

---

# Tamr - Python Client Documentation

*Release 0.0*

**Tamr**

**Feb 05, 2021**



# CONTENTS

<b>1 Example</b>	<b>3</b>
<b>2 User Guide</b>	<b>5</b>
2.1 FAQ . . . . .	5
2.2 Installation . . . . .	6
2.3 Quickstart . . . . .	7
2.4 Secure Credentials . . . . .	8
2.5 Workflows . . . . .	10
2.6 Creating and Modifying Resources . . . . .	11
2.7 Logging . . . . .	13
2.8 Geospatial Data . . . . .	14
2.9 Pandas Workflow . . . . .	16
2.10 Advanced Usage . . . . .	20
<b>3 Reference</b>	<b>23</b>
3.1 Reference . . . . .	23
<b>4 Contributor Guide</b>	<b>73</b>
4.1 Contributor guide . . . . .	73
<b>5 BETA</b>	<b>89</b>
5.1 BETA . . . . .	89
<b>Python Module Index</b>	<b>117</b>
<b>Index</b>	<b>119</b>



[View on Github](#)



**EXAMPLE**

```
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 interact 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()
```



## 2.1 FAQ

### 2.1.1 What version of the Python Client should I use?

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

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.

Otherwise, check the [Releases](#) to see:

- what new features and bug fixes are available in newer versions
- which breaking changes (if any) will require changes in your code to get those new features and bug fixes

Note: You do not need to reason about the Tamr API version nor the the Tamr app/server version.

### 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 [\[Raw HTTP requests and Unversioned API Access\]](#)([<user-guide/advanced-usage:Raw 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')
```

Similarly, if you know the name of a specific resource, you can ask for it directly via the `by_name` methods exposed by collections.

E.g. To fetch the project with name 'Number 1':

```
project = tamr.projects.by_name('Number 1')
```

---

**Note:** If working with projects across Tamr instances for migrations or promotions, use external IDs (via `by_external_id`) instead of name (via `by_name`).

---

### 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

with open("path/to/credentials.yaml") as f: # replace with your credentials.yaml path
    creds = yaml.safe_load(f)

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

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()
```

(continues on next page)

(continued from previous page)

```

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

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

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

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

```

## 2.6 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")
    .with_external_id("tamrProject1")
)
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", dtype=str) # string is the recommended data type
dataset = tamr.datasets.create_from_dataframe(df, primary_key_name="primary key name",
↳ dataset_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 Tamr 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 Logging

**IMPORTANT** Make sure to configure logging BEFORE importing from 3rd party libraries. Logging will use the first configuration it finds, and if a library configures logging before you, your configuration will be ignored.

To configure logging, simply follow the official Python logging [HOWTO](#).

For example:

```
# script.py
import logging

logging.basicConfig(filename="script.log", level=logging.INFO)

# configure logging before other imports

from tamr_unify_client import Client
from tamr_unify_client.auth import UsernamePasswordAuth

auth = UsernamePasswordAuth("my username", "my password")
tamr = Client(auth, host="myhost")

for p in tamr.projects:
    print(p)

for d in tamr.datasets:
    print(d)

# should cause an HTTP error
tamr.get("/invalid/api/path").successful()
```

This will log all API requests made and print the response bodies for any requests with HTTP error codes.

If you want to **only** configure logging for the Tamr Client:

```
import logging
logger = logging.getLogger('tamr_unify_client')
logger.setLevel(logging.INFO)
logger.addHandler(logging.FileHandler('tamr-client.log'))

# configure logging before other imports

from tamr_unify_client import Client
from tamr_unify_client import UsernamePasswordAuth

# rest of script goes here
```

## 2.8 Geospatial Data

### 2.8.1 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)
```

Note that there are currently some limitations to GeoPandas’ implementation of the Geo Interface. See below for more details.

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`.

## 2.8.2 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.

## 2.8.3 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.

## 2.8.4 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.

## 2.8.5 Note on GeoPandas data access

There is a current limitation in [GeoPandas](#) that causes the feature's ID field to be ignored in certain scenarios. The Tamr primary key is stored in this field. The result is that when loading data and updating records through the `dataset.from_geo_features()` method, records will not be overwritten as anticipated.

This issue can be circumvented by loading features into GeoPandas by re-inserting the id field into the data.

```
my_dataset = client.datasets.by_name("my_dataset")
for feature in my_dataset.itergeofeatures():
    primary_key = feature['id']
    df = gpd.GeoDataFrame.from_features([feature])
    do_something(df)
    geo.index = [primary_key]
    my_dataset.from_geo_features(df)
```

Alternatively, it is possible to load the full dataset as follows:

```
my_dataset = client.datasets.by_name("my_dataset")
def geopandas_dataset(dataset):
    for feature in dataset.itergeofeatures():
        feature['properties']['primary_key'] = feature['id']
        yield feature
df = gpd.GeoDataFrame.from_features(geopandas_dataset(my_dataset))
df.set_index('primary_key')
do_something(df)
my_dataset.from_geo_features(df)
```

## 2.9 Pandas Workflow

### 2.9.1 Connecting To Tamr

Connecting to a Tamr instance:

```
import os
import pandas as pd
from tamr_unify_client import Client
from tamr_unify_client.auth import UsernamePasswordAuth

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

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

## 2.9.2 Load dataset as Dataframe

### Loading: In Memory

Loading a dataset as a pandas dataframe is possible via the `from_records()` method that pandas provides. An example is shown below:

```
my_dataset = tamr.datasets.by_name("my_tamr_dataset")
df = pd.DataFrame.from_records(my_dataset.records())
```

This will construct a pandas dataframe based on the records that are streamed in, and stored in the pandas dataframe. Once all records have been loaded, you will be able to interact with the dataframe normally.

Note that as values are typically represented inside arrays within Tamr, the values will be encapsulated lists inside the dataframe. You can use traditional methods in pandas to deal with this; for example by calling `.explode()`, or extracting specific elements.

### Loading: Streaming

When working with large datasets it is sometimes better not to work in memory, but to iterate through a dataset, rather than load the entire dataset at once. Since `dataset.records()` is a generator, this can easily be done as follows:

```
output = []
for record in dataset.records():
    single_record_df = pd.DataFrame.from_records(record)
    output.append(do_something(single_record_df))
```

### Custom Generators

In order to customise the data loaded into the pandas dataframe, it is possible to customise the generator object `dataset.records()` by wrapping it in a different generator.

For example, it is possible to automatically flatten all lists with a length of one, and apply this to the `dataset.records()` generator as follows:

```
def unlist(lst):
    """
    If object is a list of length one, return first element.
    Otherwise, return original object.
    """
    if isinstance(lst, list) and len(lst) is 1:
        return lst[0]
    else:
        return lst

def dataset_to_pandas(dataset):
    """
    Incorporates basic unlisting for easy transfer between Tamr and Pandas.
    """
    for record in dataset.records():
        for key in record:
            record[key] = unlist(record[key])
        yield record
```

(continues on next page)

(continued from previous page)

```
df = pd.DataFrame.from_records(dataset_to_pandas(my_dataset))
```

Similarly, it is possible to filter to extracting only certain attributes, by specifying this in the generator:

```
def filter_dataset_to_pandas(dataset, colnames):
    """
    Filter the dataset to only the primary key and the columns specified as a list in
    ↪ colnames.
    """
    assert isinstance(colnames, list)
    colnames = dataset.key_attribute_names + colnames if dataset.key_attribute_
    ↪ names[0] not in colnames else colnames
    for record in dataset.records():
        yield {k: unlist(v) for k, v in record.items() if k in colnames}

df = pd.DataFrame.from_records(filter_dataset_to_pandas(my_dataset, ['City', 'new_attr
    ↪']))
```

Note that upserting these records back to the original Tamr Dataset would overwrite the existing records and attributes, and cause loss of the data stored in the removed attributes.

## 2.9.3 Upload Dataframe as Dataset

### Create New Dataset

To create a new dataset and upload data, the convenience function `datasets.create_from_dataframe()` can be used. Note that Tamr will throw an error if columns aren't generally formatted as strings. (The exception being geospatial columns. For that, see the geospatial examples.)

To format values as strings while preserving null information, specify `dtype=object` when creating a dataframe from a csv file.

```
df = pd.read_csv("my_file.csv", dtype=object)
```

Creating the dataset is as easy as calling:

```
tamr.datasets.create_from_dataframe(df, 'primaryKey', 'my_new_dataset')
```

For an already-existing dataframe, the columns can be converted to strings using:

```
df = df.astype(str)
```

Note, however, that converting this way will cause any NaN or None values to become strings like 'nan' that will persist into the created Tamr dataset.

## Changing Values

### Making Changes: In Memory

When making changes to a dataset that was loaded as a dataframe, changes can be pushed back to Tamr using the `dataset.upsert_from_dataframe()` method as follows:

```
df = pd.DataFrame.from_records(my_dataset.records())
df['column'] = 'new_value'
my_dataset.upsert_from_dataframe(df, primary_key_name='primary_key')
```

### Making Changes: Streaming

For larger datasets it might be better to stream the data and apply changes while iterating through the dataset. This way the full dataset does not need to be loaded into memory.

```
for record in dataset.records():
    single_record_df = pd.DataFrame.from_records(record)
    single_record_df['column_to_change'] = 'new_value'
    dataset.upsert_from_dataframe(single_record_df, primary_key_name='primary_key')
```

## Adding Attributes

When making changes to dataframes, new dataframe columns are not automatically created as attributes when upserting records to Tamr. In order for these changes to be recorded, these attributes first need to be created.

One way of creating these for source datasets automatically would be as follows:

```
def add_missing_attributes(dataset, df):
    """
    Detects any attributes in the dataframe that aren't in the dataset and attempts_
    ↪to add them (as strings).
    """
    existing_attributes = [att.name for att in dataset.attributes]
    new_attributes = [att for att in df.columns.to_list() if att not in existing_
    ↪attributes]

    if not new_attributes:
        return

    for new_attribute in new_attributes:
        attr_spec = {"name": new_attribute,
                    "type": {"baseType": "ARRAY", "innerType": {"baseType": "STRING"}
    ↪},
                    }
        dataset.attributes.create(attr_spec)

add_missing_attributes(my_dataset, df)
```

## 2.9.4 Troubleshooting

When running into errors upon loading `dataset.records()` into a pandas dataframe, it is good to consider the following steps. To extract a single record, the following code can be used to provide a minimal reproducible example:

```
record = next(dataset.records())
print(record)
```

### Parsing

Tamr allows for more variety in attribute names and contents than pandas does. In most cases pandas can load data correctly, but it is possible to modify the parsing using a custom generator as shown above. An example below changes an attribute name, and extracts only the first element:

```
def custom_parser(dataset):
    for record in dataset.records():
        record['pandas_column_name'] = record.pop('dataset_attribute_name')
        record['first_element_of_column'] = record['multi_value_column'][0]
        yield record

df = pd.DataFrame.from_records(custom_parser(dataset))
```

## 2.10 Advanced Usage

### 2.10.1 Asynchronous Operations

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

E.g.:

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

### 2.10.2 Raw 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)
```

## 3.1 Reference

### 3.1.1 Attributes

#### Attribute

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

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

**property** `relative_id`  
str

**Type** `type`

**property** `name`  
str

**Type** `type`

**property** `description`  
str

**Type** `type`

**property** `type`  
*AttributeType*

**Type** `type`

**property** `is_nullable`  
bool

**Type** `type`

**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`

**property resource\_id**  
`str`

**Type** `type`

## Attribute Spec

**class** `tamr_unify_client.attribute.resource.AttributesSpec` (*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

**property** **base\_type**

*str*

**Type** *type*

**property** **inner\_type**

*AttributeType*

**Type** *type*

**property attributes**

list[*SubAttribute*]

Type *type*

**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 (name, type,
                                                         is_nullable, _json,
                                                         description=None)
```

An attribute which is itself a property of another attribute.

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

### Parameters

- **name** (*str*) – Name of sub-attribute
- **description** (*Optional[str]*) – Description of sub-attribute
- **type** (*AttributeType*) – See <https://docs.tamr.com/reference#attribute-types>
- **is\_nullable** (*bool*) – If this sub-attribute can be null

```
static from_json (data)
```

Create a SubAttribute from JSON data.

**Parameters** *data* (*Dict[str, Any]*) – JSON data received from Tamr server.

**Return type** *SubAttribute*

## 3.1.2 Auth

```
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)
```

## 3.1.3 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*

**property 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*

**property description**

str

**Type** *type*

**property external\_id**

str

**Type** *type*

**input\_datasets** ()

Retrieve a collection of this project's input datasets.

**Returns** The project's input datasets.

**Return type** *DatasetCollection*

**property name**

str

**Type** *type*

**property relative\_id**

str

**Type** *type*

**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*

**property resource\_id**

str

**Type** *type*

**spec** ()

Returns this project's spec.

**Returns** The spec for the project.

**Return type** *ProjectSpec*

**property type**

[//docs.tamr.com/reference#create-a-project](https://docs.tamr.com/reference#create-a-project).

**Type** *str*

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

**unified\_dataset** ()

Unified dataset for this project.

**Returns** Unified dataset for this project.

**Return type** *Dataset*

## Categories

### Category

**class** tamr\_unify\_client.categorization.category.resource.**Category** (*client, data, alias=None*)

A category of a taxonomy

**property name**

str

**Type** *type*

**property description**

str

**Type** *type*

**property path**

list[str]

**Type** *type*

**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*

**property relative\_id**

str

**Type** *type*

**property resource\_id**

str

**Type** *type*

## 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

**property name**

*str*

**Type** *type*

**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*

**property relative\_id**

*str*

**Type** *type*

**property resource\_id**

*str*

**Type** *type*

### 3.1.4 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** (*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')
- **protocol** (*str*) – Either 'http' or 'https'
- **port** (*Optional[int]*) – Tamr instance main port
- **base\_path** (*str*) – Base API path. Requests made by this client will be relative to this path.
- **session** (*Optional[Session]*) – Session to use for API calls. If none is provided, will use a new `requests.Session`.

#### Example

```
>>> from tamr_unify_client import Client
>>> from tamr_unify_client.auth import UsernamePasswordAuth
>>> auth = UsernamePasswordAuth('my username', 'my password')
>>> tamr_local = Client(auth) # on http://localhost:9100
>>> tamr_remote = Client(auth, protocol='https', host='10.0.10.0') # on https://
↳10.0.10.0:9100
>>> tamr_remote = Client(auth, protocol='https', host='10.0.10.0', port=None) #
↳on https://10.0.10.0
```

#### property origin

HTTP origin i.e. <protocol>://<host>[:<port>].

For additional information, see [MDN web docs](#) .

**Return type** *str*

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

Sends a request to Tamr.

The URL for the request will be <origin>/<base\_path>/<endpoint>. The request is authenticated via `Client.auth`.

#### Parameters

- **method** (*str*) – The HTTP method to use (e.g. 'GET' or 'POST')
- **endpoint** (*str*) – API endpoint to call (relative to the Base API path for this client).

**Return type** *Response*

**Returns** HTTP response from the Tamr server

**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.

**property projects**  
Collection of all projects on this Tamr instance.  
**Return type** *ProjectCollection*  
**Returns** Collection of all projects.

**property datasets**  
Collection of all datasets on this Tamr instance.  
**Return type** *DatasetCollection*  
**Returns** Collection of all datasets.

### 3.1.5 Datasets

#### Dataset

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

**property name**  
str  
**Type** *type*

**property external\_id**  
str  
**Type** *type*

**property description**  
str  
**Type** *type*

**property version**  
str  
**Type** *type*

**property tags**  
list[str]  
**Type** *type*

**property key\_attribute\_names**  
list[str]  
**Type** *type*

**property attributes**  
Attributes of this dataset.  
**Returns** Attributes of this dataset.

**Return type** `AttributeCollection`

**upsert\_from\_dataframe** (*df*, \*, *primary\_key\_name*, *ignore\_nan=None*)

Upserts a record for each row of *df* with attributes for each column in *df*.

**Parameters**

- **df** (*pd.DataFrame*) – The data to upsert records from.
- **primary\_key\_name** (*str*) – The name of the primary key of the dataset. Must be a column of *df*.
- **ignore\_nan** (*Optional[bool]*) – Legacy parameter that does nothing. Deprecated.

**Return type** `dict`

**Returns** JSON response body from the server.

**Raises** `KeyError` – If *primary\_key\_name* is not a column in *df*.

**upsert\_records** (*records*, *primary\_key\_name*, \*, *ignore\_nan=False*)

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.
- **ignore\_nan** (*bool*) – Whether to convert *NaN* values to *null* when upserting records. If *False* and *NaN* is found this function will fail. Deprecated.

**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

**Returns** JSON response body from server.

**Return type** `dict`

**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 unintended 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]`

**property relative\_id**

`str`

**Type** `type`

**property resource\_id**

`str`

**Type** `type`

## 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 unintended 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=None)**

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*) – Legacy parameter that does nothing

**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.

property dataset_name
    The name of the associated dataset.
    Type str
    Return type str

property relative_dataset_id
    The relative dataset ID of the associated dataset.
    Type str
    Return type str

property is_up_to_date
    Whether the associated dataset is up to date.
    Type bool
    Return type bool

property profiled_data_version
    The profiled data version.
    Type str
    Return type str

property profiled_at
    Info about when profile info was generated.
    Type dict
    Return type dict

property simple_metrics
    Simple metrics for profiled dataset.
```

**Type** `list`

**Return type** `list`

**property** `attribute_profiles`

Simple metrics for profiled dataset.

**Type** `list`

**Return 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`

**property** `relative_id`

`str`

**Type** `type`

**property** `resource_id`

`str`

**Type** `type`

## Dataset Status

**class** `tamr_unify_client.dataset.status.DatasetStatus` (*client, data, alias=None*)

Streamability status of a Tamr dataset.

**property** `dataset_name`

The name of the associated dataset.

**Type** `str`

**Return type** `str`

**property** `relative_dataset_id`

The relative dataset ID of the associated dataset.

**Type** `str`

**Return type** `str`

**property** `is_streamable`

Whether the associated dataset is available to be streamed.

**Type** `bool`

**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`

**property relative\_id**

str

**Type** `type`

**property resource\_id**

str

**Type** `type`

## Dataset URI

**class** `tamr_unify_client.dataset.uri.DatasetURI` (*client, uri*)

Identifier of a dataset.

**Parameters**

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

**property resource\_id**

str

**Type** `type`

**property relative\_id**

str

**Type** `type`

**property uri**

str

**Type** `type`

**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>

**property relative\_id**

str

**Type** `type`

**property usage**

*DatasetUse*

Type *type*

**property dependencies**

list[*DatasetUse*]

Type *type*

**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*

**property resource\_id**

str

Type *type*

## 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.

**property dataset\_id**

str

Type *type*

**property dataset\_name**

str

Type *type*

**property input\_to\_project\_steps**

list[*ProjectStep*]

Type *type*

**property output\_from\_project\_steps**

list[*ProjectStep*]

Type *type*

**dataset ()**

Retrieves the *Dataset* this use represents.

**Returns** The dataset being used.

**Return type** *Dataset*

### 3.1.6 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*

**property** `relative_id`

str

**Type** *type*

**property** `resource_id`

str

**Type** *type*

### 3.1.7 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`

**property relative\_id**

`str`

**Type** `type`

**property resource\_id**

`str`

**Type** `type`

## Estimated Pair Counts

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

Estimated Pair Counts info for Mastering Project

**property is\_up\_to\_date**

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

**Return type** `bool`

**property 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]`

**property 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`

**property relative\_id**

str

**Type** `type`

**property resource\_id**

str

**Type** `type`

## 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*

**property 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*

**property description**

str

**Type** *type*

**property external\_id**

str

**Type** *type*

**input\_datasets** ()

Retrieve a collection of this project's input datasets.

**Returns** The project's input datasets.

**Return type** *DatasetCollection*

**property name**

str

**Type** *type*

**property relative\_id**

str

**Type** *type*

**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*

**property resource\_id**

str

**Type** *type*

**spec()**  
 Returns this project's spec.  
**Returns** The spec for the project.  
**Return type** *ProjectSpec*

**property type**  
 //docs.tamr.com/reference#create-a-project.  
**Type** *str*  
**Type** A Tamr project type, listed in https

**unified\_dataset()**  
 Unified dataset for this project.  
**Returns** Unified dataset for this project.  
**Return type** *Dataset*

## Published Clusters

### 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.

**property name**  
*str*

**Type** *type*

**property value**  
*str*

**Type** *type*

### 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*.

**property id**  
*str*

**Type** *type*

**property versions**  
 list[*PublishedClusterVersion*]

**Type** *type*

## Published Cluster Configuration

**class** tamr\_unify\_client.mastering.published\_cluster.configuration.**PublishedClustersConfigur**

The configuration of published clusters in a project.

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

**property** relative\_id

str

Type type

**property** versions\_time\_to\_live

str

Type type

**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.PublishedClustersConfigurationSpec

**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`

**property** resource\_id

str

Type type

## 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.

**property** version

str

Type type

**property** timestamp

str

Type type

**property** name

str

Type type

**property** metrics

list[*Metric*]

Type `type`

property `record_ids`  
list[dict[str, str]]

Type `type`

## 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*.

property `entity_id`  
str

Type `type`

property `source_id`  
str

Type `type`

property `origin_entity_id`  
str

Type `type`

property `origin_source_id`  
str

Type `type`

property `versions`  
list[*RecordPublishedClusterVersion*]

Type `type`

## Record Published Cluster Version

**class** `tamr_unify_client.mastering.published_cluster.record_version.RecordPublishedClusterVersion`  
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.

property `version`  
str

Type `type`

property `timestamp`  
str

Type `type`

`property cluster_id`  
str

Type `type`

### 3.1.8 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()`.

**classmethod** `from_resource_id` (*client*, *resource\_id*)

Get an operation by resource ID.

**Parameters**

- **client** (*Client*) – Delegate underlying API calls to this client.
- **resource\_id** (*str*) – The ID of the operation

**Returns** The specified operation

**Return type** *Operation*

**classmethod** `from_response` (*client*, *response*)

Handle idiosyncrasies in constructing Operations from Tamr responses. When a Tamr API call would start an operation, but all results that would be produced by that operation are already up-to-date, Tamr returns *HTTP 204 No Content*

To make it easy for client code to handle these API responses without checking the response code, this method will either construct an Operation, or a dummy *NoOp* operation representing the 204 Success response.

**Parameters**

- **client** (*Client*) – Delegate underlying API calls to this client.
- **response** (*requests.Response*) – HTTP Response from the request that started the operation.

**Returns** Operation

**Return type** *Operation*

**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*

**property type**

str

**Type type**

**property description**

str

**Type type**

**property state**

Server-side state of this operation.

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

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

**Usage:**

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

**poll()**

Poll this operation for server-side updates.

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

**Returns** Updated representation of this operation.

**Return type** *Operation*

**wait** (*poll\_interval\_seconds=3, timeout\_seconds=None*)

Continuously polls for this operation's server-side state.

**Parameters**

- **poll\_interval\_seconds** (*int*) – Time interval (in seconds) between subsequent polls.
- **timeout\_seconds** (*int*) – Time (in seconds) to wait for operation to resolve.

**Raises** **TimeoutError** – If operation takes longer than *timeout\_seconds* to resolve.

**Returns** Resolved operation.

**Return type** *Operation*

**succeeded()**

Convenience method for checking if operation was successful.

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

**Return type** bool

**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

**property relative\_id**

str

**Type** type

**property resource\_id**

str

**Type** type

## 3.1.9 Projects

### Attribute Configurations

#### Attribute Configuration

**class** tamr\_unify\_client.project.attribute\_configuration.resource.**AttributeConfiguration** (*client*, *data*, *alias*)

The configurations of Tamr Attributes.

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

**property relative\_id**

str

**Type** type

**property id**

str

**Type** type

**property relative\_attribute\_id**

str

**Type** type

**property attribute\_role**

str

**Type** type

**property similarity\_function**

str

**Type** type

**property enabled\_for\_ml**  
bool

Type `type`

**property tokenizer**  
str

Type `type`

**property numeric\_field\_resolution**  
list

Type `type`

**property attribute\_name**  
str

Type `type`

**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`

**property resource\_id**  
str

Type `type`

## 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.AttributeConfigurationCollection`

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 Mappings

### 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

**property** **id**

*str*

**Type** *type*

**property** **relative\_id**

*str*

**Type** *type*

**property** **input\_attribute\_id**

*str*

**Type** *type*

**property** **relative\_input\_attribute\_id**

*str*

**Type** *type*

**property** **input\_dataset\_name**

*str*

**Type** *type*

**property** `input_attribute_name`  
 str  
 Type `type`

**property** `unified_attribute_id`  
 str  
 Type `type`

**property** `relative_unified_attribute_id`  
 str  
 Type `type`

**property** `unified_dataset_name`  
 str  
 Type `type`

**property** `unified_attribute_name`  
 str  
 Type `type`

**property** `resource_id`  
 str  
 Type `type`

**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` (*client*, *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.

**property name**

str

**Type** `type`

**property external\_id**

str

**Type** `type`

**property description**

str

**Type** `type`

**property type**

`//docs.tamr.com/reference#create-a-project.`

**Type** `str`

**Type** A Tamr project type, listed in [https](https://docs.tamr.com/reference#create-a-project)

**property 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`

**property relative\_id**

str

**Type** `type`

**property resource\_id**

str

**Type** `type`

## 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)
```

**by\_name** (*project\_name*)

Get project by name

Fetches a specific project in this collection by exact-match on name.

**Parameters** **project\_name** (*str*) – Name of the desired project.

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

**Return type** *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.

**property** `project_step_id`

*str*

**Type** *type*

**property** `project_step_name`

*str*

**Type** *type*

**property** `project_name`

str

**Type** `type`

**property** `type`

[//docs.tamr.com/reference#create-a-project](https://docs.tamr.com/reference#create-a-project).

**Type** `str`

**Type** A Tamr project type, listed in [https](https://docs.tamr.com/reference#create-a-project)

**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.



## CONTRIBUTOR GUIDE

### 4.1 Contributor guide

Thank you for learning how to contribute to Tamr's Python Client! Your contribution will help you and many others in the Tamr community. Before you begin, make sure you are viewing the [latest version of Contributor Guide](#).

#### 4.1.1 Feedback

Before submitting a new issue, [you can search existing issues](#). If the bug/feature has been submitted already, leave a like on the description of the Github Issue. Maintainers will consider number of likes when prioritizing issues.

##### Bug reports

Submit bug reports as [Github issues](#).

##### Feature requests

Submit feature requests as [Github issues](#).

#### 4.1.2 Documentation

##### How to write docs

Before you begin to add content, decide which of the three types of content you want to add:

1. Tutorial
2. How-To guide
3. Explanation

---

**Note:** There is fourth type of content, known as Reference.

For the Tamr Client, you don't need to add reference topics manually because reference documentation for the Tamr Client is generated automatically based on the source code.

For more details, see Reference description below.

---

For more information about each type of content, see the following descriptions. Also see [Divio's documentation system manual](#).

### Tutorial

Tutorials are learning-oriented and ...

- Must include an end-to-end walkthrough for a specific use case, such as “Tutorial: Deduplicating buildings in Cambridge”.
- Must have a clearly stated goal and allow the users to achieve it after they complete the steps in the tutorial.
- Must provide the sample data and input configuration that are necessary for the user to complete the tutorial. Include this information upfront, at the start of your tutorial.
- Must be self-contained, but can include links to procedures described elsewhere in this documentation.

Tutorials are useful if the use case is both simple and in high demand. Not every use case deserves a tutorial. Before writing a tutorial, think first of a use case that has a high learning value, and then prepare the assets needed to complete your tutorial, such as a sample dataset and sample configuration.

Tutorials are in high demand. If you write a good one, many users will reference it and thank you for your work!

### How-To

How-Tos are task-oriented and ...

- Must include a list of numbered steps, known as a task, or a procedure, to help users complete a specific, domain-agnostic task, such as running a request, copying a file, installing, exporting, or other. For example, you can create a task titled “How to stream datasets out of Tamr”.
- Must include a context paragraph, such as “It is often useful to stream datasets from Tamr, to load them into business analytics applications, such as Tableau, for analysis.” Context may also include checks needed to be in place before users start the task, and links to related concepts. Context must provide information needed to begin the task, such as, it can list the host and port URL at which the endpoint for the service is served.
- Must include a stem sentence, such as: “To stream a dataset out of Tamr:” The stem sentence is followed by numbered steps.
- Must include a numbered list of steps where each step must begin with an imperative verb, such as: “Run the following curl request.”, or “Save the file”. For more examples see [Use Imperatives in Procedures](#).

### Explanation

Explanations are understanding-oriented and ...

- Must explain a single concept of the Tamr Python client. If you'd like to write another concept, create it separately.
- Must [keep sentences short](#).
- May include examples of code or text examples.

## Reference

Reference is information-oriented.

It is something that users cannot remember and want to be able to refer to, often. Reference provides details, such as configuration parameters for a particular method or call. It never contains tasks, or concepts. Reference is often automatically-generated from code, to ensure it is up-to-date and accurate at all times.

---

**Note:** Our reference documentation is automatically generated by [autodoc](<https://www.sphinx-doc.org/en/master/usage/extensions/autodoc.html>) based on type annotations and docstrings in the source code.

---

### 4.1.3 Code

#### Installation

##### Prerequisites

1. Install build dependencies for pyenv
2. Install pyenv
3. Install poetry

##### Clone + install

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

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

2. Install all Python versions in `.python-version`:

Dev tasks will use these Python versions.

```
# run `pyenv install` for each line in `.python-version`
cat .python-version | xargs -L 1 pyenv install
```

3. Install project dependencies via poetry:

```
poetry install
```

##### Run dev tasks

This project uses `nox`.

Since `nox` will be running inside of a `poetry` environment (to guarantee you are running the same version of `nox` as everyone else), we recommend adding the following alias to your `.bashrc` / `.zshrc` to save you some keystrokes:

```
alias prn='poetry run nox'
```

To run all checks:

```
prn # with alias
poetry run nox # without alias
```

### Linting

To run linter:

```
prn -s lint # with alias
poetry run nox -s lint # without alias
```

### Formatting

To run formatter:

```
prn -s format # with alias
poetry run nox -s format # without alias
```

Run the formatter with the `--fix` flag to autofix formatting:

```
prn -s format -- --fix # with alias
poetry run nox -s format -- --fix # without alias
```

### Typechecks

To run typechecks:

```
prn -s typecheck # with alias
poetry run nox -s typecheck # without alias
```

### Tests

To run all tests:

```
prn -s test # with alias
poetry run nox -s test # without alias
```

---

To run tests for a specific Python version e.g. 3.6:

```
prn -s test-3.6 # with alias
poetry run nox -s test-3.6 # without alias
```

See `nox --list` for more details.

---

To run specific tests, see [these pytest docs](#) and pass `pytest` args after `--` e.g.:

```
prn -s test -- tests/unit/test_attribute.py # with alias
poetry run nox -s test -- tests/unit/test_attribute.py # without alias
```

## Docs

To build the docs:

```
prn -s docs # with alias
poetry run nox -s docs # without alias
```

After docs are build, view them by:

```
open -a 'firefox' docs/_build/index.html # open in Firefox
open -a 'Google Chrome' docs/_build/index.html # open in Chrome
```

## Configure your Text Editor

### Atom

### VS Code

### IntelliJ

## Architectural Decision Records

Important architectural decisions are logged as Architectural Decision Records (ADRs) and are housed here.

For more on ADRs, see:

To author new ADRs, we recommend [adr-tools](#).

## ADRs

### 1. Record architecture decisions

Date: 2020-08-14

### Status

Accepted

### Context

We need to record the architectural decisions made on this project.

## Decision

We will use Architecture Decision Records, as [described by Michael Nygard](#).

## Consequences

See Michael Nygard's article, linked above. For a lightweight ADR toolset, see Nat Pryce's [adr-tools](#).

## 2. Linting and formatting

Date: 2019-01-14

## Status

Accepted

## Context

Inconsistent code formatting slows down development and the review process.

Code should be linted for things like:

- unused imports and variables
- consistent import order

Code formatting should be done automatically or programmatically, taking the burden off of reviewers.

## Decision

For linting, use [flake8](#) and [flake8-import-order](#).

For formatting, use [black](#).

## Consequences

All linting and formatting are enforced programmatically.

Most linting and formatting errors can be autofixed.

Text editors and IDEs are able to integrate with our linting and formattings tools to automatically fix (most) errors on save.

### 3. Reproducibility

Date: 2019-06-05

#### Status

Accepted

#### Context

Reproducing results from a program is challenging when operating systems, language versions, and dependency versions can vary.

For this codebase, we will focus on consistent Python versions and dependency versions.

#### Decision

Manage multiple Python versions via [pyenv](#).

Manage dependencies via [poetry](#).

Define tests via [nox](#).

Run tests in automation/CI via [Github Actions](#).

#### Consequences

This solution lets us:

- keep track of [abstract and concrete versions](#) for dependencies (think `.lock` file)
- locally test against multiple Python versions
- run the same tests locally as we do in [Continuous Integration \(CI\)](#)
- easily view CI test results within the review context

### 4. Documentation and docstrings

Date: 2019-10-03

#### Status

Accepted

### Context

Documentation can take four forms:

1. Explanation
2. Tutorial
3. How-to
4. Reference

We need a way to author and host prosey documentation and generate reference docs based on source code.

### Decision

Doc compilation will be done via [sphinx](#).

Prosey documentation (1-3) via [recommonmark](#).

Reference documentation (4) will be generated based on type annotations and docstrings via:

- Automatic docs based on docstrings via [sphinx-autodoc](#), [sphinx-autodoc-typehints](#)
- Google-style docstrings via [napoleon](#)
- Hosting on [ReadTheDocs](#) (RTD)
- Build docs in CI and fail on errors or warnings.

### Consequences

Prosey documentation can be written in Markdown (.md), which is more familiar to our contributors than .rst format.

Reference doc generation makes docs more maintainable and consistent with actual code.

Google-style docstrings are easier to read than sphinx-style docstrings.

RTD natively compiles documentation using sphinx and simultaneously hosts docs at each version.

## 5. Composable functions

Date: 2019-11-01

### Status

Accepted

## Context

We need a reasonable tradeoff between ease-of-use and maintainability.

Specifically, we need composable, combinable units that can be improved independently.

### Approach 1: Classes + Methods

One approach is to embrace Object-Oriented Programming (OOP) with fluent interfaces (i.e. method chaining):

```
project
    .create(...)
    .update(...)
    .delete(...)
```

Characteristics:

- Ease-of-use is maximized, but this requires each method to `return self`.
- Also, this approach implies that if a function can be called with X different object types, each of those object types should have a corresponding method that applies that functionality and then `return self`.

How to enforce these characteristics?

Any solution will be a tax on maintainability, as code that adheres to these characteristics will include many non-semantic lines simply going through the motions of `return self` and copying function usage into dedicated methods for each class.

### Approach 2: Types + Functions

Another approach is to embrace a functional programming style: simple types and functions (no methods).

Usage is not as terse as for OOP:

```
p = tc.project.create(...)
u = tc.project.update(p, ...)
d = tc.project.delete(p, ...)
```

Characteristics:

- Ease-of-use is not optimized, but still reasonable.
  - With tab-completion, ease-of-use is comparable to OOP.
- Each type can be made immutable
- Each function can be made pure
- Functionality can be shared by calling the same function in user-land, not copying function calls in contributor-land.

### Decision

Use `@dataclass(frozen=True)` to model types and plain Python modules and functions to capture business logic.

### Consequences

Immutable types and pure functions make the code much easier to reason about, drastically cutting down the time to ramp up and debug.

Functions are easily composable without accumulating undesired side-effects, unlike methods.

Note that not all types and functions *have* to be immutable and pure, but immutable types and pure functions should be the default.

If there are good reasons to make exceptions, we can do so, but we should include comments to explain why that exception was made.

## 6. Type-checking

Date: 2020-01-29

### Status

Accepted

### Context

Static type-checking is available for Python, making us of the type annotations already in the codebase.

### Decision

Type-check via `mypy`.

### Consequences

Testing is still important, but type checking helps to eliminate bugs via static checking, even for parts of the code not exercised during tests.

Additionally, type-checking relies on our type annotations, ensuring that the annotations are correct and complete.

## 7. tamr\_client package

Date: 2020-04-03

### Status

Accepted

### Context

We have an existing userbase that relies on `tamr_unify_client` and cannot painlessly make backwards-incompatible changes.

But, we want to rearchitect this codebase as a *library of composable functions*.

### Decision

Implement rearchitected design as a new package named `tamr_client`.

Require the `TAMR_CLIENT_BETA=1` feature flag for `tamr_client` package usage.

Warn users who attempt to use `tamr_client` package to opt-in if they want to beta test the new design.

### Consequences

Continue to support `tamr_unify_client`, but any new functionality:

- must be included in `tamr_client`
- may be included in `tamr_unify_client`

Users are required to explicitly opt-in to new features, preserving backward compatibility for current users.

Once we reach feature parity with `tamr_unify_client`, we can undergo a deprecation cycle and subsequently remove `tamr_unify_client`.

## 8. Standardized imports

Date: 2020-06-01

### Status

Accepted

### Context

Python has many ways of importing:

```
# option 1: import module

# option 1.a
import foo.bar.bazaar as baz
baz.do_the_thing()

# option 1.b
from foo.bar import bazaar as baz
baz.do_the_thing()

# option 2: import value
from foo.bar.bazaar import do_the_thing
do_the_thing()
```

Not to mention that each of these styles may be done with relative imports (replacing `foo.bar` with `.bar` if the `bar` package is a sibling).

Confusingly, Option 1.a and Option 1.b are *conceptually* the same, but mechanically there are [subtle differences](#).

### Decision

Imports within `tamr_client`:

- Must import statements for modules, classes, and exceptions
- Must `from foo import bar` instead of `import foo.bar as bar`
- Must not import functions directly. Instead import the containing module and use `module.function(...)`
- Must not use relative imports. Use absolute imports instead.

### Consequences

Standardized import style helps linter correctly order imports.

Choosing import styles is a syntactic choice without semantic meaning. Removing this choice should speed up development and review.

## 9. Separate types and functions

Date: 2020-06-29

## Status

Accepted

## Context

Code must be organized to be compatible with:

- Static type-checking via [mypy](#)
- Runtime execution during normal usage and running tests via [pytest](#)
- Static doc generation via [sphinx-autodoc-typehints](#)

Additionally:

- Functions should be able to refer to any type
- Most types depend on other types non-recursively, but some types (e.g. `SubAttribute` and `AttributeType`) do depend on each other recursively / cyclically.

## Decision

Put types (`@dataclass(frozen=True)`) into the `_types` module and have all function modules depend on the `_types` module to define their inputs and outputs.

## Consequences

Separating types into a `_types` module (e.g. `tc.Project` is an alias for `tc._types.project.Project`) and functions into namespaced modules (e.g. `tc.project` is a module containing project-specific utilities) allows all of our tooling to run successfully.

Also, splitting up types and functions means that we can author a function like `tc.dataset.attributes` in the `tc.dataset` module while still having the `tc.attribute` module depend on `tc.Dataset` type.

Finally, for the rare cases where cyclical dependencies for types are unavoidable, we can use `typing.TYPE_CHECKING` since `mypy` and Python are smart enough to resolve these cyclical correctly via [forward references](#).

## How to write tests

Our test suite uses `pytest`.

See the [pytest docs](#) for:

- how to run specific tests
- how to capture `print` output for debugging tests
- etc...

Note that you will need to pass any `pytest` arguments after `--` so that `nox` passes the arguments correctly to `pytest`:

```
prn -s test-3.6 -- -s tests/tamr_client/test_project.py::test_from_resource_id_
↪mastering
```

### Unit tests

Each unit test:

- must be in a Python file whose name starts with `test_`
- must be a function whose name starts with `test_`
- should test *one* specific feature.
- should use `tests.tamr_client.fake` utility to fake resources and Tamr server responses as necessary

For example, testing a simple feature that does not require communication with a Tamr server could look like:

```
# test_my_feature.py
import tamr_client as tc
from tests.tamr_client import fake

def test_my_feature_works():
    # prerequisites
    p = fake.project()
    d = fake.dataset()

    # test my feature
    result = tc.my_feature(p, d)
    assert result.is_correct()
```

After using the `fake` utilities to set up your prerequisites, the rest of the test code should be as representative of real user code as possible.

Test code that exercises the feature should not contain any test-specific logic.

### Faking responses

If the tested feature requires communication with a Tamr server, you will need to fake Tamr server responses.

In general, any feature that takes a session argument will need faked responses.

You can fake responses via the `@fake.json` decorator:

```
# test_my_feature.py
import tamr_client as tc
from tests.tamr_client import fake

@fake.json
def test_my_feature():
    # prerequisites
    s = fake.session()
    p = fake.project()

    # test my feature
    result = tc.my_feature(s, p)
    assert result.is_correct()
```

`@fake.json` will look for a corresponding fake JSON file within `tests/tamr_client/fake_json`, specifically `tests/tamr_client/fake_json/<name of test file>/<name of test function>`.

In the example, that would be `tests/tamr_client/fake_json/test_my_feature/test_my_feature_works.json`.

The fake JSON file should be formatted as a list of request/response pairs in order of execution.

For a real examples, see existing fake JSON files within `tests/tamr_client/fake_json`.

## Contributing pull requests

### RFCs

If the proposed changes require design input, [open a Request For Comment 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.

### Pull requests

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

1. Fork it: [https://github.com/\[your-github-username\]/tamr-client/fork](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

### Commits

Split and squash commits as necessary to create a clean `git` history. Once you ask for review, only add new commits (do not change existing commits) for reviewer convenience. You may change commits in your PR only if reviewers are ok with it.

Commit messages **must** follow the [Conventional Commits](#). CI for pull requests will enforce this and fail if commit messages are not formatted correctly.

We recommend the [Commitizen CLI](#) to make writing Conventional Commits easy, but you may write commit messages manually or use any other tools.

Also, your commit messages should [explain any things that are not obvious](#) from reading your code!

## CI checks

Continuous integration (CI) checks are run automatically for all pull requests. CI runs the same [dev tasks](#) that you can run locally.

You should run dev tasks locally *before* submitting your PR to cut down on subsequent commits to fix the CI checks.

### 4.1.4 Release process

Releases are automated by [semantic-release](#).

### 4.1.5 Maintainers

Maintainer responsibilities:

- Triage issues
- Review + merge pull requests
- Discuss RFCs
- Publish new releases

Current maintainers:

Want to become a maintainer? Open a pull request that adds your name to the list of current maintainers!

## 5.1 BETA

**WARNING:** Do not rely on BETA features in production workflows. Support from Tamr may be limited.

### 5.1.1 Tutorials

#### Tutorial: Get Tamr version

This tutorial will cover basic Python client usage by guiding you through:

1. Configuring the connection to a Tamr instance
2. Retrieving the version of that instance

#### Prerequisites

To complete this tutorial you will need:

- `tamr-unify-client` *installed*
- access to a Tamr instance, specifically:
  - a username and password that allow you to log in to Tamr
  - the socket address of the instance

The socket address is composed of

1. The protocol, such as "https" or "http"
2. The host, which may be "localhost" if the instance is deployed from the same machine from which your Python code will be run
3. The port at which you access the Tamr user interface, typically 9100

When you view the Tamr user interface in a browser, the url is `<protocol>://<host>:<port>`. If the port is missing, the URL is simply `<protocol>://host`.

### Steps

#### The Session

The Tamr Python client uses a `Session` to persist the user's authentication details across requests made to the server where Tamr is hosted.

A `Session` carries authentication credentials derived from a username and password, and is not explicitly tied to any single Tamr instance. For more details, see the documentation for the [Requests library](#).

- Use your username and password to create an instance of `tamr_client.UsernamePasswordAuth`.
- Use the function `tamr_client.session.from_auth` to create a `Session`.

```
from getpass import getpass

import tamr_client as tc

username = input("Tamr Username:")
password = getpass("Tamr Password:")

auth = tc.UsernamePasswordAuth(username, password)
session = tc.session.from_auth(auth)
```

#### The Instance

An `Instance` models the installation or instance of Tamr with which a user interacts via the Python client.

- Create an `Instance` using the protocol, host, and port of your Tamr instance.

```
protocol = "http"
host = "localhost"
port = 9100

instance = tc.Instance(protocol=protocol, host=host, port=port)
```

#### Getting the version of Tamr

With the `Session` and `Instance` defined, you can now interact with the API of the Tamr instance. One simple example is fetching the version of the Tamr software running on the server.

- Use the function `tc.instance.version` and print the returned value.

```
print(tc.instance.version(session, instance))
```

All of the above steps can be combined into the following script `get_tamr_version.py`:

```
from getpass import getpass

import tamr_client as tc

username = input("Tamr Username:")
password = getpass("Tamr Password:")

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

(continues on next page)

(continued from previous page)

```
session = tc.session.from_auth(auth)

protocol = "http"
host = "localhost"
port = 9100

instance = tc.Instance(protocol=protocol, host=host, port=port)

print(tc.instance.version(session, instance))
```

To run the script via command line:

```
TAMR_CLIENT_BETA=1 python get_tamr_version.py
```

If successful, the printed result should be similar to `v2020.016.0`.

Congratulations! This is just the start of what can be done with the Tamr Python client.

To continue learning, see other tutorials and examples.

## Tutorial: Continuous Mastering

This tutorial will cover using the Python client to keep a Mastering project up-to-date. This includes carrying new data through to the end of the project and using any new labels to update the machine-learning model.

While this is intended to propagate changes such as pair labeling that may be applied in the Tamr user interface, at no point during this tutorial is it necessary to interact with the user interface in any way.

## Prerequisites

To complete this tutorial you will need:

- `tamr-unify-client` *installed*
- access to a Tamr instance, specifically:
  - a username and password that allow you to log in to Tamr
  - the socket address of the instance
- an existing Mastering project in the following state
  - the schema mapping between the attributes of the source datasets and the unified dataset has been defined
  - the blocking model has been defined
  - labels have been applied to pairs

It is recommended that you first complete the tutorial [here](#). Alternatively, a different Mastering project can be used as long as the above conditions are met.

### Steps

#### 1. Configure the Session and Instance

- Use your username and password to create an instance of `tamr_client.UsernamePasswordAuth`.
- Use the function `tamr_client.session.from_auth` to create a `Session`.

```
from getpass import getpass

import tamr_client as tc

username = input("Tamr Username:")
password = getpass("Tamr Password:")

auth = tc.UsernamePasswordAuth(username, password)
session = tc.session.from_auth(auth)
```

- Create an Instance using the `protocol`, `host`, and `port` of your Tamr instance. Replace these with the corresponding values for your Tamr instance.

```
protocol = "http"
host = "localhost"
port = 9100

instance = tc.Instance(protocol=protocol, host=host, port=port)
```

#### 2. Get the Tamr Mastering project to be updated

Use the function `tc.project.by_name` to retrieve the project information from the server by its name.

```
project = tc.project.by_name(session, instance, "MasteringTutorial")
```

Ensure that the retrieved project is a Mastering project by checking its type:

```
if not isinstance(project, tc.MasteringProject):
    raise RuntimeError(f"{project.name} is not a mastering project.")
```

#### 3. Update the unified dataset

To update the unified dataset, use the function `tc.mastering.update_unified_dataset`. This function:

- Applies the [attribute mapping configuration](#)
- Applies any transformations
- Updates the unified dataset with updated source data

```
operation_1 = tc.mastering.update_unified_dataset(session, project)
tc.operation.check(session, operation_1)
```

This function and all others in this tutorial are *synchronous*, meaning that they will not return until the job in Tamr has resolved, either successfully or unsuccessfully. The function `tc.operation.check` will raise an exception and halt the script if the job started in Tamr fails for any reason.

#### 4. Generate pairs

To generate pairs according to the `configured pair filter rules`, use the function `tc.mastering.generate_pairs`.

```
operation_2 = tc.mastering.generate_pairs(session, project)
tc.operation.check(session, operation_2)
```

#### 5. Train the model with new Labels

Running all of the functions in this section and in the “Apply the model” section that follows is equivalent to initiating “Apply feedback and update results” in the Tamr user interface.

To `update the machine-learning model` with newly-applied labels use the function `tc.mastering.apply_feedback`.

```
operation_3 = tc.mastering.apply_feedback(session, project)
tc.operation.check(session, operation_3)
```

#### 6. Apply the model

Running all of the functions in the previous “Train the model with new labels” section and in this section is equivalent to initiating “Apply feedback and update results” in the Tamr user interface.

Running the functions in this section alone is equivalent to initiating “Update results only” in the Tamr user interface.

Applying the trained machine-learning model requires three functions.

- To update the pair prediction results, use the function `tc.mastering.update_pair_results`.

```
operation_4 = tc.mastering.update_pair_results(session, project)
tc.operation.check(session, operation_4)
```

- To update the list of `high-impact pairs`, use the function `tc.mastering.update_high_impact_pairs`.

```
operation_5 = tc.mastering.update_high_impact_pairs(session, project)
tc.operation.check(session, operation_5)
```

- To update the clustering results, use the function `tc.mastering.update_cluster_results`.

```
operation_6 = tc.mastering.update_cluster_results(session, project)
tc.operation.check(session, operation_6)
```

#### 7. Publish the clusters

To publish the record clusters, use the function `tc.mastering.publish_clusters`.

```
operation_7 = tc.mastering.publish_clusters(session, project)
tc.operation.check(session, operation_7)
```

All of the above steps can be combined into the following script `continuous_mastering.py`:

```
from getpass import getpass

import tamr_client as tc

username = input("Tamr Username:")
password = getpass("Tamr Password:")

auth = tc.UsernamePasswordAuth(username, password)
session = tc.session.from_auth(auth)

protocol = "http"
host = "localhost"
port = 9100

instance = tc.Instance(protocol=protocol, host=host, port=port)

project = tc.project.by_name(session, instance, "MasteringTutorial")

if not isinstance(project, tc.MasteringProject):
    raise RuntimeError(f"{project.name} is not a mastering project.")

operation_1 = tc.mastering.update_unified_dataset(session, project)
tc.operation.check(session, operation_1)

operation_2 = tc.mastering.generate_pairs(session, project)
tc.operation.check(session, operation_2)

operation_3 = tc.mastering.apply_feedback(session, project)
tc.operation.check(session, operation_3)

operation_4 = tc.mastering.update_pair_results(session, project)
tc.operation.check(session, operation_4)

operation_5 = tc.mastering.update_high_impact_pairs(session, project)
tc.operation.check(session, operation_5)

operation_6 = tc.mastering.update_cluster_results(session, project)
tc.operation.check(session, operation_6)

operation_7 = tc.mastering.publish_clusters(session, project)
tc.operation.check(session, operation_7)
```

To run the script via command line:

```
TAMR_CLIENT_BETA=1 python continuous_mastering.py
```

To continue learning, see other tutorials and examples.

## 5.1.2 Reference

### Attribute

### Attribute

**class** `tamr_client.Attribute` (*url, name, type, is\_nullable, description=None*)  
A Tamr Attribute.

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

#### Parameters

- **url** (URL) –
- **name** (*str*) –
- **type** (`Union[PrimitiveType, Array, Map, Record]`) –
- **is\_nullable** (*bool*) –
- **description** (`Optional[str]`) –

`tamr_client.attribute.by_resource_id` (*session, dataset, id*)  
Get attribute by resource ID

Fetches attribute from Tamr server

#### Parameters

- **dataset** (*Dataset*) – Dataset containing this attribute
- **id** (*str*) – Attribute ID

#### Raises

- **`attribute.NotFound`** – If no attribute could be found at the specified URL. Corresponds to a 404 HTTP error.
- **`requests.HTTPError`** – If any other HTTP error is encountered.

**Return type** `Attribute`

`tamr_client.attribute.to_json` (*attr*)  
Serialize attribute into JSON

**Parameters** **attr** (*Attribute*) – Attribute to serialize

**Return type** `Dict[str, Any]`

**Returns** JSON data representing the attribute

`tamr_client.attribute.create` (*session, dataset, \*, name, is\_nullable, type=Array(inner\_type=<PrimitiveType.STRING: 5>), description=None*)

Create an attribute

Posts a creation request to the Tamr server

#### Parameters

- **dataset** (*Dataset*) – Dataset that should contain the new attribute
- **name** (*str*) – Name for the new attribute
- **type** (`Union[PrimitiveType, Array, Map, Record]`) – Attribute type for the new attribute

- **is\_nullable** (`bool`) – Determines if the new attribute can contain NULL values
- **description** (`Optional[str]`) – Description of the new attribute
- **force** – If `True`, skips reserved attribute name check

**Return type** `Attribute`

**Returns** The newly created attribute

**Raises**

- **`attribute.ReservedName`** – If attribute name is reserved.
- **`attribute.AlreadyExists`** – If an attribute already exists at the specified URL. Corresponds to a 409 HTTP error.
- **`requests.HTTPError`** – If any other HTTP error is encountered.

`tamr_client.attribute.update` (`session`, `attribute`, \*, `description=None`)

Update an existing attribute

PUTS an update request to the Tamr server

**Parameters**

- **`attribute`** (`Attribute`) – Existing attribute to update
- **`description`** (`Optional[str]`) – Updated description for the existing attribute

**Return type** `Attribute`

**Returns** The newly updated attribute

**Raises**

- **`attribute.NotFound`** – If no attribute could be found at the specified URL. Corresponds to a 404 HTTP error.
- **`requests.HTTPError`** – If any other HTTP error is encountered.

`tamr_client.attribute.delete` (`session`, `attribute`)

Deletes an existing attribute

Sends a deletion request to the Tamr server

**Parameters** **`attribute`** (`Attribute`) – Existing attribute to delete

**Raises**

- **`attribute.NotFound`** – If no attribute could be found at the specified URL. Corresponds to a 404 HTTP error.
- **`requests.HTTPError`** – If any other HTTP error is encountered.

## Exceptions

**class** `tamr_client.attribute.AlreadyExists`

Raised when trying to create an attribute that already exists on the server

**class** `tamr_client.attribute.NotFound`

Raised when referencing (e.g. updating or deleting) an attribute that does not exist on the server.

**class** `tamr_client.attribute.ReservedName`

Raised when attempting to create an attribute with a reserved name

## AttributeType

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

`tamr_client.attribute.type.BOOLEAN = <PrimitiveType.BOOLEAN: 1>`  
 An enumeration.

`tamr_client.attribute.type.DOUBLE = <PrimitiveType.DOUBLE: 2>`  
 An enumeration.

`tamr_client.attribute.type.INT = <PrimitiveType.INT: 3>`  
 An enumeration.

`tamr_client.attribute.type.LONG = <PrimitiveType.LONG: 4>`  
 An enumeration.

`tamr_client.attribute.type.STRING = <PrimitiveType.STRING: 5>`  
 An enumeration.

`tamr_client.attribute.type.DEFAULT = Array(inner_type=<PrimitiveType.STRING: 5>)`  
[//docs.tamr.com/reference#attribute-types](https://docs.tamr.com/reference#attribute-types)

---

**Note:** *sphinx\_autodoc\_typehints* cannot handle forward reference to *AttributeType*, so reference docs are written manually for this type

---

**Parameters** `inner_type` –

**Type** See <https://docs.tamr.com/reference#attribute-types>

`tamr_client.attribute.type.GEOSPATIAL = Record(attributes=(SubAttribute(name='point', type=`  
[//docs.tamr.com/reference#attribute-types](https://docs.tamr.com/reference#attribute-types)

**Parameters** `attributes` –

**Type** See <https://docs.tamr.com/reference#attribute-types>

**class** `tamr_client.attribute.type.Array` (*inner\_type*)

**Parameters** `inner_type` (*AttributeType*) –

**class** `tamr_client.attribute.type.Map` (*inner\_type*)

**Parameters** `inner_type` (*AttributeType*) –

**class** `tamr_client.attribute.type.Record` (*attributes*)

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

**Parameters** `attributes` (*Tuple[SubAttribute,...]*) –

`tamr_client.attribute.type.from_json` (*data*)

Make an attribute type from JSON data (deserialize)

**Parameters** `data` (*Dict[str, Any]*) – JSON data from Tamr server

**Return type** *Union[PrimitiveType, Array, Map, Record]*

`tamr_client.attribute.type.to_json` (*attr\_type*)

Serialize attribute type to JSON

**Parameters** `attr_type` (*Union[PrimitiveType, Array, Map, Record]*) – Attribute type to serialize

**Return type** *Dict[str, Any]*

## SubAttribute

**class** `tamr_client.SubAttribute` (*name*, *type*, *is\_nullable*, *description=None*)

### Parameters

- **name** (`str`) –
- **type** (`AttributeType`) –
- **is\_nullable** (`bool`) –
- **description** (Optional [`str`]) –

`tamr_client.attribute.sub.from_json` (*data*)  
Make a SubAttribute from JSON data (deserialize)

**Parameters** **data** (`Dict[str, Any]`) – JSON data received from Tamr server.

**Return type** `SubAttribute`

`tamr_client.attribute.sub.to_json` (*subattr*)  
Serialize subattribute into JSON

**Parameters** **subattr** (`SubAttribute`) – SubAttribute to serialize

**Return type** `Dict[str, Any]`

## Auth

**class** `tamr_client.UsernamePasswordAuth` (*username*, *password*)

Provides username/password authentication for Tamr.

Sets the *Authorization* HTTP header with Tamr’s custom *BasicCreds* format.

### Parameters

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

## Example

```
>>> import tamr_client as tc
>>> auth = tc.UsernamePasswordAuth('my username', 'my password')
>>> s = tc.Session(auth)
```

## Backup

**class** `tamr_client.Backup` (*url*, *path*, *state*, *error\_message*)

A Tamr backup

See <https://docs.tamr.com/new/docs/configuration-backup-and-restore>

### Parameters

- **url** (`URL`) –
- **path** (`str`) –
- **state** (`str`) –

- **error\_message** (*str*) –

`tamr_client.backup.get_all` (*session, instance*)

Get all backups that have been initiated for a Tamr instance.

**Parameters**

- **session** (*Session*) – Tamr session
- **instance** (*Instance*) – Tamr instance

**Return type** `List[Backup]`

**Returns** A list of Tamr backups

**Raises** `backup.NotFound` – If no backup found at the specified URL

`tamr_client.backup.by_resource_id` (*session, instance, resource\_id*)

Get information on a specific Tamr backup.

**Parameters**

- **session** (*Session*) – Tamr session
- **instance** (*Instance*) – Tamr instance
- **resource\_id** (*str*) – Resource ID of the backup

**Return type** `Backup`

**Returns** A Tamr backup

**Raises** `backup.NotFound` – If no backup found at the specified URL

`tamr_client.backup.initiate` (*session, instance*)

Initiate a Tamr backup.

**Parameters**

- **session** (*Session*) – Tamr session
- **instance** (*Instance*) – Tamr instance

**Return type** `Backup`

**Returns** Initiated backup

**Raises** `backup.InvalidOperation` – If attempting an invalid operation

`tamr_client.backup.cancel` (*session, backup*)

Cancel a Tamr backup.

**Parameters**

- **session** (*Session*) – Tamr session
- **backup** (*Backup*) – A Tamr backup

**Return type** `Backup`

**Returns** Canceled backup

**Raises**

- `backup.NotFound` – If no backup found at the specified URL
- `backup.InvalidOperation` – If attempting an invalid operation

`tamr_client.backup.poll` (*session*, *backup*)

Poll this backup for server-side updates.

Does not update the Backup object. Instead, returns a new Backup.

### Parameters

- **session** (`Session`) – Tamr session
- **backup** (`Backup`) – Tamr backup to be polled

**Return type** `Backup`

**Returns** A Tamr backup

**Raises** `backup.NotFound` – If no backup found at the specified URL

## Exceptions

**class** `tamr_client.backup.NotFound`

Raised when referencing a backup that does not exist on the server.

**class** `tamr_client.backup.InvalidOperation`

Raised when attempting an invalid operation.

## Categorization

### Categorization

`tamr_client.categorization.update_unified_dataset` (*session*, *project*)

Apply changes to the unified dataset and wait for the operation to complete

**Parameters** **project** (`CategorizationProject`) – Tamr Categorization project

**Return type** `Operation`

`tamr_client.categorization.apply_feedback` (*session*, *project*)

Train the categorization model according to verified labels and wait for the operation to complete

**Parameters** **project** (`CategorizationProject`) – Tamr Categorization project

**Return type** `Operation`

`tamr_client.categorization.update_results` (*session*, *project*)

Generate classifications based on the latest categorization model and wait for the operation to complete

**Parameters** **project** (`CategorizationProject`) – Tamr Categorization project

**Return type** `Operation`

`tamr_client.categorization.manual_labels` (*session*, *project*)

Get manual labels from a Categorization project.

**Parameters** **project** (`CategorizationProject`) – Tamr project containing labels

**Return type** `Dataset`

**Returns** Dataset containing manual labels

**Raises**

- `dataset.NotFound` – If no dataset could be found at the specified URL

- `dataset.Ambiguous` – If multiple targets match dataset name

## Categorization Project

**class** `tamr_client.CategorizationProject` (*url, name, description=None*)

A Tamr Categorization project

See <https://docs.tamr.com/reference/the-project-object>

### Parameters

- **url** (URL) –
- **name** (str) –
- **description** (Optional[str]) –

`tamr_client.categorization.project.create` (*session, instance, name, description=None, external\_id=None, unified\_dataset\_name=None*)

Create a Categorization project in Tamr.

### Parameters

- **instance** (Instance) – Tamr instance
- **name** (str) – Project name
- **description** (Optional[str]) – Project description
- **external\_id** (Optional[str]) – External ID of the project
- **unified\_dataset\_name** (Optional[str]) – Unified dataset name. If None, will be set to project name + `'_unified_dataset'`

**Return type** `Union[CategorizationProject, MasteringProject, SchemaMappingProject, GoldenRecordsProject]`

**Returns** Project created in Tamr

### Raises

- **project.AlreadyExists** – If a project with these specifications already exists
- **requests.HTTPError** – If any other HTTP error is encountered

## Dataset

### Dataset

**class** `tamr_client.Dataset` (*url, name, key\_attribute\_names, description=None*)

A Tamr dataset

See <https://docs.tamr.com/reference/dataset-models>

### Parameters

- **url** (URL) – The canonical dataset-based URL for this dataset e.g. `/datasets/1`
- **name** (str) –
- **key\_attribute\_names** (Tuple[str, ...]) –
- **description** (Optional[str]) –

`tamr_client.dataset.by_resource_id(session, instance, id)`

Get dataset by resource ID

Fetches dataset from Tamr server

### Parameters

- **instance** (`Instance`) – Tamr instance containing this dataset
- **id** (`str`) – Dataset ID

### Raises

- **`dataset.NotFound`** – If no dataset could be found at the specified URL. Corresponds to a 404 HTTP error.
- **`requests.HTTPError`** – If any other HTTP error is encountered.

**Return type** `Dataset`

`tamr_client.dataset.by_name(session, instance, name)`

Get dataset by name

Fetches dataset from Tamr server

### Parameters

- **instance** (`Instance`) – Tamr instance containing this dataset
- **name** (`str`) – Dataset name

### Raises

- **`dataset.NotFound`** – If no dataset could be found with that name.
- **`dataset.Ambiguous`** – If multiple targets match dataset name.
- **`requests.HTTPError`** – If any other HTTP error is encountered.

**Return type** `Dataset`

`tamr_client.dataset.attributes(session, dataset)`

Get all attributes from a dataset

**Parameters** **dataset** (`Dataset`) – Dataset containing the desired attributes

**Return type** `Tuple[Attribute, ...]`

**Returns** The attributes for the specified dataset

**Raises** **`requests.HTTPError`** – If an HTTP error is encountered.

`tamr_client.dataset.materialize(session, dataset)`

Materialize a dataset and wait for the operation to complete. Materializing consists of updating the dataset (including records) in persistent storage (HBase) based on upstream changes to data.

**Parameters** **dataset** (`Dataset`) – A Tamr dataset which will be materialized

**Return type** `Operation`

`tamr_client.dataset.delete(session, dataset, *, cascade=False)`

Deletes an existing dataset

Sends a deletion request to the Tamr server

### Parameters

- **dataset** (`Dataset`) – Existing dataset to delete

- **cascade** (`bool`) – Whether to delete all derived datasets as well

**Raises**

- **dataset.NotFound** – If no dataset could be found at the specified URL. Corresponds to a 404 HTTP error.
- **requests.HTTPError** – If any other HTTP error is encountered.

`tamr_client.dataset.get_all` (*session, instance, \*, filter=None*)

Get all datasets from an instance

**Parameters**

- **instance** (`Instance`) – Tamr instance from which to get datasets
- **filter** (`Union[str, List[str], None]`) – Filter expression, e.g. “externalId==wobbly” Multiple expressions can be passed as a list

**Return type** `Tuple[Dataset,...]`

**Returns** The datasets retrieved from the instance

**Raises** **requests.HTTPError** – If an HTTP error is encountered.

`tamr_client.dataset.create` (*session, instance, \*, name, key\_attribute\_names, description=None, external\_id=None*)

Create a dataset in Tamr.

**Parameters**

- **instance** (`Instance`) – Tamr instance
- **name** (`str`) – Dataset name
- **key\_attribute\_names** (`Tuple[str,...]`) – Dataset primary key attribute names
- **description** (`Optional[str]`) – Dataset description
- **external\_id** (`Optional[str]`) – External ID of the dataset

**Return type** `Dataset`

**Returns** Dataset created in Tamr

**Raises**

- **dataset.AlreadyExists** – If a dataset with these specifications already exists.
- **requests.HTTPError** – If any other HTTP error is encountered.

**Exceptions**

**class** `tamr_client.dataset.NotFound`

Raised when referencing (e.g. updating or deleting) a dataset that does not exist on the server.

**class** `tamr_client.dataset.Ambiguous`

Raised when referencing a dataset by name that matches multiple possible targets.

**class** `tamr_client.dataset.AlreadyExists`

Raised when a dataset with these specifications already exists.

## Record

See <https://docs.tamr.com/reference/record> “The recommended approach for modifying records is to use the `upsert()` and `delete()` functions for all use cases they can handle. For more advanced use cases, the underlying `_update()` function can be used directly.”

`record.upsert(session, dataset, records, *, primary_key_name=None)`

Create or update the specified records.

### Parameters

- **dataset** (`Dataset`) – Dataset to receive record updates
- **records** (`Iterable[Dict]`) – The records to update, as dictionaries
- **primary\_key\_name** (`Optional[str]`) – The primary key for these records, which must be a key in each record dictionary. By default the `key_attribute_name` of dataset

**Return type** `Dict[str, Any]`

**Returns** JSON response body from server

### Raises

- `requests.HTTPError` – If an HTTP error is encountered
- `primary_key.NotFound` – If `primary_key_name` does not match dataset primary key
- `primary_key.NotFound` – If `primary_key_name` not in a record dictionary

`record.delete(session, dataset, records, *, primary_key_name=None)`

Deletes the specified records, based on primary key values. Does not check that other attribute values match.

### Parameters

- **dataset** (`Dataset`) – Dataset from which to delete records
- **records** (`Iterable[Dict]`) – The records to update, as dictionaries
- **primary\_key\_name** (`Optional[str]`) – The primary key for these records, which must be a key in each record dictionary. By default the `key_attribute_name` of dataset

**Return type** `Dict[str, Any]`

**Returns** JSON response body from server

### Raises

- `requests.HTTPError` – If an HTTP error is encountered
- `primary_key.NotFound` – If `primary_key_name` does not match dataset primary key
- `primary_key.NotFound` – If `primary_key_name` not in a record dictionary

`record._update(session, dataset, updates)`

Send a batch of record creations/updates/deletions to this dataset. You probably want to use `upsert()` or `delete()` instead.

### Parameters

- **dataset** (`Dataset`) – Dataset containing records to be updated
- **updates** (`Iterable[Dict]`) – Each update should be formatted as specified in the [Public Docs for Dataset updates](#).

**Return type** `Dict[str, Any]`

**Returns** JSON response body from server

**Raises** `requests.HTTPError` – If an HTTP error is encountered

`record.stream(session, dataset)`

Stream the records in this dataset as Python dictionaries.

**Parameters** `dataset` (`Union[Dataset, UnifiedDataset]`) – Dataset from which to stream records

**Return type** `Iterator[Dict[str, Any]]`

**Returns** Python generator yielding records

`record.delete_all(session, dataset)`

Delete all records in this dataset

**Parameters** `dataset` (`Union[Dataset, UnifiedDataset]`) – Dataset from which to delete records

## Dataframe

`dataframe.upsert(session, dataset, df, *, primary_key_name=None)`

Upserts a record for each row of `df` with attributes for each column in `df`.

### Parameters

- **dataset** (`Dataset`) – Dataset to receive record updates
- **df** (`pd.DataFrame`) – The DataFrame containing records to be upserted
- **primary\_key\_name** (`Optional[str]`) – The primary key of the dataset. Must be a column of `df`. By default the `key_attribute_name` of dataset

**Return type** `Dict[str, Any]`

**Returns** JSON response body from the server

### Raises

- `requests.HTTPError` – If an HTTP error is encountered
- `primary_key.NotFound` – If `primary_key_name` is not a column in `df` or the index of `df`
- `ValueError` – If `primary_key_name` matches both a column in `df` and the index of `df`

## Unified

`class tamr_client.dataset.unified.UnifiedDataset(url, name, key_attribute_names, description=None)`

A Tamr unified dataset

See <https://docs.tamr.com/reference/dataset-models>

### Parameters

- **url** (`URL`) – The project-based alias for this dataset e.g. `/projects/1/unifiedDataset`
- **name** (`str`) –
- **key\_attribute\_names** (`Tuple[str, ...]`) –
- **description** (`Optional[str]`) –

`tamr_client.dataset.unified.from_project` (*session*, *project*)

Get unified dataset of a project

Fetches the unified dataset of a given project from Tamr server

**Parameters** `project` (`Union[CategorizationProject, MasteringProject, SchemaMappingProject, GoldenRecordsProject]`) – Tamr project of this Unified Dataset

**Raises**

- `unified.NotFound` – If no unified dataset could be found at the specified URL. Corresponds to a 404 HTTP error.
- `requests.HTTPError` – If any other HTTP error is encountered.

**Return type** `UnifiedDataset`

`tamr_client.dataset.unified.apply_changes` (*session*, *unified\_dataset*)

Applies changes to the unified dataset and waits for the operation to complete

**Parameters** `unified_dataset` (`UnifiedDataset`) – The Unified Dataset which will be committed

**Return type** `Operation`

### Exceptions

`class` `tamr_client.dataset.unified.NotFound`

Raised when referencing (e.g. updating or deleting) a unified dataset that does not exist on the server.

### Golden Records

#### Golden Records

`tamr_client.golden_records.update` (*session*, *project*)

Update the draft golden records and wait for the operation to complete

**Parameters** `project` (`GoldenRecordsProject`) – Tamr Golden Records project

**Return type** `Operation`

`tamr_client.golden_records.publish` (*session*, *project*)

Publish the golden records and wait for the operation to complete

**Parameters** `project` (`GoldenRecordsProject`) – Tamr Golden Records project

**Return type** `Operation`

## Golden Records Project

**class** `tamr_client.GoldenRecordsProject` (*url, name, description=None*)

A Tamr Golden Records project

See <https://docs.tamr.com/reference/the-project-object>

### Parameters

- **url** (URL) –
- **name** (str) –
- **description** (Optional[str]) –

## Instance

**class** `tamr_client.Instance` (*protocol='http', host='localhost', port=None*)

Connection parameters for a running Tamr instance

### Parameters

- **protocol** (str) –
- **host** (str) –
- **port** (Optional[int]) –

`tamr_client.instance.origin` (*instance*)

HTTP origin i.e. <protocol>://<host>[:<port>].

For additional information, see [MDN web docs](#) .

**Return type** str

`tamr_client.instance.version` (*session, instance*)

Return the Tamr version for an instance.

### Parameters

- **session** (Session) – Tamr Session
- **instance** (Instance) – Tamr instance

Returns: Version

**Return type** str

## Mastering

### Mastering

`tamr_client.mastering.update_unified_dataset` (*session, project*)

Apply changes to the unified dataset and wait for the operation to complete

**Parameters** **project** (MasteringProject) – Tamr Mastering project

**Return type** Operation

`tamr_client.mastering.estimate_pairs` (*session, project*)

Update the estimated pair counts and wait for the operation to complete

**Parameters** **project** (MasteringProject) – Tamr Mastering project

**Return type** Operation

`tamr_client.mastering.generate_pairs` (*session*, *project*)

Generate pairs according to the binning model and wait for the operation to complete

**Parameters** `project` (MasteringProject) – Tamr Mastering project

**Return type** Operation

`tamr_client.mastering.apply_feedback` (*session*, *project*)

Train the pair-matching model according to verified labels and wait for the operation to complete

**Parameters** `project` (MasteringProject) – Tamr Mastering project

**Return type** Operation

`tamr_client.mastering.update_pair_results` (*session*, *project*)

Update record pair predictions according to the latest pair-matching model and wait for the operation to complete

**Parameters** `project` (MasteringProject) – Tamr Mastering project

**Return type** Operation

`tamr_client.mastering.update_high_impact_pairs` (*session*, *project*)

Produce new high-impact pairs according to the latest pair-matching model and wait for the operation to complete

**Parameters** `project` (MasteringProject) – Tamr Mastering project

**Return type** Operation

`tamr_client.mastering.update_cluster_results` (*session*, *project*)

Generate clusters based on the latest pair-matching model and wait for the operation to complete

**Parameters** `project` (MasteringProject) – Tamr Mastering project

**Return type** Operation

`tamr_client.mastering.publish_clusters` (*session*, *project*)

Publish current record clusters and wait for the operation to complete

**Parameters** `project` (MasteringProject) – Tamr Mastering project

**Return type** Operation

## Mastering Project

**class** `tamr_client.MasteringProject` (*url*, *name*, *description=None*)

A Tamr Mastering project

See <https://docs.tamr.com/reference/the-project-object>

**Parameters**

- `url` (URL) –
- `name` (str) –
- `description` (Optional[str]) –

`tamr_client.mastering.project.create` (*session*, *instance*, *name*, *description=None*, *external\_id=None*, *unified\_dataset\_name=None*)

Create a Mastering project in Tamr.

**Parameters**

- **instance** (Instance) – Tamr instance
- **name** (str) – Project name
- **description** (Optional[str]) – Project description
- **external\_id** (Optional[str]) – External ID of the project
- **unified\_dataset\_name** (Optional[str]) – Unified dataset name. If None, will be set to project name + `'_unified_dataset'`

**Return type** Union[`CategorizationProject`, `MasteringProject`, `SchemaMappingProject`, `GoldenRecordsProject`]

**Returns** Project created in Tamr

**Raises**

- **project.AlreadyExists** – If a project with these specifications already exists.
- **requests.HTTPError** – If any other HTTP error is encountered.

## Operation

**class** `tamr_client.Operation` (*url*, *type*, *status=None*, *description=None*)  
A Tamr operation

See <https://docs.tamr.com/new/reference/the-operation-object>

**Parameters**

- **url** (URL) –
- **type** (str) –
- **status** (Optional[Dict[str, str]]) –
- **description** (Optional[str]) –

`tamr_client.operation.check` (*session*, *operation*)

Waits for the operation to finish and raises an exception if the operation was not successful.

**Parameters** *operation* (Operation) – Operation to be checked.

**Raises** *Failed* – If the operation failed.

`tamr_client.operation.poll` (*session*, *operation*)

Poll this operation for server-side updates.

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

**Parameters** *operation* (Operation) – Operation to be polled.

**Return type** Operation

`tamr_client.operation.wait` (*session*, *operation*, \*, *poll\_interval\_seconds=3*, *time-out\_seconds=None*)

Continuously polls for this operation's server-side state.

**Parameters**

- **operation** (Operation) – Operation to be polled.
- **poll\_interval\_seconds** (int) – Time interval (in seconds) between subsequent polls.

- **timeout\_seconds** (Optional[int]) – Time (in seconds) to wait for operation to resolve.

**Raises** **TimeoutError** – If operation takes longer than *timeout\_seconds* to resolve.

**Return type** Operation

`tamr_client.operation.succeeded(operation)`

Convenience method for checking if operation was successful.

**Return type** bool

`tamr_client.operation.by_resource_id(session, instance, resource_id)`

Get operation by ID

**Parameters** **resource\_id** (str) – The ID of the operation

**Return type** Operation

## Exceptions

**class** `tamr_client.operation.Failed`

Raised when checking a failed operation.

**class** `tamr_client.operation.NotFound`

Raised when referencing an operation that does not exist on the server.

## Primary Key

### Exceptions

**class** `tamr_client.primary_key.Ambiguous`

Raised when referencing a primary key by name that matches multiple possible targets.

**class** `tamr_client.primary_key.NotFound`

Raised when referencing a primary key by name that does not exist.

## Project

`tamr_client.project.by_resource_id(session, instance, id)`

Get project by resource ID. Fetches project from Tamr server.

### Parameters

- **instance** (Instance) – Tamr instance containing this dataset
- **id** (str) – Project ID

### Raises

- **project.NotFound** – If no project could be found at the specified URL. Corresponds to a 404 HTTP error.
- **requests.HTTPError** – If any other HTTP error is encountered.

**Return type** Union[`CategorizationProject`, `MasteringProject`, `SchemaMappingProject`, `GoldenRecordsProject`]

`tamr_client.project.by_name(session, instance, name)`

Get project by name Fetches project from Tamr server.

### Parameters

- **instance** (Instance) – Tamr instance containing this project
- **name** (str) – Project name

### Raises

- **project.NotFound** – If no project could be found with that name.
- **project.Ambiguous** – If multiple targets match project name.
- **requests.HTTPError** – If any other HTTP error is encountered.

**Return type** Union[CategorizationProject, MasteringProject, SchemaMappingProject, GoldenRecordsProject]

`tamr_client.project.get_all(session, instance, *, filter=None)`

Get all projects from an instance

### Parameters

- **instance** (Instance) – Tamr instance from which to get projects
- **filter** (Union[str, List[str], None]) – Filter expression, e.g. “externalId==wobbly” Multiple expressions can be passed as a list

**Return type** Tuple[Union[CategorizationProject, MasteringProject, SchemaMappingProject, GoldenRecordsProject], ...]

**Returns** The projects retrieved from the instance

**Raises** **requests.HTTPError** – If an HTTP error is encountered.

## Exceptions

**class** `tamr_client.project.NotFound`

Raised when referencing (e.g. updating or deleting) a project that does not exist on the server.

**class** `tamr_client.project.Ambiguous`

Raised when referencing a project by name that matches multiple possible targets.

## Restore

**class** `tamr_client.Restore(url, backup_path, state, error_message)`

A Tamr restore

See <https://docs.tamr.com/new/docs/configuration-backup-and-restore>

### Parameters

- **url** (URL) –
- **backup\_path** (str) –
- **state** (str) –
- **error\_message** (str) –

`tamr_client.restore.get(session, instance)`

Get information on the latest Tamr restore, if any.

### Parameters

- **session** (*Session*) – Tamr session
- **instance** (*Instance*) – Tamr instance

**Return type** *Restore*

**Returns** Latest Tamr restore

**Raises** *restore.NotFound* – If no backup found at the specified URL

`tamr_client.restore.initiate(session, instance, backup_path)`

Initiate a Tamr restore.

**Parameters**

- **session** (*Session*) – Tamr session
- **instance** (*Instance*) – Tamr instance
- **backup\_path** (*str*) – Path to the backup

**Return type** *Restore*

**Returns** Initiated restore

**Raises** *restore.InvalidOperation* – If attempting an invalid operation

`tamr_client.restore.cancel(session, restore)`

Cancel a Tamr restore.

**Parameters**

- **session** (*Session*) – Tamr session
- **restore** (*Restore*) – A Tamr restore

**Return type** *Restore*

**Returns** Canceled restore

**Raises**

- *restore.NotFound* – If no backup file found at the specified path
- *restore.InvalidOperation* – If attempting an invalid operation

## Exceptions

**class** `tamr_client.restore.NotFound`

Raised when referencing a restore that does not exist on the server.

**class** `tamr_client.restore.InvalidOperation`

Raised when attempting an invalid operation.

## Schema Mapping

### Schema Mapping

`tamr_client.schema_mapping.update_unified_dataset` (*session, project*)

Apply changes to the unified dataset and wait for the operation to complete

**Parameters** `project` (SchemaMappingProject) – Tamr Schema Mapping project

**Return type** Operation

### Schema Mapping Project

**class** `tamr_client.SchemaMappingProject` (*url, name, description=None*)

A Tamr Schema Mapping project

See <https://docs.tamr.com/reference/the-project-object>

#### Parameters

- **url** (URL) –
- **name** (str) –
- **description** (Optional[str]) –

`tamr_client.schema_mapping.project.create` (*session, instance, name, description=None, external\_id=None, unified\_dataset\_name=None*)

Create a Schema Mapping project in Tamr.

#### Parameters

- **instance** (Instance) – Tamr instance
- **name** (str) – Project name
- **description** (Optional[str]) – Project description
- **external\_id** (Optional[str]) – External ID of the project
- **unified\_dataset\_name** (Optional[str]) – Unified dataset name. If None, will be set to project name + `'_unified_dataset'`

**Return type** Union[`CategorizationProject`, `MasteringProject`, `SchemaMappingProject`, `GoldenRecordsProject`]

**Returns** Project created in Tamr

#### Raises

- **project.AlreadyExists** – If a project with these specifications already exists.
- **requests.HTTPError** – If any other HTTP error is encountered.

## Transformations

`tamr_client.transformations.get_all` (*session*, *project*)

Get the transformations of a Project

**Parameters** `project` (Union[CategorizationProject, MasteringProject, SchemaMappingProject, GoldenRecordsProject]) – Project containing transformations

**Raises** `requests.HTTPError` – If any HTTP error is encountered.

### Example

```
>>> import tamr_client as tc
>>> session = tc.session.from_auth('username', 'password')
>>> instance = tc.instance.Instance(host="localhost", port=9100)
>>> project1 = tc.project.by_resource_id(session, instance, id='1')
>>> print(tc.transformations.get_all(session, project1))
```

**Return type** Transformations

`tamr_client.transformations.replace_all` (*session*, *project*, *tx*)

Replaces the transformations of a Project

### Parameters

- **project** (Union[CategorizationProject, MasteringProject, SchemaMappingProject, GoldenRecordsProject]) – Project to place transformations within
- **tx** (Transformations) – Transformations to put into project

**Raises** `requests.HTTPError` – If any HTTP error is encountered.

### Example

```
>>> import tamr_client as tc
>>> session = tc.session.from_auth('username', 'password')
>>> instance = tc.instance.Instance(host="localhost", port=9100)
>>> project1 = tc.project.by_resource_id(session, instance, id='1')
>>> dataset3 = tc.dataset.by_resource_id(session, instance, id='3')
>>> new_input_tx = tc.InputTransformation("SELECT *, upper(name) as name;", ↵
↵[dataset3])
>>> all_tx = tc.Transformations(
... input_scope=[new_input_tx],
... unified_scope=["SELECT *, 1 as one;"]
... )
>>> tc.transformations.replace_all(session, project1, all_tx)
```

**Return type** Response

## Response

Utilities for working with `requests.Response`.

`tamr_client.response.successful(response)`

Ensure response does not contain an HTTP error.

Delegates to `requests.Response.raise_for_status()`

**Return type** `Response`

**Returns** The response being checked.

**Raises** `requests.exceptions.HTTPError` – If an HTTP error is encountered.

`tamr_client.response.ndjson(response, **kwargs)`

Stream newline-delimited JSON from the response body

Analog to `requests.Response.json()` but for `.ndjson`-formatted body.

**Recommended:** For memory efficiency, use `stream=True` when sending the request corresponding to this response.

### Parameters

- **response** (`Response`) – Response whose body should be streamed as newline-delimited JSON.
- **\*\*kwargs** – Keyword arguments passed to underlying `requests.Response.iter_lines()` call.

**Returns** Each line of the response body, parsed as JSON

## Example

```
>>> import tamr_client as tc
>>> s = tc.session.from_auth(...)
>>> r = s.get(..., stream=True)
>>> for data in tc.response.ndjson(r):
...     assert data['my key'] == 'my_value'
```

**Return type** `Iterator[Dict[str, Any]]`

## Session

The `Session` type is an alias for `requests.Session`.

For more information, see the official `requests.Session` docs.

`tamr_client.session.from_auth(auth)`

Create a new authenticated session

**Parameters** `auth` (`HTTPBasicAuth`) – Authentication

**Return type** `Session`



## PYTHON MODULE INDEX

t

`tamr_client.record`, 104



## INDEX

### A

- add\_input\_dataset ()  
     (*tamr\_unify\_client.categorization.project.CategorizationProject*  
     *method*), 29
- add\_input\_dataset ()  
     (*tamr\_unify\_client.mastering.project.MasteringProject*  
     *method*), 51
- add\_input\_dataset ()  
     (*tamr\_unify\_client.project.resource.Project*  
     *method*), 66
- AlreadyExists (*class in tamr\_client.attribute*), 96
- AlreadyExists (*class in tamr\_client.dataset*), 103
- Ambiguous (*class in tamr\_client.dataset*), 103
- Ambiguous (*class in tamr\_client.primary\_key*), 110
- Ambiguous (*class in tamr\_client.project*), 111
- apply\_changes () (in *module*  
     *tamr\_client.dataset.unified*), 106
- apply\_feedback () (in *module*  
     *tamr\_client.categorization*), 100
- apply\_feedback () (in *module*  
     *tamr\_client.mastering*), 108
- apply\_options () (*tamr\_unify\_client.operation.Operation*  
     *method*), 56
- as\_categorization ()  
     (*tamr\_unify\_client.categorization.project.CategorizationProject*  
     *method*), 29
- as\_categorization ()  
     (*tamr\_unify\_client.mastering.project.MasteringProject*  
     *method*), 51
- as\_categorization ()  
     (*tamr\_unify\_client.project.resource.Project*  
     *method*), 66
- as\_mastering () (*tamr\_unify\_client.categorization.project.CategorizationProject*  
     *method*), 29
- as\_mastering () (*tamr\_unify\_client.mastering.project.MasteringProject*  
     *method*), 51
- as\_mastering () (*tamr\_unify\_client.project.resource.Project*  
     *method*), 66
- Attribute (*class in tamr\_client*), 95
- Attribute (*class in*  
     *tamr\_unify\_client.attribute.resource*), 23
- attribute\_configurations ()  
     (*tamr\_unify\_client.categorization.project.CategorizationProject*  
     *method*), 29
- attribute\_configurations ()  
     (*tamr\_unify\_client.mastering.project.MasteringProject*  
     *method*), 51
- attribute\_configurations ()  
     (*tamr\_unify\_client.project.resource.Project*  
     *method*), 67
- attribute\_mappings ()  
     (*tamr\_unify\_client.categorization.project.CategorizationProject*  
     *method*), 29
- attribute\_mappings ()  
     (*tamr\_unify\_client.mastering.project.MasteringProject*  
     *method*), 52
- attribute\_mappings ()  
     (*tamr\_unify\_client.project.resource.Project*  
     *method*), 67
- attribute\_name () (*tamr\_unify\_client.project.attribute\_configuration.*  
     *property*), 59
- attribute\_profiles ()  
     (*tamr\_unify\_client.dataset.profile.DatasetProfile*  
     *property*), 44
- attribute\_role () (*tamr\_unify\_client.project.attribute\_configuration.*  
     *property*), 58
- AttributeCollection (*class in*  
     *tamr\_unify\_client.attribute.collection*), 25
- AttributeConfiguration (*class in*  
     *tamr\_unify\_client.project.attribute\_configuration.resource*),  
     58
- AttributeConfigurationCollection (*class in*  
     *tamr\_unify\_client.project.attribute\_configuration.collection*),  
     61
- AttributeConfigurationSpec (*class in*  
     *tamr\_unify\_client.project.attribute\_configuration.resource*),  
     59
- AttributeMapping (*class in*  
     *tamr\_unify\_client.project.attribute\_mapping.resource*),  
     62
- AttributeMappingCollection (*class in*  
     *tamr\_unify\_client.project.attribute\_mapping.collection*),  
     65
- AttributeMappingSpec (*class in*



`cluster_id()` (*tamr\_unify\_client.mastering.published\_cluster\_spec.version*, *Property*), 36  
`cluster_id()` (*tamr\_unify\_client.mastering.published\_cluster\_spec.version*, *Property*), 36  
`create()` (*in module tamr\_client.attribute*), 95  
`create()` (*in module tamr\_client.categorization.project*), 101  
`create()` (*in module tamr\_client.dataset*), 103  
`create()` (*in module tamr\_client.mastering.project*), 108  
`create()` (*in module tamr\_client.schema\_mapping.project*), 113  
`create()` (*tamr\_unify\_client.attribute.collection.AttributeCollection* *method*), 26  
`create()` (*tamr\_unify\_client.categorization.category.collection.CategoryCollection* *method*), 33  
`create()` (*tamr\_unify\_client.dataset.collection.DatasetCollection* *method*), 42  
`create()` (*tamr\_unify\_client.project.attribute\_configuration.collection.AttributeConfigurationCollection* *method*), 62  
`create()` (*tamr\_unify\_client.project.attribute\_mapping.collection.AttributeMappingCollection* *method*), 65  
`create()` (*tamr\_unify\_client.project.collection.ProjectCollection* *method*), 70  
`create_from_dataframe()` (*tamr\_unify\_client.dataset.collection.DatasetCollection* *method*), 42  
`create_profile()` (*tamr\_unify\_client.dataset.resource.Dataset* *method*), 38  
`create_taxonomy()` (*tamr\_unify\_client.categorization.project.CategorizationProject* *method*), 29  
*CreationError* (*class in tamr\_unify\_client.dataset.collection*), 43  
**D**  
`dataframe.upsert()` (*in module tamr\_client*), 105  
*Dataset* (*class in tamr\_client*), 101  
*Dataset* (*class in tamr\_unify\_client.dataset.resource*), 36  
`dataset()` (*tamr\_unify\_client.dataset.uri.DatasetURI* *method*), 45  
`dataset()` (*tamr\_unify\_client.dataset.use.DatasetUse* *method*), 46  
`dataset_id()` (*tamr\_unify\_client.dataset.use.DatasetUse* *property*), 46  
`dataset_name()` (*tamr\_unify\_client.dataset.profile.DatasetProfile* *property*), 43  
`dataset_name()` (*tamr\_unify\_client.dataset.status.DatasetStatus* *property*), 44  
`dataset_name()` (*tamr\_unify\_client.dataset.use.DatasetUse* *property*), 46  
*DatasetCollection* (*class in tamr\_unify\_client.dataset.collection*), 41  
*DatasetProfile* (*class in tamr\_unify\_client.dataset.profile*), 43  
`delete()` (*tamr\_unify\_client.attribute.resource.Attribute* *method*), 23  
`delete()` (*tamr\_unify\_client.categorization.project.CategorizationProject* *method*), 30  
`delete()` (*tamr\_unify\_client.categorization.taxonomy.Taxonomy* *method*), 34  
`delete()` (*tamr\_unify\_client.Client* *method*), 36  
`delete()` (*tamr\_unify\_client.dataset.profile.DatasetProfile* *method*), 44  
`delete()` (*tamr\_unify\_client.dataset.resource.Dataset* *method*), 39  
`delete()` (*tamr\_unify\_client.dataset.status.DatasetStatus* *method*), 44  
`delete()` (*tamr\_unify\_client.dataset.usage.DatasetUsage* *method*), 46  
`delete()` (*tamr\_unify\_client.mastering.binning\_model.BinningModel* *method*), 48  
`delete()` (*tamr\_unify\_client.mastering.estimated\_pair\_counts.EstimatedPairCounts* *method*), 49  
`delete()` (*tamr\_unify\_client.mastering.project.MasteringProject* *method*), 52  
`delete()` (*tamr\_unify\_client.mastering.published\_cluster.configuration.PublishedClusterConfiguration* *method*), 54  
`delete()` (*tamr\_unify\_client.operation.Operation* *method*), 58  
`delete()` (*tamr\_unify\_client.project.attribute\_configuration.resource.AttributeConfiguration* *method*), 59  
`delete()` (*tamr\_unify\_client.project.attribute\_mapping.resource.AttributeMapping* *method*), 63  
`delete()` (*tamr\_unify\_client.project.resource.Project* *method*), 67  
`delete_all_records()` (*tamr\_unify\_client.dataset.resource.Dataset* *method*), 37  
`delete_by_resource_id()` (*tamr\_unify\_client.attribute.collection.AttributeCollection* *method*), 26  
`DEFAULT` (*in module tamr\_client.attribute.type*), 97  
`delete()` (*in module tamr\_client.dataset*), 102  
`delete()` (*tamr\_unify\_client.attribute.resource.Attribute* *method*), 23  
`delete()` (*tamr\_unify\_client.categorization.project.CategorizationProject* *method*), 30  
`delete()` (*tamr\_unify\_client.categorization.taxonomy.Taxonomy* *method*), 34  
`delete()` (*tamr\_unify\_client.Client* *method*), 36  
`delete()` (*tamr\_unify\_client.dataset.profile.DatasetProfile* *method*), 44  
`delete()` (*tamr\_unify\_client.dataset.resource.Dataset* *method*), 39  
`delete()` (*tamr\_unify\_client.dataset.status.DatasetStatus* *method*), 44  
`delete()` (*tamr\_unify\_client.dataset.usage.DatasetUsage* *method*), 46  
`delete()` (*tamr\_unify\_client.mastering.binning\_model.BinningModel* *method*), 48  
`delete()` (*tamr\_unify\_client.mastering.estimated\_pair\_counts.EstimatedPairCounts* *method*), 49  
`delete()` (*tamr\_unify\_client.mastering.project.MasteringProject* *method*), 52  
`delete()` (*tamr\_unify\_client.mastering.published\_cluster.configuration.PublishedClusterConfiguration* *method*), 54  
`delete()` (*tamr\_unify\_client.operation.Operation* *method*), 58  
`delete()` (*tamr\_unify\_client.project.attribute\_configuration.resource.AttributeConfiguration* *method*), 59  
`delete()` (*tamr\_unify\_client.project.attribute\_mapping.resource.AttributeMapping* *method*), 63  
`delete()` (*tamr\_unify\_client.project.resource.Project* *method*), 67  
`delete_all_records()` (*tamr\_unify\_client.dataset.resource.Dataset* *method*), 37  
`delete_by_resource_id()` (*tamr\_unify\_client.attribute.collection.AttributeCollection* *method*), 26

*method*), 26  
 delete\_by\_resource\_id() (tamr\_unify\_client.categorization.category.collection.CategoryCollection *method*), 34  
 delete\_by\_resource\_id() (tamr\_unify\_client.dataset.collection.DatasetCollection *method*), 42  
 delete\_by\_resource\_id() (tamr\_unify\_client.project.attribute\_configuration.collection.AttributeConfigurationCollection *method*), 62  
 delete\_by\_resource\_id() (tamr\_unify\_client.project.attribute\_mapping.collection.AttributeMappingCollection *method*), 65  
 delete\_by\_resource\_id() (tamr\_unify\_client.project.collection.ProjectCollection *method*), 70  
 delete\_records() (tamr\_unify\_client.dataset.resource.Dataset *method*), 37  
 delete\_records\_by\_id() (tamr\_unify\_client.dataset.resource.Dataset *method*), 37  
 dependencies() (tamr\_unify\_client.dataset.usage.DatasetUsage *property*), 46  
 description() (tamr\_unify\_client.attribute.resource.Attribute *property*), 23  
 description() (tamr\_unify\_client.categorization.category.resource.Category *property*), 31  
 description() (tamr\_unify\_client.categorization.project.CategorizationProject *property*), 30  
 description() (tamr\_unify\_client.dataset.resource.Dataset *property*), 36  
 description() (tamr\_unify\_client.mastering.project.MasteringProject *property*), 52  
 description() (tamr\_unify\_client.operation.Operation *property*), 57  
 description() (tamr\_unify\_client.project.resource.Project *property*), 66  
 DOUBLE (in module tamr\_client.attribute.type), 97  
**E**  
 enabled\_for\_ml() (tamr\_unify\_client.project.attribute\_configuration.resource.AttributeConfiguration *property*), 58  
 entity\_id() (tamr\_unify\_client.mastering.published\_cluster\_record.RecordPublishedCluster *property*), 55  
 estimate\_pairs() (in module tamr\_client.mastering), 107  
 estimate\_pairs() (tamr\_unify\_client.mastering.project.MasteringProject *method*), 51  
 EstimatedPairCounts (class in tamr\_unify\_client.mastering.estimated\_pair\_counts), 48  
 external\_id() (tamr\_unify\_client.categorization.project.CategorizationProject *property*), 30  
 external\_id() (tamr\_unify\_client.dataset.resource.Dataset *property*), 36  
 external\_id() (tamr\_unify\_client.mastering.project.MasteringProject *property*), 52  
 external\_id() (tamr\_unify\_client.project.resource.Project *property*), 66  
**F**  
 from\_auth() (in module tamr\_client.session), 115  
 from\_data() (tamr\_unify\_client.attribute.resource.AttributeSpec *method*), 32  
 from\_data() (tamr\_unify\_client.categorization.category.resource.Category *method*), 40  
 from\_data() (tamr\_unify\_client.dataset.resource.DatasetSpec *method*), 59  
 from\_data() (tamr\_unify\_client.project.attribute\_configuration.resource.AttributeConfigurationSpec *method*), 68  
 from\_data() (tamr\_unify\_client.project.resource.ProjectSpec *method*), 38  
 from\_geo\_features() (in module tamr\_client.attribute.sub), 98  
 from\_json() (in module tamr\_client.attribute.type), 98  
 from\_json() (tamr\_unify\_client.attribute.subattribute.SubAttribute *method*), 28  
 from\_project() (in module tamr\_client.dataset.unified), 105  
 from\_resource\_id() (tamr\_unify\_client.operation.Operation *class method*), 56  
 from\_response() (tamr\_unify\_client.operation.Operation *class method*), 56  
**G**  
 generate\_pairs() (in module tamr\_client.mastering), 108  
 GEOSPATIAL (in module tamr\_client.attribute.type), 97  
 get() (tamr\_unify\_client.Client *method*), 35  
 get\_all() (in module tamr\_client.dataset), 103  
 get\_all() (in module tamr\_client.project), 111  
 get\_all() (in module tamr\_client.transformations), 114  
 GoldenRecordsProject (class in tamr\_client), 107  
 high\_impact\_pairs() (tamr\_unify\_client.mastering.project.MasteringProject *method*), 50

I

id() (*tamr\_unify\_client.mastering.published\_cluster.resource.PublishedCluster* property), 53  
 id() (*tamr\_unify\_client.project.attribute\_configuration.resource.AttributeConfiguration* property), 58  
 id() (*tamr\_unify\_client.project.attribute\_mapping.resource.AttributeMapping* property), 62  
 initiate() (in module *tamr\_client.backup*), 99  
 initiate() (in module *tamr\_client.restore*), 112  
 inner\_type() (*tamr\_unify\_client.attribute.type.AttributeType* property), 26  
 input\_attribute\_id() (*tamr\_unify\_client.project.attribute\_mapping.resource.AttributeMapping* property), 62  
 input\_attribute\_name() (*tamr\_unify\_client.project.attribute\_mapping.resource.AttributeMapping* property), 62  
 input\_dataset\_name() (*tamr\_unify\_client.project.attribute\_mapping.resource.AttributeMapping* property), 62  
 input\_datasets() (*tamr\_unify\_client.categorization.project.CategorizationProject* method), 30  
 input\_datasets() (*tamr\_unify\_client.mastering.project.MasteringProject* method), 52  
 input\_datasets() (*tamr\_unify\_client.project.resource.Project* method), 67  
 input\_to\_project\_steps() (*tamr\_unify\_client.dataset.use.DatasetUse* property), 46  
 Instance (class in *tamr\_client*), 107  
 INT (in module *tamr\_client.attribute.type*), 97  
 InvalidOperation (class in *tamr\_client.backup*), 100  
 InvalidOperation (class in *tamr\_client.restore*), 112  
 is\_nullable() (*tamr\_unify\_client.attribute.resource.Attribute* property), 23  
 is\_streamable() (*tamr\_unify\_client.dataset.status.DatasetStatus* property), 44  
 is\_up\_to\_date() (*tamr\_unify\_client.dataset.profile.DatasetProfile* property), 43  
 is\_up\_to\_date() (*tamr\_unify\_client.mastering.estimated\_pair\_counts.EstimatedPairCounts* property), 48  
 itergeofeatures() (*tamr\_unify\_client.dataset.resource.Dataset* method), 39

K

key\_attribute\_names() (*tamr\_unify\_client.dataset.resource.Dataset* property), 36

L

LONG (in module *tamr\_client.attribute.type*), 97

M

MasteringPublishedClusterModel (class in *tamr\_unify\_client.base\_model*), 47  
 MasteringPublishedClusterModel (class in *tamr\_client.categorization*), 100  
 MasteringProject (class in *tamr\_client*), 108  
 MasteringProject (class in *tamr\_unify\_client.mastering.project*), 49  
 materialize() (in module *tamr\_client.dataset*), 102  
 Metric (class in *tamr\_unify\_client.mastering.published\_cluster.metric*), 53  
 metrics() (*tamr\_unify\_client.mastering.published\_cluster.version.PublishedClusterVersion* property), 54  
 model() (*tamr\_unify\_client.categorization.project.CategorizationProject* method), 28  
 record (*tamr\_client.record*), 104

N

name() (*tamr\_unify\_client.attribute.resource.Attribute* property), 2  
 name() (*tamr\_unify\_client.categorization.category.resource.Category* property), 31  
 name() (*tamr\_unify\_client.categorization.project.CategorizationProject* property), 30  
 name() (*tamr\_unify\_client.categorization.taxonomy.Taxonomy* property), 34  
 name() (*tamr\_unify\_client.dataset.resource.Dataset* property), 36  
 name() (*tamr\_unify\_client.mastering.project.MasteringProject* property), 52  
 name() (*tamr\_unify\_client.mastering.published\_cluster.metric.Metric* property), 53  
 name() (*tamr\_unify\_client.mastering.published\_cluster.version.PublishedClusterVersion* property), 54  
 name() (*tamr\_unify\_client.project.resource.Project* property), 66  
 new() (in module *tamr\_client.response*), 115  
 new() (*tamr\_unify\_client.attribute.resource.AttributeSpec* static method), 24  
 new() (*tamr\_unify\_client.attribute.type.AttributeTypeSpec* static method), 24  
 new() (*tamr\_unify\_client.categorization.category.resource.CategorySpec* static method), 32  
 new() (*tamr\_unify\_client.dataset.resource.DatasetSpec* static method), 40  
 new() (*tamr\_unify\_client.project.attribute\_configuration.resource.AttributeConfiguration* static method), 59  
 new() (*tamr\_unify\_client.project.attribute\_mapping.resource.AttributeMapping* static method), 63  
 new() (*tamr\_unify\_client.project.resource.ProjectSpec* static method), 68  
 NotFound (class in *tamr\_client.attribute*), 96  
 NotFound (class in *tamr\_client.backup*), 100

NotFound (class in tamr\_client.dataset), 103  
 NotFound (class in tamr\_client.dataset.unified), 106  
 NotFound (class in tamr\_client.operation), 110  
 NotFound (class in tamr\_client.primary\_key), 110  
 NotFound (class in tamr\_client.project), 111  
 NotFound (class in tamr\_client.restore), 112  
 numeric\_field\_resolution()  
 (tamr\_unify\_client.project.attribute\_configuration.resource.AttributeConfigurationDataset.profile.DatasetProfile property), 59

## O

of () (tamr\_unify\_client.attribute.resource.AttributeSpec static method), 24  
 of () (tamr\_unify\_client.attribute.type.AttributeTypeSpec static method), 27  
 of () (tamr\_unify\_client.categorization.category.resource.CategorySpec static method), 32  
 of () (tamr\_unify\_client.dataset.resource.DatasetSpec static method), 40  
 of () (tamr\_unify\_client.project.attribute\_configuration.resource.AttributeConfigurationSpec static method), 59  
 of () (tamr\_unify\_client.project.attribute\_mapping.resource.AttributeMappingSpec static method), 63  
 of () (tamr\_unify\_client.project.resource.ProjectSpec static method), 68  
 Operation (class in tamr\_client), 109  
 Operation (class in tamr\_unify\_client.operation), 56  
 origin() (in module tamr\_client.instance), 107  
 origin() (tamr\_unify\_client.Client property), 35  
 origin\_entity\_id()  
 (tamr\_unify\_client.mastering.published\_cluster.record.RecordPublishedCluster property), 55  
 origin\_source\_id()  
 (tamr\_unify\_client.mastering.published\_cluster.record.RecordPublishedCluster property), 55  
 output\_from\_project\_steps()  
 (tamr\_unify\_client.dataset.use.DatasetUse property), 46

## P

pair\_matching\_model()  
 (tamr\_unify\_client.mastering.project.MasteringProject method), 49  
 pairs () (tamr\_unify\_client.mastering.project.MasteringProject method), 49  
 parent () (tamr\_unify\_client.categorization.category.resource.Category method), 31  
 path () (tamr\_unify\_client.categorization.category.resource.Category property), 31  
 poll () (in module tamr\_client.backup), 99  
 poll () (in module tamr\_client.operation), 109  
 poll () (tamr\_unify\_client.operation.Operation method), 57  
 post () (tamr\_unify\_client.Client method), 35

predict () (tamr\_unify\_client.base\_model.MachineLearningModel method), 47  
 profile () (tamr\_unify\_client.dataset.resource.Dataset method), 38  
 profiled\_at () (tamr\_unify\_client.dataset.profile.DatasetProfile property), 43  
 profiled\_data\_version()  
 (tamr\_unify\_client.dataset.profile.DatasetProfile property), 43  
 Project (class in tamr\_unify\_client.project.resource), 66  
 project () (tamr\_unify\_client.project.step.ProjectStep method), 71  
 project\_name () (tamr\_unify\_client.project.step.ProjectStep property), 70  
 project\_step\_id()  
 (tamr\_unify\_client.project.step.ProjectStep property), 70  
 project\_step\_name()  
 (tamr\_unify\_client.project.step.ProjectStep property), 70  
 ProjectCollection (class in tamr\_unify\_client.project.collection), 69  
 projects () (tamr\_unify\_client.Client property), 36  
 ProjectSpec (class in tamr\_unify\_client.project.resource), 68  
 ProjectStep (class in tamr\_unify\_client.project.step), 70  
 publish () (in module tamr\_client.golden\_records), 106  
 publish\_published\_cluster()  
 (in module tamr\_client.mastering), 108  
 published\_cluster\_ids()  
 (tamr\_unify\_client.mastering.project.MasteringProject method), 50  
 published\_cluster\_stats()  
 (tamr\_unify\_client.mastering.project.MasteringProject method), 50  
 published\_cluster\_versions()  
 (tamr\_unify\_client.mastering.project.MasteringProject method), 50  
 published\_clusters()  
 (tamr\_unify\_client.mastering.project.MasteringProject method), 50  
 published\_clusters\_configuration()  
 (tamr\_unify\_client.mastering.project.MasteringProject method), 50  
 published\_clusters\_with\_data()  
 (tamr\_unify\_client.mastering.project.MasteringProject method), 51  
 PublishedCluster (class in tamr\_unify\_client.mastering.published\_cluster.resource), 53  
 PublishedClustersConfiguration (class in

*tamr\_unify\_client.mastering.published\_cluster.configuration* property), 43  
 54 relative\_dataset\_id()  
 PublishedClusterVersion (class in *tamr\_unify\_client.dataset.status.DatasetStatus*  
*tamr\_unify\_client.mastering.published\_cluster.version*), property), 44  
 54 relative\_id() (*tamr\_unify\_client.attribute.resource.Attribute*  
 put () (*tamr\_unify\_client.attribute.resource.AttributeSpec* property), 23  
 method), 25 relative\_id() (*tamr\_unify\_client.base\_model.MachineLearningMode*  
 put () (*tamr\_unify\_client.Client* method), 36 property), 47  
 put () (*tamr\_unify\_client.dataset.resource.DatasetSpec* relative\_id() (*tamr\_unify\_client.categorization.category.resource.Ca*  
 method), 41 property), 31  
 put () (*tamr\_unify\_client.project.attribute\_configuration.resource.AttributeConfigurationSpec* relative\_id() (*tamr\_unify\_client.categorization.project.Categorization*  
 method), 60 property), 30  
 put () (*tamr\_unify\_client.project.resource.ProjectSpec* relative\_id() (*tamr\_unify\_client.categorization.taxonomy.Taxonomy*  
 method), 69 property), 34  
 relative\_id() (*tamr\_unify\_client.dataset.profile.DatasetProfile*  
 property), 44  
 relative\_id() (*tamr\_unify\_client.dataset.resource.Dataset*  
 property), 39  
 relative\_id() (*tamr\_unify\_client.dataset.status.DatasetStatus*  
 property), 45  
 relative\_id() (*tamr\_unify\_client.dataset.uri.DatasetURI*  
 property), 45  
 relative\_id() (*tamr\_unify\_client.dataset.usage.DatasetUsage*  
 property), 45  
 relative\_id() (*tamr\_unify\_client.mastering.binning\_model.BinningM*  
 property), 48  
 relative\_id() (*tamr\_unify\_client.mastering.estimated\_pair\_counts.EstimatedPairCounts*  
 property), 49  
 relative\_id() (*tamr\_unify\_client.mastering.published\_cluster.version.PublishedClusterVersion*  
 property), 52  
 relative\_id() (*tamr\_unify\_client.mastering.published\_cluster.configuration*  
 property), 54  
 relative\_id() (*tamr\_unify\_client.operation.Operation*  
 property), 58  
 relative\_id() (*tamr\_unify\_client.project.attribute\_configuration.resource.AttributeConfiguration*  
 property), 58  
 relative\_id() (*tamr\_unify\_client.project.attribute\_mapping.resource.AttributeMapping*  
 property), 62  
 relative\_id() (*tamr\_unify\_client.project.resource.Project*  
 property), 67  
 relative\_input\_attribute\_id()  
 records () (*tamr\_unify\_client.dataset.resource.Dataset* (*tamr\_unify\_client.project.attribute\_mapping.resource.AttributeM*  
 method), 38 property), 62  
 records () (*tamr\_unify\_client.mastering.binning\_model.BinningModel* (*tamr\_unify\_client.project.attribute\_mapping.resource.AttributeM*  
 method), 47 property), 62  
 refresh () (*tamr\_unify\_client.dataset.profile.DatasetProfile* relative\_unified\_attribute\_id()  
 method), 44 (*tamr\_unify\_client.project.attribute\_mapping.resource.AttributeM*  
 property), 63  
 refresh () (*tamr\_unify\_client.dataset.resource.Dataset* method), 37  
 refresh () (*tamr\_unify\_client.mastering.estimated\_pair\_counts.EstimatedPairCounts* (*tamr\_unify\_client.categorization.project.CategorizationProject*  
 method), 49 method), 30  
 relative\_attribute\_id() remove\_input\_dataset ()  
 (*tamr\_unify\_client.project.attribute\_configuration.resource.AttributeConfiguration* (*tamr\_unify\_client.mastering.project.MasteringProject*  
 property), 58 method), 52  
 relative\_dataset\_id() remove\_input\_dataset ()  
 (*tamr\_unify\_client.dataset.profile.DatasetProfile* (*tamr\_unify\_client.project.resource.Project*  
 method), 52)

method), 67  
 replace\_all() (in module `tamr_client.transformations`), 114  
 request() (*tamr\_unify\_client*.Client method), 35  
 ReservedName (class in *tamr\_client*.attribute), 96  
 resource\_id() (*tamr\_unify\_client*.attribute.resource.Attribute property), 24  
 resource\_id() (*tamr\_unify\_client*.base\_model.MachineLearningMethod property), 47  
 resource\_id() (*tamr\_unify\_client*.categorization.category.resource.Category property), 31  
 resource\_id() (*tamr\_unify\_client*.categorization.project.CategorizationProject property), 30  
 resource\_id() (*tamr\_unify\_client*.categorization.taxonomy.Taxonomy property), 34  
 resource\_id() (*tamr\_unify\_client*.dataset.profile.DatasetProfile property), 44  
 resource\_id() (*tamr\_unify\_client*.dataset.resource.Dataset property), 39  
 resource\_id() (*tamr\_unify\_client*.dataset.status.DatasetStatus property), 45  
 resource\_id() (*tamr\_unify\_client*.dataset.uri.DatasetURI property), 45  
 resource\_id() (*tamr\_unify\_client*.dataset.usage.DatasetUsage property), 46  
 resource\_id() (*tamr\_unify\_client*.mastering.binning\_model.BinningModel property), 48  
 resource\_id() (*tamr\_unify\_client*.mastering.estimated\_pair\_counts.EstimatedPairCounts property), 49  
 resource\_id() (*tamr\_unify\_client*.mastering.project.MasteringProject property), 52  
 resource\_id() (*tamr\_unify\_client*.mastering.published\_cluster\_configuration.PublishedClustersConfiguration property), 54  
 resource\_id() (*tamr\_unify\_client*.operation.Operation property), 58  
 resource\_id() (*tamr\_unify\_client*.project.attribute\_configuration.resource.AttributeConfiguration property), 59  
 resource\_id() (*tamr\_unify\_client*.project.attribute\_mapping.resource.AttributeMapping property), 63  
 resource\_id() (*tamr\_unify\_client*.project.resource.Project property), 67  
 Restore (class in *tamr\_client*), 111  
 spec() (*tamr\_unify\_client*.attribute.type.AttributeType method), 27  
 spec() (*tamr\_unify\_client*.categorization.category.resource.Category method), 31  
 spec() (*tamr\_unify\_client*.categorization.project.CategorizationProject method), 30  
 spec() (*tamr\_unify\_client*.dataset.resource.Dataset method), 39  
 spec() (*tamr\_unify\_client*.mastering.project.MasteringProject method), 52  
 spec() (*tamr\_unify\_client*.mastering.published\_cluster\_configuration.PublishedClustersConfiguration method), 59  
 spec() (*tamr\_unify\_client*.project.attribute\_configuration.resource.AttributeMapping method), 63  
 spec() (*tamr\_unify\_client*.project.attribute\_mapping.resource.AttributeMapping method), 63  
 spec() (*tamr\_unify\_client*.project.resource.Project method), 67  
 spec() (*tamr\_unify\_client*.project.resource.Project method), 67  
 state() (*tamr\_unify\_client*.operation.Operation property), 57  
 status() (*tamr\_unify\_client*.dataset.resource.Dataset method), 38  
 stream() (*tamr\_unify\_client*.attribute.collection.AttributeCollection method), 26  
 stream() (*tamr\_unify\_client*.categorization.category.collection.Category method), 33  
 stream() (*tamr\_unify\_client*.dataset.collection.DatasetCollection method), 46  
 stream() (*tamr\_unify\_client*.project.attribute\_configuration.collection.AttributeConfiguration method), 65  
 stream() (*tamr\_unify\_client*.project.collection.ProjectCollection method), 69  
 STRING (in module *tamr\_client*.attribute.type), 97  
 String (class in *tamr\_client*.attribute.type), 97  
 succeeded() (in module *tamr\_client*.operation), 110  
 succeeded() (*tamr\_unify\_client*.operation.Operation method), 57  
 successful() (in module *tamr\_client*.response), 115

## S

SchemaMappingProject (class in *tamr\_client*), 113  
 similarity\_function() (*tamr\_unify\_client*.project.attribute\_configuration.resource.AttributeConfiguration property), 58  
 simple\_metrics() (*tamr\_unify\_client*.dataset.profile.DatasetProfile.attribute.type.Map class), 43  
 source\_id() (*tamr\_unify\_client*.mastering.published\_cluster\_record.RecordPublishedCluster module), 55  
 spec() (*tamr\_unify\_client*.attribute.resource.Attribute method), 23

## T

tags() (*tamr\_unify\_client*.dataset.resource.Dataset property), 36  
 tamr\_client.AttributeConfiguration (built-in class), 97  
 tamr\_client.AttributeConfiguration.type.Array (built-in class), 97  
 tamr\_client.AttributeConfiguration.type.Map (built-in class), 97  
 tamr\_client.RecordPublishedCluster module, 104  
 tamr\_client.SubAttribute (built-in class), 98



version() (*tamr\_unify\_client.mastering.published\_cluster.record.version.RecordPublishedClusterVersion* property), 55

version() (*tamr\_unify\_client.mastering.published\_cluster.version.PublishedClusterVersion* property), 54

versions() (*tamr\_unify\_client.mastering.published\_cluster.record.RecordPublishedClusterVersion* property), 55

versions() (*tamr\_unify\_client.mastering.published\_cluster.resource.PublishedClusterVersion* property), 53

versions\_time\_to\_live() (*tamr\_unify\_client.mastering.published\_cluster.configuration.PublishedClusterConfiguration* property), 54

**W**

wait() (in module *tamr\_client.operation*), 109

wait() (*tamr\_unify\_client.operation.Operation* method), 57

with\_attribute\_name() (*tamr\_unify\_client.project.attribute\_configuration.resource.AttributeConfigurationSpec* method), 60

with\_attribute\_role() (*tamr\_unify\_client.project.attribute\_configuration.resource.AttributeConfigurationSpec* method), 60

with\_attributes() (*tamr\_unify\_client.attribute.type.AttributeTypeSpec* method), 27

with\_base\_type() (*tamr\_unify\_client.attribute.type.AttributeTypeSpec* method), 27

with\_description() (*tamr\_unify\_client.attribute.resource.AttributeSpec* method), 24

with\_description() (*tamr\_unify\_client.categorization.category.resource.CategorySpec* method), 32

with\_description() (*tamr\_unify\_client.dataset.resource.DatasetSpec* method), 40

with\_description() (*tamr\_unify\_client.project.resource.ProjectSpec* method), 68

with\_enabled\_for\_ml() (*tamr\_unify\_client.project.attribute\_configuration.resource.AttributeConfigurationSpec* method), 60

with\_external\_id() (*tamr\_unify\_client.dataset.resource.DatasetSpec* method), 40

with\_external\_id() (*tamr\_unify\_client.project.resource.ProjectSpec* method), 68

with\_inner\_type() (*tamr\_unify\_client.attribute.type.AttributeTypeSpec* method), 27

with\_input\_attribute\_id() (*tamr\_unify\_client.project.attribute\_mapping.resource.AttributeMappingSpec* method), 64

with\_name() (*tamr\_unify\_client.attribute.resource.AttributeSpec* method), 24

with\_name() (*tamr\_unify\_client.categorization.category.resource.CategorySpec* method), 32

with\_name() (*tamr\_unify\_client.dataset.resource.DatasetSpec* method), 40

with\_name() (*tamr\_unify\_client.project.resource.ProjectSpec* method), 68

with\_numeric\_field\_resolution() (*tamr\_unify\_client.project.attribute\_configuration.resource.AttributeConfigurationSpec* method), 60

with\_path() (*tamr\_unify\_client.categorization.category.resource.CategorySpec* method), 32

with\_relative\_input\_attribute\_id() (*tamr\_unify\_client.project.attribute\_mapping.resource.AttributeMappingSpec* method), 64

with\_relative\_unified\_attribute\_id() (*tamr\_unify\_client.project.attribute\_mapping.resource.AttributeMappingSpec* method), 64

with\_similarity\_function() (*tamr\_unify\_client.project.attribute\_configuration.resource.AttributeConfigurationSpec* method), 60

with\_tags() (*tamr\_unify\_client.dataset.resource.DatasetSpec* method), 41

with\_tokenizer() (*tamr\_unify\_client.project.attribute\_configuration.resource.AttributeConfigurationSpec* method), 60

with\_traceback() (*tamr\_unify\_client.dataset.collection.CreationError* method), 43

with\_type() (*tamr\_unify\_client.attribute.resource.AttributeSpec* method), 24

with\_type() (*tamr\_unify\_client.project.resource.ProjectSpec* method), 68

with\_unified\_attribute\_id() (*tamr\_unify\_client.project.attribute\_mapping.resource.AttributeMappingSpec* method), 64

with\_unified\_attribute\_name() (*tamr\_unify\_client.project.attribute\_mapping.resource.AttributeMappingSpec* method), 65

with\_unified\_dataset\_name() (*tamr\_unify\_client.project.attribute\_mapping.resource.AttributeMappingSpec* method), 64

*(tamr\_unify\_client.project.resource.ProjectSpec  
method), 69*