatures():
  do_something(feature)
```

Note that many packages that consume the Python Geo Interface will be able to consume this iterator directly. For example:

```
from geopandas import GeoDataFrame
df = GeoDataFrame.from_features(my_dataset.itergeofeatures())
```

This allows construction of a GeoDataFrame directly from the stream of records, without materializing the intermediate dataset.

2.8 Advanced Usage

2.8.1 Asynchronous Operations

You can opt-in to an asynchronous interface via the asynchronous keyword argument for methods that kick-off Tamr 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()
```

2.8.2 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 tamr_unify_client import UsernamePasswordAuth
import logging

auth = UsernamePasswordAuth("username", "password")
tamr = 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"
)
tamr.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({
```

(continues on next page)

(continued from previous page)

```
  "request": f"{method} {url}",
  "status": response.status_code,
  "json": response.json(),
})

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

2.8.3 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 Tamr; for example, Tamr 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

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

# now I NEED to send a request to a specific endpoint
response = tamr.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

tamr = Client(auth)

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

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

```
# e.g. `get` convenience method
response = tamr.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

tamr = api.Client(auth)
# do stuff with the `tamr` 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')
```

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['TAMR_USERNAME']
password =  os.environ['TAMR_PASSWORD']
auth = UsernamePasswordAuth(username, password)

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

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 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{]}/tamr-client/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 Install

This project uses `poetry` as its package manager. For details on `poetry`, see the official documentation .

1. Install pyenv:

```
curl https://pyenv.run | bash
```

2. Clone your fork and `cd` into the project:

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

3. Use `pyenv` to install a compatible Python version (`3.6` or newer; e.g. `3.7.3`):

```
pyenv install 3.7.3
```

4. Set that Python version to be your version for this project(e.g. `3.7.3`):

```
pyenv local 3.7.3
```

5. Check that your Python version matches the version specified in `.python-version`:

```
cat .python-version
python --version
```

6. Install `poetry` as described here:

```
curl -sSL https://raw.githubusercontent.com/sdispater/poetry/master/get-poetry.py␣
↪| python
```

7. Install dependencies via `poetry`:

```
poetry install
```

3.1.5 Run tests

To run all tests:

```
poetry run pytest .
```

To run specific tests, see these pytest docs .

3.1.6 Run style checks

To run linter:

```
poetry run flake8 .
```

To run formatter:

```
poetry run black --check .
```

Run the formatter without the *–check* flag to fix formatting in-place.

3.1.7 Build docs

To build the docs:

```
cd docs/
poetry run make html
```

After docs are build, view them by:

```
cd docs/ # unless you are there already
open -a 'Google Chrome' _build/html/index.html # open in your favorite browser
```

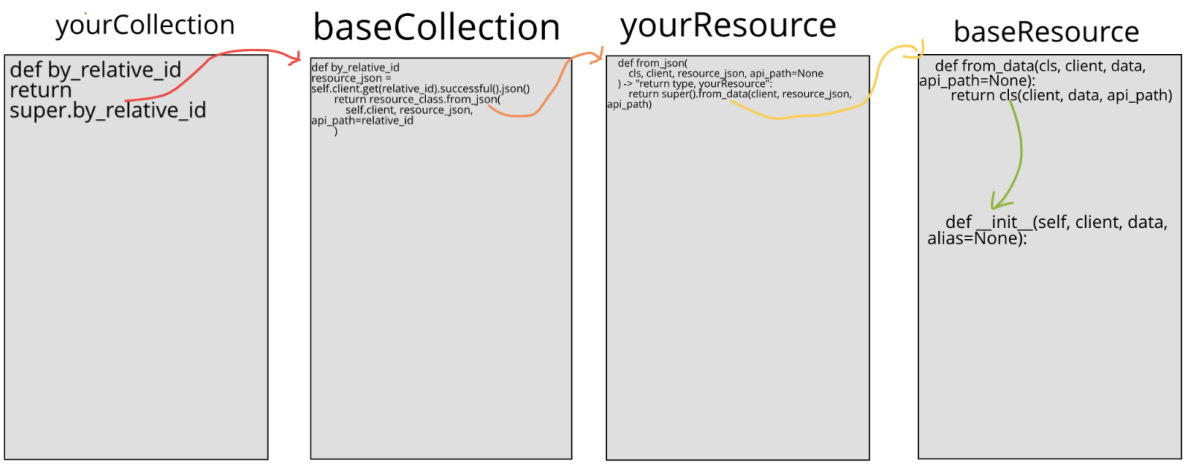
3.1.8 Editor config

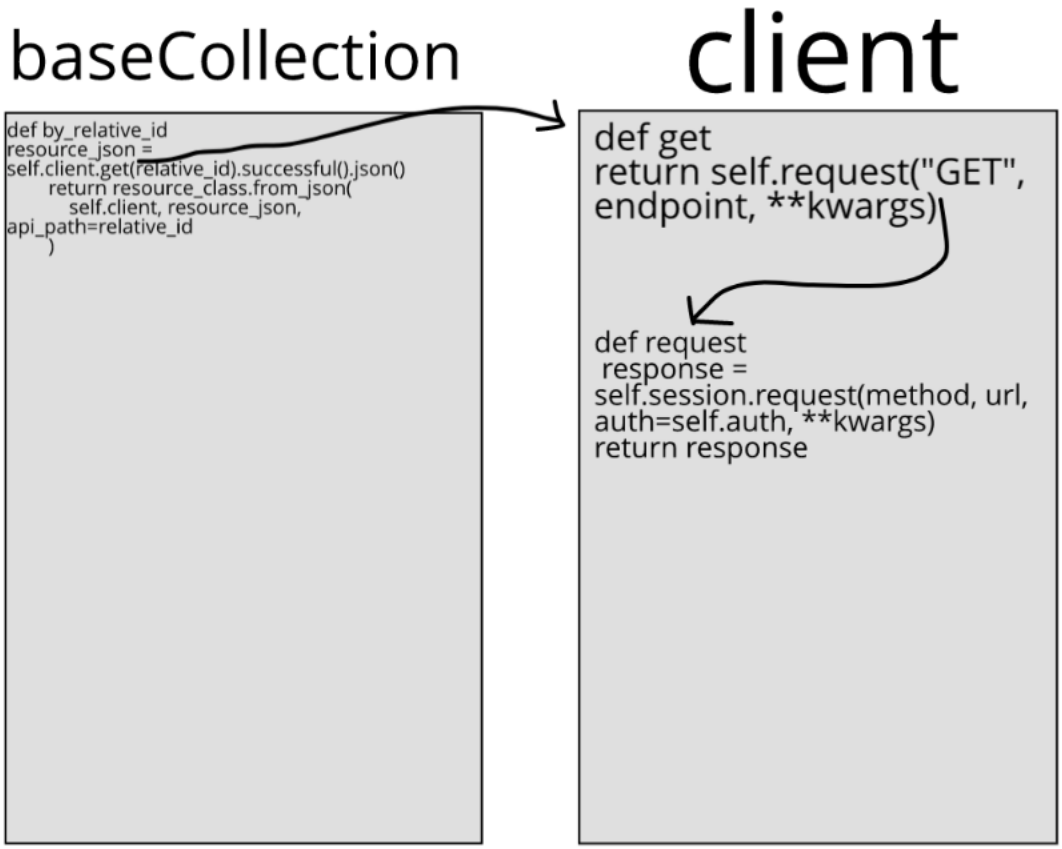
Atom :

- python-black
- linter-flake8

3.1.9 Overview of Resource and Collection interaction (from_json and from_data confusion)

yourResource and *yourCollection* are files that inherit from *baseResource* and *baseCollection*. Examples of such files would be *resource.py* and *collection.py* in the *attribute_configuration* folder under *project*.





Step 1 (red): *yourCollection*'s *by_relative_id* returns *super.by_relative_id*, which comes from *baseCollection*

Step 1a (black): within *by_relative_id*, variable *resource_json* is defined as *self.client.get.[etc]*. *Client*'s *.get* returns *self.request*

Step 1b (black): *client*'s *.request* makes a request to the provided URL (this is the method actually fetching the data)

Step 2 (orange): *baseCollection*'s *by_relative_id* returns *resource_class.from_json*, which is the *from_json* defined in *yourResource*

Step 3 (yellow): *yourResource*'s *from_json* returns *super.from_data*, which comes from *baseResource*

Step 4 (green): *baseResource*'s *from_data* returns *cls* , one of the parameters entered for *from_data*. *cls* is a *yourResource*, because in *from_json* the return type is specified to be a *yourResource*. When *cls* is returned, a *yourResource* that has been filled with the data retrieved in *client*'s *.request* is what comes back.

CHAPTER 4

Developer Interface

4.1 Developer Interface

4.1.1 Authentication

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

Provides username/password authentication for Tamr. Specifically, sets the *Authorization* HTTP header with Tamr'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 Tamr API. Each client is specific to a specific origin (protocol, host, port).

Parameters

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

- **protocol** (*str*) – Either *'http'* or *'https'*. Default: *'http'*
- **port** (*int*) – Tamr 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 Tamr instance.

Returns Collection of all projects.

Return type *ProjectCollection*

datasets
Collection of all datasets on this Tamr instance.

Returns Collection of all datasets.

Return type *DatasetCollection*

4.1.3 Attribute

Attribute

class tamr_unify_client.attribute.resource.**Attribute**(*client*, *data*, *alias=None*)
A Tamr Attribute.

See https://docs.tamr.com/reference#attribute-types

relative_id

Type str

name

Type str

description

Type str

type

Type *AttributeType*

is_nullable

Type bool

spec()
Returns a spec representation of this attribute.

Returns The attribute spec.

Return type *AttributeSpec*

delete()
Deletes this resource. Some resources do not support deletion, and will raise a 405 error if this is called.

Returns HTTP response from the server

Return type requests.Response

resource_id

Type str

Attribute Spec

class tamr_unify_client.attribute.resource.**AttributeSpec**(*client*, *data*, *api_path*)
A representation of the server view of an attribute

static of(*resource*)
Creates an attribute spec from an attribute.

Parameters **resource** (*Attribute*) – The existing attribute.

Returns The corresponding attribute spec.

Return type *AttributeSpec*

static new()
Creates a blank spec that could be used to construct a new attribute.

Returns The empty spec.

Return type *AttributeSpec*

from_data(*data*)
Creates a spec with the same client and API path as this one, but new data.

Parameters **data** (*dict*) – The data for the new spec.

Returns The new spec.

Return type *AttributeSpec*

to_dict()
Returns a version of this spec that conforms to the API representation.

Returns The spec's dict.

Return type dict

with_name(*new_name*)
Creates a new spec with the same properties, updating name.

Parameters **new_name** (*str*) – The new name.

Returns The new spec.

Return type *AttributeSpec*

with_description(*new_description*)
Creates a new spec with the same properties, updating description.

Parameters **new_description** (*str*) – The new description.

Returns The new spec.

Return type *AttributeSpec*

with_type(*new_type*)
Creates a new spec with the same properties, updating type.

Parameters **new_type** (*AttributeTypeSpec*) – The spec of the new type.

Returns The new spec.

Return type *AttributeSpec*

with_is_nullable(*new_is_nullable*)
Creates a new spec with the same properties, updating is nullable.

Parameters **new_is_nullable** (*bool*) – The new is nullable.

Returns The new spec.

Return type *AttributeSpec*

put()
Commits the changes and updates the attribute in Tamr.

Returns The updated attribute.

Return type *Attribute*

Attribute Collection

class tamr_unify_client.attribute.collection.**AttributeCollection**(*client*, *api_path*)
Collection of *Attribute* s.

Parameters

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

by_resource_id(*resource_id*)
Retrieve an attribute by resource ID.

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

Returns The specified attribute.

Return type *Attribute*

by_relative_id(*relative_id*)
Retrieve an attribute by relative ID.

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

Returns The specified attribute.

Return type *Attribute*

by_external_id(*external_id*)
Retrieve an attribute by external ID.

Since attributes do not have external IDs, this method is not supported and will raise a `NotImplementedError`.

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

Returns The specified attribute, if found.

Return type *Attribute*

Raises

- **KeyError** – If no attribute with the specified external_id is found
- **LookupError** – If multiple attributes with the specified external_id are found

stream()
Stream attributes in this collection. Implicitly called when iterating over this collection.

Returns Stream of attributes.

Return type Python generator yielding *Attribute*

Usage:

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

by_name(*attribute_name*)
Lookup a specific attribute in this collection by exact-match on name.

Parameters **attribute_name** (*str*) – Name of the desired attribute.

Returns Attribute with matching name in this collection.

Return type *Attribute*

create(*creation_spec*)
Create an Attribute in this collection

Parameters **creation_spec** (*dict[str, str]*) – Attribute creation specification should be formatted as specified in the Public Docs for adding an Attribute.

Returns The created Attribute

Return type *Attribute*

delete_by_resource_id(*resource_id*)
Deletes a resource from this collection by resource ID.

Parameters **resource_id** (*str*) – The resource ID of the resource that will be deleted.

Returns HTTP response from the server.

Return type requests.Response

Attribute Type

class tamr_unify_client.attribute.type.**AttributeType**(*data*)
The type of an *Attribute* or *SubAttribute*.

See https://docs.tamr.com/reference#attribute-types

Parameters **data** (dict) – JSON data representing this type

base_type

Type str

inner_type

Type *AttributeType*

attributes

Type list[*SubAttribute*]

spec()
Returns a spec representation of this attribute type.

Returns The attribute type spec.

Return type *AttributeTypeSpec*

Attribute Type Spec

class tamr_unify_client.attribute.type.**AttributeTypeSpec**(*data*)

static of(*resource*)
Creates an attribute type spec from an attribute type.

Parameters **resource** (*AttributeType*) – The existing attribute type.

Returns The corresponding attribute type spec.

Return type *AttributeTypeSpec*

static new()
Creates a blank spec that could be used to construct a new attribute type.

Returns The empty spec.

Return type *AttributeTypeSpec*

to_dict()
Returns a version of this spec that conforms to the API representation.

Returns The spec's dict.

Return type dict

with_base_type(*new_base_type*)
Creates a new spec with the same properties, updating the base type.

Parameters **new_base_type** (*str*) – The new base type.

Returns The new spec.

Return type *AttributeTypeSpec*

with_inner_type(*new_inner_type*)
Creates a new spec with the same properties, updating the inner type.

Parameters **new_inner_type** (*AttributeTypeSpec*) – The spec of the new inner type.

Returns The new spec.

Return type *AttributeTypeSpec*

with_attributes(*new_attributes*)
Creates a new spec with the same properties, updating attributes.

Parameters **new_attributes** (list[*AttributeSpec*]) – The specs of the new attributes.

Returns The new spec.

Return type *AttributeTypeSpec*

SubAttribute

class tamr_unify_client.attribute.subattribute.**SubAttribute**(*data*)
An attribute which is itself a property of another attribute.

See https://docs.tamr.com/reference#attribute-types

Parameters **data** (dict) – JSON data representing this attribute

name

Type str

description

Type str

type

Type *AttributeType*

is_nullable

Type bool

4.1.4 Categorization

Categorization Project

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

A Categorization project in Tamr.

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*

create_taxonomy(*creation_spec*)

Creates a *Taxonomy* for this project.

A taxonomy cannot already be associated with this project.

Parameters **creation_spec** (*dict*) – The creation specification for the taxonomy, which can include name.

Returns The new Taxonomy

Return type *Taxonomy*

taxonomy()

Retrieves the *Taxonomy* associated with this project. If a taxonomy is not already associated with this project, call *create_taxonomy()* first.

Returns The project's Taxonomy

Return type *Taxonomy*

add_input_dataset(*dataset*)

Associate a dataset with a project in Tamr.

By default, datasets are not associated with any projects. They need to be added as input to a project before they can be used as part of that project

Parameters **dataset** (*Dataset*) – The dataset to associate with the project.

Returns HTTP response from the server

Return type requests.Response

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"

attribute_configurations()
Project's attribute's configurations.

Returns The configurations of the attributes of a project.

Return type *AttributeConfigurationCollection*

attribute_mappings()
Project's attribute's mappings.

Returns The attribute mappings of a project.

Return type *AttributeMappingCollection*

attributes
Attributes of this project.

Returns Attributes of this project.

Return type *AttributeCollection*

delete()
Deletes this resource. Some resources do not support deletion, and will raise a 405 error if this is called.

Returns HTTP response from the server

Return type requests.Response

description

Type str

external_id

Type str

input_datasets()
Retrieve a collection of this project's input datasets.

Returns The project's input datasets.

Return type *DatasetCollection*

name

Type str

relative_id

Type str

remove_input_dataset(*dataset*)
Remove a dataset from a project.

Parameters **dataset** (*Dataset*) – The dataset to be removed from this project.

Returns HTTP response from the server

Return type requests.Response

resource_id

Type str

spec()
Returns this project's spec.

Returns The spec for the project.

Return type *ProjectSpec*

type
A Tamr project type, listed in https://docs.tamr.com/reference#create-a-project.

Type str

unified_dataset()
Unified dataset for this project.

Returns Unified dataset for this project.

Return type *Dataset*

Category

Category

class tamr_unify_client.categorization.category.resource.**Category**(*client*, *data*, *alias=None*)
A category of a taxonomy

name

Type str

description

Type str

path

Type list[str]

parent()
Gets the parent Category of this one, or None if it is a tier 1 category

Returns The parent Category or None

Return type *Category*

spec()
Returns this category's spec.

Returns The spec for the category.

Return type *CategorySpec*

delete()
Deletes this resource. Some resources do not support deletion, and will raise a 405 error if this is called.

Returns HTTP response from the server

Return type requests.Response

relative_id

Type str

resource_id

Type str

Category Spec

class tamr_unify_client.categorization.category.resource.**CategorySpec**(*client*, *data*, *api_path*)

A representation of the server view of a category.

static of(*resource*)

Creates a category spec from a category.

Parameters **resource** (*Category*) – The existing category.

Returns The corresponding category spec.

Return type *CategorySpec*

static new()

Creates a blank spec that could be used to construct a new category.

Returns The empty spec.

Return type *CategorySpec*

from_data(*data*)

Creates a spec with the same client and API path as this one, but new data.

Parameters **data** (*dict*) – The data for the new spec.

Returns The new spec.

Return type *CategorySpec*

to_dict()

Returns a version of this spec that conforms to the API representation.

Returns The spec's dict.

Return type dict

with_name(*new_name*)

Creates a new spec with the same properties, updating name.

Parameters **new_name** (*str*) – The new name.

Returns The new spec.

Return type *CategorySpec*

with_description(*new_description*)

Creates a new spec with the same properties, updating description.

Parameters **new_description** (*str*) – The new description.

Returns The new spec.

Return type *CategorySpec*

with_path(*new_path*)

Creates a new spec with the same properties, updating path.

Parameters **new_path** (*list[str]*) – The new path.

Returns The new spec.

Return type *CategorySpec*

Category Collection

class tamr_unify_client.categorization.category.collection.**CategoryCollection**(*client*, *api_path*)

Collection of *Category* s.

Parameters

- **client** (*Client*) – Client for API call delegation.
- **api_path** (*str*) – API path used to access this collection. E.g. `"projects/1/taxonomy/categories"`.

by_resource_id(*resource_id*)

Retrieve a category by resource ID.

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

Returns The specified category.

Return type *Category*

by_relative_id(*relative_id*)

Retrieve a category by relative ID.

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

Returns The specified category.

Return type *Category*

by_external_id(*external_id*)

Retrieve an attribute by external ID.

Since categories do not have external IDs, this method is not supported and will raise a `NotImplementedError`.

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

Returns The specified category, if found.

Return type *Category*

Raises

- **KeyError** – If no category with the specified external_id is found
- **LookupError** – If multiple categories with the specified external_id are found

stream()

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

Returns Stream of categories.

Return type Python generator yielding *Category*

Usage:

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

create(*creation_spec*)
Creates a new category.

Parameters **creation_spec** (*dict*) – Category creation specification, formatted as specified in the Public Docs for Creating a Category.

Returns The newly created category.

Return type *Category*

bulk_create(*creation_specs*)
Creates new categories in bulk.

Parameters **creation_specs** (*iterable[dict]*) – A collection of creation specifications, as detailed for create.

Returns JSON response from the server

Return type dict

delete_by_resource_id(*resource_id*)
Deletes a resource from this collection by resource ID.

Parameters **resource_id** (*str*) – The resource ID of the resource that will be deleted.

Returns HTTP response from the server.

Return type requests.Response

Taxonomy

class tamr_unify_client.categorization.taxonomy.**Taxonomy**(*client*, *data*, *alias=None*)
A project's taxonomy

name

Type str

categories()
Retrieves the categories of this taxonomy.

Returns A collection of the taxonomy categories.

Return type *CategoryCollection*

delete()
Deletes this resource. Some resources do not support deletion, and will raise a 405 error if this is called.

Returns HTTP response from the server

Return type requests.Response

relative_id

Type str

resource_id

Type str

4.1.5 Dataset

Dataset

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

name

Type str

external_id

Type str

description

Type str

version

Type str

tags

Type list[str]

key_attribute_names

Type list[str]

attributes
Attributes of this dataset.

Returns Attributes of this dataset.

Return type *AttributeCollection*

upsert_records(*records*, *primary_key_name*, ***json_args*)
Creates or updates the specified records.

Parameters

- **records** (*iterable[dict]*) – The records to update, as dictionaries.
- **primary_key_name** (*str*) – The name of the primary key for these records, which must be a key in each record dictionary.
- ****json_args** – Arguments to pass to the JSON *dumps* function, as documented here. Some of these, such as *indent*, may not work with Tamr.

Returns JSON response body from the server.

Return type dict

delete_records(*records*, *primary_key_name*)
Deletes the specified records.

Parameters

- **records** (*iterable[dict]*) – The records to delete, as dictionaries.
- **primary_key_name** (*str*) – The name of the primary key for these records, which must be a key in each record dictionary.

Returns JSON response body from the server.

Return type dict

delete_records_by_id(*record_ids*)
Deletes the specified records.

Parameters **record_ids** (*iterable*) – The IDs of the records to delete.

Returns JSON response body from the server.

Return type dict

delete_all_records()
Removes all records from the dataset.

Returns HTTP response from the server

Return type requests.Response

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

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

Returns The refresh operation.

Return type *Operation*

profile()
Returns profile information for a dataset.

If profile information has not been generated, call create_profile() first. If the returned profile information is out-of-date, you can call refresh() on the returned object to bring it up-to-date.

Returns Dataset Profile information.

Return type *DatasetProfile*

create_profile(***options*)
Create a profile for this dataset.

If a profile already exists, the existing profile will be brought up to date.

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

Returns The operation to create the profile.

Return type *Operation*

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

Returns Stream of records.

Return type Python generator yielding dict

status()
Retrieve this dataset's streamability status.

Returns Dataset streamability status.

Return type *DatasetStatus*

usage()
Retrieve this dataset's usage by recipes and downstream datasets.

Returns The dataset's usage.

Return type *DatasetUsage*

from_geo_features(*features*, *geo_attr=None*)
Upsert this dataset from a geospatial FeatureCollection or iterable of Features.

features can be:

- An object that implements `__geo_interface__` as a FeatureCollection (see https://gist.github.com/sgillies/2217756)
- An iterable of features, where each element is a feature dictionary or an object that implements the `__geo_interface__` as a Feature
- A map where the "features" key contains an iterable of features

See: geopandas.GeoDataFrame.from_features()

If geo_attr is provided, then the named Tamr attribute will be used for the geometry. If geo_attr is not provided, then the first attribute on the dataset with geometry type will be used for the geometry.

Parameters

- **features** – geospatial features
- **geo_attr** (*str*) – (optional) name of the Tamr attribute to use for the feature's geometry

upstream_datasets()
The Dataset's upstream datasets.

API returns the URIs of the upstream datasets, resulting in a list of DatasetURIs, not actual Datasets.

Returns A list of the Dataset's upstream datasets.

Return type list[*DatasetURI*]

spec()
Returns this dataset's spec.

Returns The spec of this dataset.

Return type *DatasetSpec*

delete(*cascade=False*)
Deletes this dataset, optionally deleting all derived datasets as well.

Parameters **cascade** (*bool*) – Whether to delete all datasets derived from this one. Optional, default is *False*. Do not use this option unless you are certain you need it as it can have unindended consequences.

Returns HTTP response from the server

Return type `requests.Response`

itergeofeatures(*geo_attr=None*)
Returns an iterator that yields feature dictionaries that comply with __geo_interface__

See https://gist.github.com/sgillies/2217756

Parameters **geo_attr** (*str*) – (optional) name of the Tamr attribute to use for the feature's geometry

Returns stream of features

Return type Python generator yielding `dict[str, object]`

relative_id

Type str

resource_id

Type str

Dataset Spec

class tamr_unify_client.dataset.resource.**DatasetSpec**(*client*, *data*, *api_path*)
A representation of the server view of a dataset.

static of(*resource*)
Creates a dataset spec from a dataset.

Parameters **resource** (*Dataset*) – The existing dataset.

Returns The corresponding dataset spec.

Return type *DatasetSpec*

static new()
Creates a blank spec that could be used to construct a new dataset.

Returns The empty spec.

Return type *DatasetSpec*

from_data(*data*)
Creates a spec with the same client and API path as this one, but new data.

Parameters **data** (*dict*) – The data for the new spec.

Returns The new spec.

Return type *DatasetSpec*

to_dict()
Returns a version of this spec that conforms to the API representation.

Returns The spec's dict.

Return type dict

with_name(*new_name*)
Creates a new spec with the same properties, updating name.

Parameters **new_name** (*str*) – The new name.

Returns A new spec.

Return type *DatasetSpec*

with_external_id(*new_external_id*)
Creates a new spec with the same properties, updating external ID.

Parameters **new_external_id** (*str*) – The new external ID.

Returns A new spec.

Return type *DatasetSpec*

with_description(*new_description*)
Creates a new spec with the same properties, updating description.

Parameters **new_description** (*str*) – The new description.

Returns A new spec.

Return type *DatasetSpec*

with_key_attribute_names(*new_key_attribute_names*)
Creates a new spec with the same properties, updating key attribute names.

Parameters **new_key_attribute_names** (*list[str]*) – The new key attribute names.

Returns A new spec.

Return type *DatasetSpec*

with_tags(*new_tags*)
Creates a new spec with the same properties, updating tags.

Parameters **new_tags** (*list[str]*) – The new tags.

Returns A new spec.

Return type *DatasetSpec*

put()
Updates the dataset on the server.

Returns The modified dataset.

Return type *Dataset*

Dataset Collection

class tamr_unify_client.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.

delete_by_resource_id(*resource_id*, *cascade=False*)

Deletes a dataset from this collection by resource_id. Optionally deletes all derived datasets as well.

Parameters

- **resource_id** (*str*) – The resource id of the dataset in this collection to delete.
- **cascade** (*bool*) – Whether to delete all datasets derived from the deleted one. Optional, default is *False*. Do not use this option unless you are certain you need it as it can have unindended consequences.

Returns HTTP response from the server.

Return type requests.Response

create(*creation_spec*)

Create a Dataset in Tamr

Parameters **creation_spec** (*dict[str, str]*) – Dataset creation specification should be formatted as specified in the Public Docs for Creating a Dataset.

Returns The created Dataset

Return type *Dataset*

create_from_dataframe(*df*, *primary_key_name*, *dataset_name*, *ignore_nan=True*)

Creates a dataset in this collection with the given name, creates an attribute for each column in the *df* (with *primary_key_name* as the key attribute), and upserts a record for each row of *df*.

Each attribute has the default type *ARRAY[STRING]*, besides the key attribute, which will have type *STRING*.

This function attempts to ensure atomicity, but it is not guaranteed. If an error occurs while creating attributes or records, an attempt will be made to delete the dataset that was created. However, if this request errors, it will not try again.

Parameters

- **df** (pandas.DataFrame) – The data to create the dataset with.
- **primary_key_name** (*str*) – The name of the primary key of the dataset. Must be a column of *df*.
- **dataset_name** (*str*) – What to name the dataset in Tamr. There cannot already be a dataset with this name.
- **ignore_nan** (*bool*) – Whether to convert *NaN* values to *null* before upserting records to Tamr. If *False* and *NaN* is in *df*, this function will fail. Optional, default is *True*.

Returns The newly created dataset.

Return type *Dataset*

Raises

- **KeyError** – If *primary_key_name* is not a column in *df*.
- ***CreationError*** – If a step in creating the dataset fails.

class tamr_unify_client.dataset.collection.**CreationError**(*error_message*)

An error from *create_from_dataframe()*

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

Dataset Profile

class tamr_unify_client.dataset.profile.**DatasetProfile**(*client*, *data*, *alias=None*)

Profile info of a Tamr 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_up_to_date

Whether the associated dataset is up to date.

Type bool

profiled_data_version

The profiled data version.

Type str

profiled_at

Info about when profile info was generated.

Type dict

simple_metrics

Simple metrics for profiled dataset.

Type list

attribute_profiles
Simple metrics for profiled dataset.

Type list

refresh(**options)
Updates the dataset profile if needed.

The dataset profile is updated on the server; you will need to call *profile()* to retrieve the updated profile.

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

Returns The refresh operation.

Return type *Operation*

delete()
Deletes this resource. Some resources do not support deletion, and will raise a 405 error if this is called.

Returns HTTP response from the server

Return type requests.Response

relative_id

Type str

resource_id

Type str

Dataset Status

class tamr_unify_client.dataset.status.**DatasetStatus**(*client*, *data*, *alias=None*)
Streamability status of a Tamr 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

delete()
Deletes this resource. Some resources do not support deletion, and will raise a 405 error if this is called.

Returns HTTP response from the server

Return type requests.Response

relative_id

Type str

resource_id

Type str

Dataset URI

class tamr_unify_client.dataset.uri.**DatasetURI**(*client*, *uri*)
Indentifier of a dataset.

Parameters

- **client** (Client) – Queried dataset's client.
- **uri** (str) – Queried dataset's dataset ID.

resource_id

Type str

relative_id

Type str

uri

Type str

dataset()
Fetch the dataset that this identifier points to.

Returns A Tamr dataset.

Return type

class *~tamr_unify_client.dataset.resource.Dataset*

Dataset Usage

class tamr_unify_client.dataset.usage.**DatasetUsage**(*client*, *data*, *alias=None*)
The usage of a dataset and its downstream dependencies.

See https://docs.tamr.com/reference#retrieve-downstream-dataset-usage

relative_id

Type str

usage

Type *DatasetUse*

dependencies

Type list[*DatasetUse*]

delete()
Deletes this resource. Some resources do not support deletion, and will raise a 405 error if this is called.

Returns HTTP response from the server

Return type requests.Response

resource_id

Type str

Dataset Use

class tamr_unify_client.dataset.use.**DatasetUse**(*client*, *data*)
The use of a dataset in project steps. This is not a *BaseResource* because it has no API path and cannot be directly retrieved or modified.

See https://docs.tamr.com/reference#retrieve-downstream-dataset-usage

Parameters

- **client** (*Client*) – Delegate underlying API calls to this client.
- **data** (dict) – The JSON body containing usage information.

dataset_id

Type str

dataset_name

Type str

input_to_project_steps

Type list[*ProjectStep*]

output_from_project_steps

Type list[*ProjectStep*]

dataset()
Retrieves the *Dataset* this use represents.

Returns The dataset being used.

Return type *Dataset*

4.1.6 Machine Learning Model

class tamr_unify_client.base_model.**MachineLearningModel**(*client*, *data*, *alias=None*)
A Tamr Machine Learning model.

train(***options**)
Learn from verified labels.

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

Returns The resultant operation.

Return type *Operation*

predict(***options**)
Suggest labels for unverified records.

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

Returns The resultant operation.

Return type *Operation*

delete()
Deletes this resource. Some resources do not support deletion, and will raise a 405 error if this is called.

Returns HTTP response from the server

Return type requests.Response

relative_id

Type str

resource_id

Type str

4.1.7 Mastering

Binning Model

class tamr_unify_client.mastering.binning_model.**BinningModel**(*client*, *data*, *alias=None*)

A binning model object.

records()

Stream this object's records as Python dictionaries.

Returns Stream of records.

Return type Python generator yielding dict

update_records(*records*)

Send a batch of record creations/updates/deletions to this dataset.

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

Returns JSON response body from server.

Return type dict

delete()

Deletes this resource. Some resources do not support deletion, and will raise a 405 error if this is called.

Returns HTTP response from the server

Return type requests.Response

relative_id

Type str

resource_id

Type str

Estimated Pair Counts

class tamr_unify_client.mastering.estimated_pair_counts.**EstimatedPairCounts**(*client*, *data*, *alias=None*)

Estimated Pair Counts info for Mastering Project

is_up_to_date

Whether an estimate pairs job has been run since the last edit to the binning model.

Return type bool

total_estimate
The total number of estimated candidate pairs and generated pairs for the model across all clauses.

Returns

A dictionary containing candidate pairs and estimated pairs mapped to their corresponding estimated counts. For example:

{

"candidatePairCount": "54321",

"generatedPairCount": "12345"

}

Return type dict[str, str]

clause_estimates
The estimated candidate pair count and generated pair count for each clause in the model.

Returns

A dictionary containing each clause name mapped to a dictionary containing the corresponding estimated candidate and generated pair counts. For example:

{

"Clause1": {

"candidatePairCount": "321",

"generatedPairCount": "123"

},

"Clause2": {

"candidatePairCount": "654",

"generatedPairCount": "456"

}

}

Return type dict[str, dict[str, str]]

refresh(***options*)
Updates the estimated pair counts if needed.

The pair count estimates are updated on the server; you will need to call *estimate_pairs()* to retrieve the updated estimate.

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

Returns The refresh operation.

Return type *Operation*

delete()
Deletes this resource. Some resources do not support deletion, and will raise a 405 error if this is called.

Returns HTTP response from the server

Return type requests.Response

relative_id

Type str

resource_id

Type str

Mastering Project

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

A Mastering project in Tamr.

pairs()

Record pairs generated by Tamr's binning model. Pairs are displayed on the "Pairs" page in the Tamr 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 Tamr UI.

Returns The machine learning model for pair-matching.

Return type *MachineLearningModel*

high_impact_pairs()

High-impact pairs as a dataset. Tamr 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 Tamr 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. Tamr 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 Tamr's pair-matching model.

Returns The published clusters represented as a dataset.

Return type *Dataset*

published_clusters_configuration()

Retrieves published clusters configuration for this project.

Returns The published clusters configuration

Return type *PublishedClustersConfiguration*

published_cluster_ids()
Retrieves published cluster IDs for this project.

Returns The published cluster ID dataset.

Return type *Dataset*

published_cluster_stats()
Retrieves published cluster stats for this project.

Returns The published cluster stats dataset.

Return type *Dataset*

published_cluster_versions(*cluster_ids*)
Retrieves version information for the specified published clusters. See https://docs.tamr.com/reference#retrieve-published-clusters-given-cluster-ids.

Parameters **cluster_ids** (*iterable[str]*) – The persistent IDs of the clusters to get version information for.

Returns A stream of the published clusters.

Return type Python generator yielding *PublishedCluster*

record_published_cluster_versions(*record_ids*)
Retrieves version information for the published clusters of the given records. See https://docs.tamr.com/reference#retrieve-published-clusters-given-record-ids.

Parameters **record_ids** (*iterable[str]*) – The Tamr IDs of the records to get cluster version information for.

Returns A stream of the relevant published clusters.

Return type Python generator yielding *RecordPublishedCluster*

estimate_pairs()
Returns pair estimate information for a mastering project

Returns Pairs Estimate information.

Return type *EstimatedPairCounts*

record_clusters_with_data()
Project's unified dataset with associated clusters.

Returns The record clusters with data represented as a dataset

Return type *Dataset*

published_clusters_with_data()
Project's unified dataset with associated clusters.

Returns The published clusters with data represented as a dataset

Return type *Dataset*

binning_model()
Binning model for this project.

Returns Binning model for this project.

Return type *BinningModel*

add_input_dataset(*dataset*)
Associate a dataset with a project in Tamr.

By default, datasets are not associated with any projects. They need to be added as input to a project before they can be used as part of that project

Parameters **dataset** (*Dataset*) – The dataset to associate with the project.

Returns HTTP response from the server

Return type requests.Response

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"

attribute_configurations()
Project's attribute's configurations.

Returns The configurations of the attributes of a project.

Return type *AttributeConfigurationCollection*

attribute_mappings()
Project's attribute's mappings.

Returns The attribute mappings of a project.

Return type *AttributeMappingCollection*

attributes
Attributes of this project.

Returns Attributes of this project.

Return type *AttributeCollection*

delete()
Deletes this resource. Some resources do not support deletion, and will raise a 405 error if this is called.

Returns HTTP response from the server

Return type requests.Response

description

Type str

external_id

Type str

input_datasets()
Retrieve a collection of this project's input datasets.

Returns The project's input datasets.

Return type *DatasetCollection*

name

Type str

relative_id

Type str

remove_input_dataset(*dataset*)
Remove a dataset from a project.

Parameters **dataset** (*Dataset*) – The dataset to be removed from this project.

Returns HTTP response from the server

Return type requests.Response

resource_id

Type str

spec()
Returns this project's spec.

Returns The spec for the project.

Return type *ProjectSpec*

type
A Tamr project type, listed in https://docs.tamr.com/reference#create-a-project.

Type str

unified_dataset()
Unified dataset for this project.

Returns Unified dataset for this project.

Return type *Dataset*

Published Cluster

Metric

class tamr_unify_client.mastering.published_cluster.metric.**Metric**(*data*)
A metric for a published cluster.

This is not a *BaseResource* because it does not have its own API endpoint.

Parameters **data** – The JSON entity representing this cluster.

name

Type str

value

Type str

Published Cluster

class tamr_unify_client.mastering.published_cluster.resource.**PublishedCluster**(*data*)
A representation of a published cluster in a mastering project with version information. See https://docs.tamr.com/reference#retrieve-published-clusters-given-cluster-ids.

This is not a *BaseResource* because it does not have its own API endpoint.

Parameters **data** – The JSON entity representing this *PublishedCluster*.

id

Type str

versions

Type list[*PublishedClusterVersion*]

Published Cluster Configuration

class tamr_unify_client.mastering.published_cluster.configuration.**PublishedClustersConfigu**

The configuration of published clusters in a project.

See https://docs.tamr.com/reference#the-published-clusters-configuration-object

relative_id

Type str

versions_time_to_live

Type str

spec()
Returns a spec representation of this published cluster configuration.

Returns The published cluster configuration spec.

Return type :class‘~tamr_unify_client.mastering.published_cluster.configuration.PublishedClustersConfigurationSpe

delete()
Deletes this resource. Some resources do not support deletion, and will raise a 405 error if this is called.

Returns HTTP response from the server

Return type requests.Response

resource_id

Type str

Published Cluster Version

class tamr_unify_client.mastering.published_cluster.version.**PublishedClusterVersion**(*data*)
A version of a published cluster in a mastering project.

This is not a *BaseResource* because it does not have its own API endpoint.

Parameters **data** – The JSON entity representing this version.

version

Type str

timestamp

Type str

name

Type str

metrics

Type list[*Metric*]

record_ids

Type list[dict[str, str]]

Record Published Cluster

class tamr_unify_client.mastering.published_cluster.record.**RecordPublishedCluster**(*data*)

A representation of a published cluster of a record in a mastering project with version information. See https://docs.tamr.com/reference#retrieve-published-clusters-given-record-ids.

This is not a *BaseResource* because it does not have its own API endpoint.

Parameters **data** – The JSON entity representing this *RecordPublishedCluster*.

entity_id

Type str

source_id

Type str

origin_entity_id

Type str

origin_source_id

Type str

versions

Type list[*RecordPublishedClusterVersion*]

Record Published Cluster Version

class tamr_unify_client.mastering.published_cluster.record_version.**RecordPublishedClusterVe**

A version of a published cluster in a mastering project.

This is not a *BaseResource* because it does not have its own API endpoint.

Parameters **data** – The JSON entity representing this version.

version

Type str

timestamp

Type str

cluster_id

Type str

4.1.8 Operation

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

A long-running operation performed by Tamr. Operations appear on the "Jobs" page of the Tamr 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'
```

(continues on next page)

(continued from previous page)

```
>>> 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`

delete()
Deletes this resource. Some resources do not support deletion, and will raise a 405 error if this is called.

Returns HTTP response from the server

Return type `requests.Response`

relative_id

Type str

resource_id

Type str

4.1.9 Project

Attribute Configuration

Attribute Configuration

class tamr_unify_client.project.attribute_configuration.resource.**AttributeConfiguration**(*clie*
dat
alia

The configurations of Tamr Attributes.

See https://docs.tamr.com/reference#the-attribute-configuration-object

relative_id

Type str

id

Type str

relative_attribute_id

Type str

attribute_role

Type str

similarity_function

Type str

enabled_for_ml

Type bool

tokenizer

Type str

numeric_field_resolution

Type list

attribute_name

Type str

spec()
Returns this attribute configuration's spec.

Returns The spec of this attribute configuration.

Return type *AttributeConfigurationSpec*

delete()
Deletes this resource. Some resources do not support deletion, and will raise a 405 error if this is called.

Returns HTTP response from the server

Return type requests.Response

resource_id

Type str

Attribute Configuration Spec

class tamr_unify_client.project.attribute_configuration.resource.**AttributeConfigurationSpec**

A representation of the server view of an attribute configuration.

static of(*resource*)
Creates an attribute configuration spec from an attribute configuration.

Parameters **resource** (*AttributeConfiguration*) – The existing attribute configuration.

Returns The corresponding attribute creation spec.

Return type *AttributeConfigurationSpec*

static new()
Creates a blank spec that could be used to construct a new attribute configuration.

Returns The empty spec.

Return type *AttributeConfigurationSpec*

from_data(*data*)
Creates a spec with the same client and API path as this one, but new data.

Parameters **data** (*dict*) – The data for the new spec.

Returns The new spec.

Return type *AttributeConfigurationSpec*

to_dict()
Returns a version of this spec that conforms to the API representation.

Returns The spec's dict.

Return type dict

with_attribute_role(*new_attribute_role*)
Creates a new spec with the same properties, updating attribute role.

Parameters **new_attribute_role** (*str*) – The new attribute role.

Returns A new spec.

Return type *AttributeConfigurationSpec*

with_similarity_function(*new_similarity_function*)
Creates a new spec with the same properties, updating similarity function.

Parameters **new_similarity_function** (*str*) – The new similarity function.

Returns A new spec.

Return type *AttributeConfigurationSpec*

with_enabled_for_ml(*new_enabled_for_ml*)
Creates a new spec with the same properties, updating enabled for ML.

Parameters **new_enabled_for_ml** (*bool*) – Whether the builder is enabled for ML.

Returns A new spec.

Return type *AttributeConfigurationSpec*

with_tokenizer(*new_tokenizer*)
Creates a new spec with the same properties, updating tokenizer.

Parameters **new_tokenizer** (*str*) – The new tokenizer.

Returns A new spec.

Return type *AttributeConfigurationSpec*

with_numeric_field_resolution(*new_numeric_field_resolution*)
Creates a new spec with the same properties, updating numeric field resolution.

Parameters **new_numeric_field_resolution** (*str*) – The new numeric field resolution.

Returns A new spec.

Return type *AttributeConfigurationSpec*

with_attribute_name(*new_attribute_name*)
Creates a new spec with the same properties, updating new attribute name.

Parameters **new_attribute_name** (*str*) – The new attribute name.

Returns A new spec.

Return type *AttributeConfigurationSpec*

put()
Updates the attribute configuration on the server.

Returns The modified attribute configuration.

Return type *AttributeConfiguration*

Attribute Configuration Collection

class tamr_unify_client.project.attribute_configuration.collection.**AttributeConfigurationC**

Collection of *AttributeConfiguration*

Parameters

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

by_resource_id(*resource_id*)
Retrieve an attribute configuration by resource ID.

Parameters **resource_id** (*str*) – The resource ID.

Returns The specified attribute configuration.

Return type *AttributeConfiguration*

by_relative_id(*relative_id*)
Retrieve an attribute configuration by relative ID.

Parameters **relative_id** (*str*) – The relative ID.

Returns The specified attribute configuration.

Return type *AttributeConfiguration*

by_external_id(*external_id*)
Retrieve an attribute configuration by external ID.

Since attributes do not have external IDs, this method is not supported and will raise a `NotImplementedError`.

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

Returns The specified attribute, if found.

Return type *AttributeConfiguration*

Raises

- **KeyError** – If no attribute with the specified external_id is found
- **LookupError** – If multiple attributes with the specified external_id are found
- **NotImplementedError** – AttributeConfiguration does not support external_id

stream()

Stream attribute configurations in this collection. Implicitly called when iterating over this collection.

Returns Stream of attribute configurations.

Return type Python generator yielding *AttributeConfiguration*

Usage:

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

create(*creation_spec*)

Create an Attribute configuration in this collection

Parameters **creation_spec** (*dict[str, str]*) – Attribute configuration creation specification should be formatted as specified in the Public Docs for adding an AttributeConfiguration.

Returns The created Attribute configuration

Return type *AttributeConfiguration*

delete_by_resource_id(*resource_id*)

Deletes a resource from this collection by resource ID.

Parameters **resource_id** (*str*) – The resource ID of the resource that will be deleted.

Returns HTTP response from the server.

Return type requests.Response

Attribute Mapping

Attribute Mapping

class tamr_unify_client.project.attribute_mapping.resource.**AttributeMapping**(*client*, *data*)

see https://docs.tamr.com/reference#retrieve-projects-mappings AttributeMapping and AttributeMappingCollection do not inherit from BaseResource and BaseCollection. BC and BR require a specific URL for each individual attribute mapping (ex: /projects/1/attributeMappings/1), but these types of URLs do not exist for attribute mappings

id

Type str

relative_id

Type str

input_attribute_id

Type str

relative_input_attribute_id

Type str

input_dataset_name

Type str

input_attribute_name

Type str

unified_attribute_id

Type str

relative_unified_attribute_id

Type str

unified_dataset_name

Type str

unified_attribute_name

Type str

resource_id

Type str

spec()
Returns a spec representation of this attribute mapping.

Returns The attribute mapping spec.

Return type *AttributeMappingSpec*

delete()
Delete this attribute mapping.

Returns HTTP response from the server

Return type requests.Response

Attribute Mapping Spec

class tamr_unify_client.project.attribute_mapping.resource.**AttributeMappingSpec**(*data*)
A representation of the server view of an attribute mapping

static of(*resource*)
Creates an attribute mapping spec from a attribute mapping.

Parameters **resource** (*AttributeMapping*) – The existing attribute mapping.

Returns The corresponding attribute mapping spec.

Return type *AttributeMappingSpec*

static new()
Creates a blank spec that could be used to construct a new attribute mapping.

Returns The empty spec.

Return type *AttributeMappingSpec*

to_dict()
Returns a version of this spec that conforms to the API representation.

Returns The spec's dict.

Return type dict

with_input_attribute_id(*new_input_attribute_id*)
Creates a new spec with the same properties, updating the input attribute id.

Parameters **new_input_attribute_id** (*str*) – The new input attribute id.

Returns The new spec.

Return type *AttributeMappingSpec*

with_relative_input_attribute_id(*new_relative_input_attribute_id*)
Creates a new spec with the same properties, updating the relative input attribute id.

Parameters **new_relative_input_attribute_id** (*str*) – The new relative input attribute Id.

Returns The new spec.

Return type *AttributeMappingSpec*

with_input_dataset_name(*new_input_dataset_name*)
Creates a new spec with the same properties, updating the input dataset name.

Parameters **new_input_dataset_name** (*str*) – The new input dataset name.

Returns The new spec.

Return type *AttributeMappingSpec*

with_input_attribute_name(*new_input_attribute_name*)
Creates a new spec with the same properties, updating the input attribute name.

Parameters **new_input_attribute_name** (*str*) – The new input attribute name.

Returns The new spec.

Return type *AttributeMappingSpec*

with_unified_attribute_id(*new_unified_attribute_id*)
Creates a new spec with the same properties, updating the unified attribute id.

Parameters **new_unified_attribute_id** (*str*) – The new unified attribute id.

Returns The new spec.

Return type *AttributeMappingSpec*

with_relative_unified_attribute_id(*new_relative_unified_attribute_id*)
Creates a new spec with the same properties, updating the relative unified attribute id.

Parameters **new_relative_unified_attribute_id** (*str*) – The new relative unified attribute id.

Returns The new spec.

Return type *AttributeMappingSpec*

with_unified_dataset_name(*new_unified_dataset_name*)
Creates a new spec with the same properties, updating the unified dataset name.

Parameters **new_unified_dataset_name** (*str*) – The new unified dataset name.

Returns The new spec.

Return type *AttributeMappingSpec*

with_unified_attribute_name(*new_unified_attribute_name*)
Creates a new spec with the same properties, updating the unified attribute name.

Parameters **new_unified_attribute_name** (*str*) – The new unified attribute name.

Returns The new spec.

Return type *AttributeMappingSpec*

Attribute Mapping Collection

class tamr_unify_client.project.attribute_mapping.collection.**AttributeMappingCollection**(*clie*
*api_*

Collection of *AttributeMapping*

Parameters

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

stream()
Stream attribute mappings in this collection. Implicitly called when iterating over this collection.

Returns Stream of attribute mappings.

Return type Python generator yielding *AttributeMapping*

by_resource_id(*resource_id*)
Retrieve an item in this collection by resource ID.

Parameters **resource_id** (*str*) – The resource ID.

Returns The specified attribute mapping.

Return type *AttributeMapping*

by_relative_id(*relative_id*)
Retrieve an item in this collection by relative ID.

Parameters **relative_id** (*str*) – The relative ID.

Returns The specified attribute mapping.

Return type *AttributeMapping*

create(*creation_spec*)
Create an Attribute mapping in this collection

Parameters **creation_spec** (*dict[str, str]*) – Attribute mapping creation specification should be formatted as specified in the Public Docs for adding an AttributeMapping.

Returns The created Attribute mapping

Return type *AttributeMapping*

delete_by_resource_id(*resource_id*)
Delete an attribute mapping using its Resource ID.

Parameters **resource_id** (*str*) – the resource ID of the mapping to be deleted.

Returns HTTP response from the server

Return type requests.Response

Project

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

A Tamr project.

name

Type str

external_id

Type str

description

Type str

type

A Tamr project type, listed in https://docs.tamr.com/reference#create-a-project.

Type str

attributes

Attributes of this project.

Returns Attributes of this project.

Return type *AttributeCollection*

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"

add_input_dataset(*dataset*)

Associate a dataset with a project in Tamr.

By default, datasets are not associated with any projects. They need to be added as input to a project before they can be used as part of that project

Parameters **dataset** (*Dataset*) – The dataset to associate with the project.

Returns HTTP response from the server

Return type requests.Response

remove_input_dataset(*dataset*)
Remove a dataset from a project.

Parameters **dataset** (*Dataset*) – The dataset to be removed from this project.

Returns HTTP response from the server

Return type requests.Response

input_datasets()
Retrieve a collection of this project's input datasets.

Returns The project's input datasets.

Return type *DatasetCollection*

attribute_configurations()
Project's attribute's configurations.

Returns The configurations of the attributes of a project.

Return type *AttributeConfigurationCollection*

attribute_mappings()
Project's attribute's mappings.

Returns The attribute mappings of a project.

Return type *AttributeMappingCollection*

spec()
Returns this project's spec.

Returns The spec for the project.

Return type *ProjectSpec*

delete()
Deletes this resource. Some resources do not support deletion, and will raise a 405 error if this is called.

Returns HTTP response from the server

Return type requests.Response

relative_id

Type str

resource_id

Type str

Project Spec

class tamr_unify_client.project.resource.**ProjectSpec**(*client*, *data*, *api_path*)
A representation of the server view of a project.

static of(*resource*)
Creates a project spec from a project.

Parameters **resource** (*Project*) – The existing project.

Returns The corresponding project spec.

Return type *ProjectSpec*

static new()
Creates a blank spec that could be used to construct a new project.

Returns The empty spec.

Return type *ProjectSpec*

from_data(*data*)
Creates a spec with the same client and API path as this one, but new data.

Parameters **data** (*dict*) – The data for the new spec.

Returns The new spec.

Return type *ProjectSpec*

to_dict()
Returns a version of this spec that conforms to the API representation.

Returns The spec's dict.

Return type dict

with_name(*new_name*)
Creates a new spec with the same properties, updating name.

Parameters **new_name** (*str*) – The new name.

Returns The new spec.

Return type *ProjectSpec*

with_description(*new_description*)
Creates a new spec with the same properties, updating description.

Parameters **new_description** (*str*) – The new description.

Returns The new spec.

Return type *ProjectSpec*

with_type(*new_type*)
Creates a new spec with the same properties, updating type.

Parameters **new_type** (*str*) – The new type.

Returns The new spec.

Return type *ProjectSpec*

with_external_id(*new_external_id*)
Creates a new spec with the same properties, updating external ID.

Parameters **new_external_id** (*str*) – The new external ID.

Returns The new spec.

Return type *ProjectSpec*

with_unified_dataset_name(*new_unified_dataset_name*)
Creates a new spec with the same properties, updating unified dataset name.

Parameters **new_unified_dataset_name** (*str*) – The new unified dataset name.

Returns The new spec.

Return type *ProjectSpec*

put()
Commits these changes by updating the project in Tamr.

Returns The updated project.

Return type *Project*

Project Collection

class tamr_unify_client.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)
```

create(*creation_spec*)
Create a Project in Tamr

Parameters **creation_spec** (*dict[str, str]*) – Project creation specification should be formatted as specified in the Public Docs for Creating a Project.

Returns The created Project

Return type *Project*

delete_by_resource_id(*resource_id*)
Deletes a resource from this collection by resource ID.

Parameters **resource_id** (*str*) – The resource ID of the resource that will be deleted.

Returns HTTP response from the server.

Return type requests.Response

Project Step

class tamr_unify_client.project.step.**ProjectStep**(*client*, *data*)
A step of a Tamr project. This is not a *BaseResource* because it has no API path and cannot be directly retrieved or modified.

See https://docs.tamr.com/reference#retrieve-downstream-dataset-usage

Parameters

- **client** (*Client*) – Delegate underlying API calls to this client.
- **data** (dict) – The JSON body containing project step information.

project_step_id

Type str

project_step_name

Type str

project_name

Type str

type
A Tamr project type, listed in https://docs.tamr.com/reference#create-a-project.

Type str

project()
Retrieves the *Project* this step is associated with.

Returns This step's project.

Return type *Project*

Raises

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

Index

A

B

C

D

E

F

G

H

I

K

M

N

O

P

R

S

T

U

V

W