td-client-python

Release 1.7.0

Arm Treasure Data

Jan 29, 2026

CONTENTS

Treasure Data API library for Python

CHAPTER

ONE

REQUIREMENTS

`td-client` supports the following versions of Python.

- Python 3.10+
- PyPy

CHAPTER

TWO

INSTALL

You can install the releases from PyPI.

```
$ pip install td-client
```

It'd be better to install certifi to enable SSL certificate verification.

```
$ pip install certifi
```

CHAPTER

THREE

EXAMPLES

Please see also the examples at Treasure Data Documentation.

The td-client documentation is hosted at https://tdclient.readthedocs.io/, or you can go directly to the API documentation.

For information on the parameters that may be used when reading particular types of data, see File import parameters.

3.1 Listing jobs

Treasure Data API key will be read from environment variable `TD_API_KEY`, if none is given via `apikey=` argument passed to `tdclient.Client`.

Treasure Data API endpoint `https://api.treasuredata.com` is used by default. You can override this with environment variable `TD_API_SERVER`, which in turn can be overridden via `endpoint=` argument passed to `tdclient.Client`. List of available Treasure Data sites and corresponding API endpoints can be found here.

```
import tdclient

with tdclient.Client() as td:
    for job in td.jobs():
        print(job.job_id)
```

3.2 Running jobs

Running jobs on Treasure Data.

```
import tdclient

with tdclient.Client() as td:
    job = td.query("sample_datasets", "SELECT COUNT(1) FROM www_access", type="hive")
    job.wait()
    for row in job.result():
        print(repr(row))
```

3.3 Running jobs via DBAPI2

td-client-python implements PEP 0249 Python Database API v2.0. You can use td-client-python with external libraries which supports Database API such like pandas.

```
import pandas
import tdclient

def on_waiting(cursor):
    print(cursor.job_status())

with tdclient.connect(db="sample_datasets", type="presto", wait_callback=on_waiting) as␣
↪td:
    data = pandas.read_sql("SELECT symbol, COUNT(1) AS c FROM nasdaq GROUP BY symbol",␣
↪td)
    print(repr(data))
```

We offer another package for pandas named pytd with some advanced features. You may prefer it if you need to do complicated things, such like exporting result data to Treasure Data, printing job's progress during long execution, etc.

3.4 Importing data

Importing data into Treasure Data in streaming manner, as similar as fluentd is doing.

```
import sys
import tdclient

with tdclient.Client() as td:
    for file_name in sys.argv[:1]:
        td.import_file("mydb", "mytbl", "csv", file_name)
```

Warning

Importing data in streaming manner requires certain amount of time to be ready to query since schema update will be executed with delay.

3.5 Bulk import

Importing data into Treasure Data in batch manner.

```
import sys
import tdclient
import uuid
import warnings

if len(sys.argv) <= 1:
    sys.exit(0)

with tdclient.Client() as td:
    session_name = "session-{}".format(uuid.uuid1())
    bulk_import = td.create_bulk_import(session_name, "mydb", "mytbl")
    try:
        for file_name in sys.argv[1:]:
            part_name = "part-{}".format(file_name)
            bulk_import.upload_file(part_name, "json", file_name)
```

(continues on next page)

(continued from previous page)

```
        bulk_import.freeze()
    except:
        bulk_import.delete()
        raise
    bulk_import.perform(wait=True)
    if 0 < bulk_import.error_records:
        warnings.warn("detected {} error records.".format(bulk_import.error_records))
    if 0 < bulk_import.valid_records:
        print("imported {} records.".format(bulk_import.valid_records))
    else:
        raise(RuntimeError("no records have been imported: {}".format(bulk_import.name)))
    bulk_import.commit(wait=True)
    bulk_import.delete()
```

If you want to import data as msgpack format, you can write as follows:

```
import io
import time
import uuid
import warnings

import tdclient

t1 = int(time.time())
l1 = [{"a": 1, "b": 2, "time": t1}, {"a": 3, "b": 9, "time": t1}]

with tdclient.Client() as td:
    session_name = "session-{}".format(uuid.uuid1())
    bulk_import = td.create_bulk_import(session_name, "mydb", "mytbl")
    try:
        _bytes = tdclient.util.create_msgpack(l1)
        bulk_import.upload_file("part", "msgpack", io.BytesIO(_bytes))
        bulk_import.freeze()
    except:
        bulk_import.delete()
        raise
    bulk_import.perform(wait=True)
    # same as the above example
```

3.5.1 Changing how CSV and TSV columns are read

The `td-client` package will generally make sensible choices on how to read the columns in CSV and TSV data, but sometimes the user needs to override the default mechanism. This can be done using the optional file import parameters `dtypes` and `converters`.

For instance, consider CSV data that starts with the following records:

```
time,col1,col2,col3
1575454204,a,0001,a;b;c
1575454204,b,0002,d;e;f
```

If that data is read using the defaults, it will produce values that look like:

```
1575454204, "a", 1, "a;b;c"
1575454204, "b", 2, "d;e;f"
```

that is, an integer, a string, an integer and another string.

If the user wants to keep the leading zeroes in `col2`, then they can specify the column datatype as string. For instance, using `bulk_import.upload_file` to read data from `input_data`:

```
bulk_import.upload_file(
    "part", "msgpack", input_data,
    dtypes={"col2": "str"},
)
```

which would produce:

```
1575454204, "a", "0001", "a;b;c"
1575454204, "b", "0002", "d;e;f"
```

If they also wanted to treat `col3` as a sequence of strings, separated by semicolons, then they could specify a function to process `col3`:

```
bulk_import.upload_file(
    "part", "msgpack", input_data,
    dtypes={"col2": "str"},
    converters={"col3", lambda x: x.split(";")},
)
```

which would produce:

```
1575454204, "a", "0001", ["a", "b", "c"]
1575454204, "b", "0002", ["d", "e", "f"]
```

CHAPTER

FOUR

TYPE HINTS

td-client-python includes comprehensive type hints (PEP 484) for improved development experience with static type checkers like mypy and pyright. Type hints are available for all public APIs.

Features:

- Fully typed public API with precise type annotations
- `py.typed` marker file for PEP 561 compliance
- Type aliases in `tdclient.types` for common patterns
- Support for type checking with mypy, pyright, and other tools

Example with type checking:

```
import tdclient

# Type checkers will understand the types
with tdclient.Client(apikey="your_api_key") as client:
    # client is inferred as tdclient.Client
    job = client.query("sample_db", "SELECT COUNT(1) FROM table", type="presto")
    # job is inferred as tdclient.models.Job
    job.wait()
    for row in job.result():
        # row is inferred as dict[str, Any]
        print(row)
```

Using type aliases:

```
from tdclient.types import QueryEngineType, Priority

def run_query(engine: QueryEngineType, priority: Priority) -> None:
    with tdclient.Client() as client:
        job = client.query("mydb", "SELECT 1", type=engine, priority=priority)
        job.wait()
```

CHAPTER

FIVE

DEVELOPMENT

5.1 Running tests

Install the project dependencies with uv (runtime and test extras) and execute pytest via `uv run`.

```
$ uv sync --extra test
$ uv run pytest tdclient/test
```

To run the coverage suite locally, use:

```
$ uv run coverage run --source=tdclient -m pytest tdclient/test
$ uv run coverage report
```

5.2 Linting and type checking

Install the development extras and invoke `ruff` and `pyright` using `uv run`.

```
$ uv sync --dev
$ uv run ruff format tdclient --diff --exit-non-zero-on-fix
$ uv run ruff check tdclient
$ uv run pyright tdclient
```

5.3 Running tests (tox)

You can run tests against all supported Python versions with `tox`. I'd recommend you to install pyenv to manage additional interpreters.

```
$ pyenv shell system
$ for version in $(cat .python-version); do [ -d "$(pyenv root)/versions/${version}" ]␣
↪|| pyenv install "${version}"; done
$ pyenv shell --unset
```

Install the development extras (which include `tox`) with `uv`.

```
$ uv sync --dev
```

Then, run `tox` via `uv`.

```
$ uv run tox
```

5.4 Release

1. Update version *x.x.x* in *pyproject.toml*.
2. Create a PR with *release-x.x.x* branch. Request and merge the PR.
3. Create and push a tag *x.x.x* on *release-x.x.x* merge commit.
4. Create a Release on GitHub will publish new version to PyPI.

5.4.1 Manual release

If you want to release manually, you can upload by twine.

```
$ python -m build
$ twine upload dist/*
```

CHAPTER

SIX

LICENSE

Apache Software License, Version 2.0

6.1 File import parameters

`str or file-like` parameters specify where to read the input data from. They can be:

- a file name.
- a file object, representing a file opened in binary mode.
- an object that acts like an instance of io.BufferedIOBase. Reading from it returns bytes.

`format` is a string specifying an input format. The following input formats are supported:

- "msgpack" - the data is MessagePack serialized
- "json" - the data is JSON serialized.
- "csv" - the data is CSV, and will be read using the Python CSV module.
- "tsv" - the data is TSV (tab separated data), and will be read using the Python CSV module with `dialect=csv.excel_tab` explicitly set.

If `.gz` is appended to the format name (for instance, `"json.gz"`) then the data is assumed to be gzip compressed, and will be uncompressed as it is read.

Both MessagePack and JSON data are composed of an array of records, where each record is a dictionary (hash or mapping) of column name to column value.

In all import formats, every record must have a column named "time".

6.1.1 JSON data

JSON data is read using the utf-8 encoding.

6.1.2 CSV data

When reading CSV data, the following parameters may also be supplied, all of which are optional:

- `dialect` specifies the CSV dialect. The default is `csv.excel`.
- `encoding` specifies the encoding that will be used to turn the binary input data into string data. The default encoding is `"utf-8"`
- `columns` is a list of strings, giving names for the CSV columns. The default is `None`, meaning that the column names will be taken from the first record in the CSV data.

- `dtypes` is a dictionary used to specify a datatype for individual columns, for instance `{"col1": "int"}`. The available datatypes are `"bool"`, `"float"`, `"int"`, `"str"` and `"guess"`, where `"guess"` means to use the function guess_csv_value.
- `converters` is a dictionary used to specify a function that will be used to parse individual columns, for instace `{"col1", int}`. The function must take a string as its single input parameter, and return a value of the required type.

If a column is named in both `dtypes` and `converters`, then the function given in `converters` will be used to parse that column.

If a column is not named in either `dtypes` or `converters`, then it will be assumed to have datatype `"guess"`, and will be parsed with guess_csv_value.

Note that errors raised when calling a function from the `converters` dictionary will not be caught. So if `converters={"col1": int}` and "col1" contains `"not-an-int"`, the resulting `ValueError` will not be caught.

To summarise, the default for reading CSV files is:

```
dialect=csv.excel, encoding="utf-8", columns=None, dtypes=None, converters=None
```

6.1.3 TSV data

When reading TSV data, the parameters that may be used are the same as for CSV, except that:

- `dialect` may not be specified, and `csv.excel_tab` will be used.

The default for reading TSV files is:

```
encoding="utf-8", columns=None, dtypes=None, converters=None
```

6.2 API Reference

6.2.1 Client

tdclient.client.Client class is a public interface for *tdclient*. It provides methods for executions for REST API.

tdclient.client

class tdclient.client.**Client**(**args: Any*, ***kwargs: Any*)

Bases: `object`

API Client for Treasure Data Service

add_apikey(*name: str*) → bool

Parameters
name (*str*) – name of the user

Returns
True if success

add_user(*name: str*, *org: str*, *email: str*, *password: str*) → bool

Add a new user

Parameters

- **name** (*str*) – name of the user
- **org** (*str*) – organization
- **email** – (str): e-mail address

- **password** (*str*) – password

Returns
True if success

bulk_import(*name: str*) → *BulkImport*

Get a bulk import session

Parameters
name (*str*) – name of a bulk import session

Returns
tdclient.models.BulkImport

bulk_import_delete_part(*name: str*, *part_name: str*) → bool

Delete a part from a bulk import session

Parameters

- **name** (*str*) – name of a bulk import session
- **part_name** (*str*) – name of a part of the bulk import session

Returns
True if success

bulk_import_error_records(*name: str*) → Iterator[dict[str, Any]]

Parameters
name (*str*) – name of a bulk import session

Returns
an iterator of error records

bulk_import_upload_file(*name: str*, *part_name: str*, *format: Literal['msgpack', 'msgpack.gz', 'json', 'json.gz', 'csv', 'csv.gz', 'tsv', 'tsv.gz']*, *file: str | bytes | IO[bytes]*, ***kwargs: Any*) → None

Upload a part to Bulk Import session, from an existing file on filesystem.

Parameters

- **name** (*str*) – name of a bulk import session
- **part_name** (*str*) – name of a part of the bulk import session
- **format** (*str*) – format of data type (e.g. "msgpack", "json", "csv", "tsv")
- **file** (*str or file-like*) – the name of a file, or a file-like object, containing the data
- ****kwargs** – extra arguments.

There is more documentation on *format*, *file* and ***kwargs* at file import parameters.

In particular, for "csv" and "tsv" data, you can change how data columns are parsed using the `dtypes` and `converters` arguments.

- `dtypes` is a dictionary used to specify a datatype for individual columns, for instance `{"col1": "int"}`. The available datatypes are `"bool"`, `"float"`, `"int"`, `"str"` and `"guess"`. If a column is also mentioned in `converters`, then the function will be used, NOT the datatype.
- `converters` is a dictionary used to specify a function that will be used to parse individual columns, for instance `{"col1", int}`.

The default behaviour is `"guess"`, which makes a best-effort to decide the column datatype. See file import parameters for more details.

bulk_import_upload_part(*name: str*, *part_name: str*, *bytes_or_stream: bytes | bytearray | IO[bytes]*, *size: int*) → None

Upload a part to a bulk import session

Parameters

- **name** (*str*) – name of a bulk import session
- **part_name** (*str*) – name of a part of the bulk import session
- **bytes_or_stream** (*file-like*) – a file-like object contains the part
- **size** (*int*) – the size of the part

bulk_imports() → list[*BulkImport*]

List bulk import sessions

Returns
a list of *tdclient.models.BulkImport*

change_database(*db_name: str*, *table_name: str*, *new_db_name: str*) → bool

Move a target table from it's original database to new destination database.

Parameters

- **db_name** (*str*) – Target database name.
- **table_name** (*str*) – Target table name.
- **new_db_name** (*str*) – Destination database name to be moved.

Returns
True if succeeded.

Return type
bool

close() → None

Close opened API connections.

commit_bulk_import(*name: str*) → bool

Commit a bulk import session

Parameters
name (*str*) – name of a bulk import session

Returns
True if success

create_bulk_import(*name: str*, *database: str*, *table: str*, *params: BulkImportParams | None = None*) → *BulkImport*

Create new bulk import session

Parameters

- **name** (*str*) – name of new bulk import session
- **database** (*str*) – name of a database
- **table** (*str*) – name of a table

Returns
tdclient.models.BulkImport

create_database(*db_name: str*, ***kwargs: Any*) → bool

Parameters
db_name (*str*) – name of a database to create

Returns
True if success

create_log_table(*db_name: str*, *table_name: str*) → bool

Parameters

- **db_name** (*str*) – name of a database
- **table_name** (*str*) – name of a table to create

Returns
True if success

create_result(*name: str*, *url: str*, *params: ResultParams | None = None*) → bool

Create a new authentication with the specified name.

Parameters

- **name** (*str*) – Authentication name.
- **url** (*str*) – Url of the authentication to be created. e.g. "ftp://test.com/"
- **params** (*dict, optional*) – Extra parameters.

Returns
True if succeeded.

Return type
bool

create_schedule(*name: str*, *params: ScheduleParams | None = None*) → datetime | None

Create a new scheduled query with the specified name.

Parameters

- **name** (*str*) – Scheduled query name.
- **params** (*dict, optional*) – Extra parameters.
  - **type (str):**
    Query type. {"presto", "hive"}. Default: "hive"
  - **database (str):**
    Target database name.
  - **timezone (str):**
    Scheduled query's timezone. e.g. "UTC" For details, see also: https://gist.github.com/frsyuki/4533752
  - **cron (str, optional):**
    Schedule of the query. {"@daily", "@hourly", "10 * * * *" (custom cron)} See also: https://docs.treasuredata.com/articles/#!pd/Scheduling-Jobs-Using-TD-Console
  - **delay (int, optional):**
    A delay ensures all buffered events are imported before running the query. Default: 0

- **query (str):**
  Is a language used to retrieve, insert, update and modify data. See also: https://docs.treasuredata.com/articles/#!pd/SQL-Examples-of-Scheduled-Queries
- **priority (int, optional):**
  Priority of the query. Range is from -2 (very low) to 2 (very high). Default: 0
- **retry_limit (int, optional):**
  Automatic retry count. Default: 0
- **engine_version (str, optional):**
  Engine version to be used. If none is specified, the account's default engine version would be set. {"stable", "experimental"}
- **pool_name (str, optional):**
  For Presto only. Pool name to be used, if not specified, default pool would be used.
- **result (str, optional):**
  Location where to store the result of the query. e.g. 'tableau://user:password@host.com:1234/datasource'

Returns
Start date time.

Return type
`datetime.datetime`

database(*db_name: str*) → *Database*

Parameters
db_name (*str*) – name of a database

Returns
tdclient.models.Database

databases() → list[*Database*]

Returns
a list of *tdclient.models.Database*

delete_bulk_import(*name: str*) → bool

Delete a bulk import session

Parameters
name (*str*) – name of a bulk import session

Returns
True if success

delete_database(*db_name: str*) → bool

Parameters
db_name (*str*) – name of database to delete

Returns
True if success

delete_result(*name: str*) → bool

Delete the authentication having the specified name.

Parameters
name (*str*) – Authentication name.

Returns
True if succeeded.

Return type
bool

`delete_schedule`(*name: str*) → tuple[str, str]

Delete the scheduled query with the specified name.

Parameters
name (*str*) – Target scheduled query name.

Returns
Tuple of cron and query.

Return type
(str, str)

`delete_table`(*db_name: str*, *table_name: str*) → str

Delete a table

Parameters

- **db_name** (*str*) – name of a database
- **table_name** (*str*) – name of a table

Returns
a string represents the type of deleted table

`download_job_result`(*job_id: str | int*, *path: str*, *num_threads: int = 4*) → bool

Save the job result into a msgpack.gz file. :param job_id: job id :type job_id: str :param path: path to save the result :type path: str :param num_threads: number of threads to download the result.

Default: 4

Returns
True if success

`export_data`(*db_name: str*, *table_name: str*, *storage_type: str*, *params: ExportParams | None = None*) → *Job*

Export data from Treasure Data Service

Parameters

- **db_name** (*str*) – name of a database
- **table_name** (*str*) – name of a table
- **storage_type** (*str*) – type of the storage
- **params** (*dict*) – optional parameters. Assuming the following keys:
  - **access_key_id (str):**
    ID to access the information to be exported.
  - **secret_access_key (str):**
    Password for the *access_key_id*.
  - **file_prefix (str, optional):**
    Filename of exported file. Default: "<database_name>/<table_name>"

- **file_format (str, optional):**
  File format of the information to be exported. {"jsonl.gz", "tsv.gz", "json.gz"}
- **from (int, optional):**
  From Time of the data to be exported in Unix epoch format.
- **to (int, optional):**
  End Time of the data to be exported in Unix epoch format.
- assume_role (str, optional): Assume role.
- **bucket (str):**
  Name of bucket to be used.
- **domain_key (str, optional):**
  Job domain key.
- **pool_name (str, optional):**
  For Presto only. Pool name to be used, if not specified, default pool would be used.

Returns
tdclient.models.Job

freeze_bulk_import(*name: str*) → bool

Freeze a bulk import session

Parameters
name (*str*) – name of a bulk import session

Returns
True if success

history(*name: str*, *_from: int | None = None*, *to: int | None = None*) → list[*ScheduledJob*]

Get the history details of the saved query for the past 90days.

Parameters

- **name** (*str*) – Target name of the scheduled query.
- **_from** (*int, optional*) – Indicates from which nth record in the run history would be fetched. Default: 0. Note: Count starts from zero. This means that the first record in the list has a count of zero.
- **to** (*int, optional*) – Indicates up to which nth record in the run history would be fetched. Default: 20

Returns
[*tdclient.models.ScheduledJob*]

import_data(*db_name: str*, *table_name: str*, *format: Literal['msgpack', 'msgpack.gz', 'json', 'json.gz', 'csv', 'csv.gz', 'tsv', 'tsv.gz']*, *bytes_or_stream: bytes | bytearray | IO[bytes]*, *size: int*, *unique_id: str | None = None*) → float

Import data into Treasure Data Service

Parameters

- **db_name** (*str*) – name of a database
- **table_name** (*str*) – name of a table
- **format** (*str*) – format of data type (e.g. "msgpack.gz")

- **bytes_or_stream** (*str or file-like*) – a byte string or a file-like object contains the data
- **size** (*int*) – the length of the data
- **unique_id** (*str*) – a unique identifier of the data

Returns
second in float represents elapsed time to import data

import_file(*db_name: str*, *table_name: str*, *format: Literal['msgpack', 'msgpack.gz', 'json', 'json.gz', 'csv', 'csv.gz', 'tsv', 'tsv.gz']*, *file: str | bytes | IO[bytes]*, *unique_id: str | None = None*) → float

Import data into Treasure Data Service, from an existing file on filesystem.

This method will decompress/deserialize records from given file, and then convert it into format acceptable from Treasure Data Service ("msgpack.gz").

Parameters

- **db_name** (*str*) – name of a database
- **table_name** (*str*) – name of a table
- **format** (*str*) – format of data type (e.g. "msgpack", "json")
- **file** (*str or file-like*) – a name of a file, or a file-like object contains the data
- **unique_id** (*str*) – a unique identifier of the data

Returns
float represents the elapsed time to import data

job(*job_id: str | int*) → *Job*

Get a job from *job_id*

Parameters
job_id (*str*) – job id

Returns
tdclient.models.Job

job_result(*job_id: str | int*) → list[Any]

Parameters
job_id (*str*) – job id

Returns
a list of each rows in result set

job_result_each(*job_id: str | int*) → Iterator[dict[str, Any]]

Parameters
job_id (*str*) – job id

Returns
an iterator of result set

job_result_format(*job_id: str | int*, *format: Literal['msgpack', 'json', 'csv', 'tsv']*, *header: bool = False*) → list[Any]

Parameters

- **job_id** (*str*) – job id
- **format** (*str*) – output format of result set

Returns
a list of each rows in result set

job_result_format_each(*job_id: str | int*, *format: Literal['msgpack', 'json', 'csv', 'tsv']*, *header: bool = False*, *store_tmpfile: bool = False*, *num_threads: int = 4*) → Iterator[dict[str, Any]]

Parameters

- **job_id** (*str*) – job id
- **format** (*str*) – output format of result set
- **header** (*bool, optional*) – include header in the result set. Default: False
- **store_tmpfile** (*bool, optional*) – store result to a temporary file. Works only when fmt is "msgpack". Default is False.
- **num_threads** (*int, optional*) – number of threads to download result. Works only when store_tmpfile is True. Default is 4.

Returns
an iterator of rows in result set

job_status(*job_id: str | int*) → str

Parameters
job_id (*str*) – job id

Returns
a string represents the status of the job ("success", "error", "killed", "queued", "running")

jobs(*_from: int | None = None*, *to: int | None = None*, *status: str | None = None*, *conditions: dict[str, Any] | None = None*) → list[*Job*]

List jobs

Parameters

- **_from** (*int, optional*) – Gets the Job from the nth index in the list. Default: 0.
- **to** (*int, optional*) – Gets the Job up to the nth index in the list. By default, the first 20 jobs in the list are displayed
- **status** (*str, optional*) – Filter by given status. {"queued", "running", "success", "error"}
- **conditions** (*dict[str, Any], optional*) – Condition for `TIMESTAMPDIFF()` to search for slow queries. Avoid using this parameter as it can be dangerous.

Returns
a list of *tdclient.models.Job*

kill(*job_id: str | int*) → str | None

Parameters
job_id (*str*) – job id

Returns
a string represents the status of killed job ("queued", "running")

list_apikeys(*name: str*) → list[str]

Parameters
name (*str*) – name of the user

Returns
a list of string of API key

list_bulk_import_parts(*name: str*) → list[str]

List parts of a bulk import session

Parameters
name (*str*) – name of a bulk import session

Returns
a list of string represents the name of parts

perform_bulk_import(*name: str*) → *Job*

Perform a bulk import session

Parameters
name (*str*) – name of a bulk import session

Returns
tdclient.models.Job

query(*db_name: str*, *q: str*, *result_url: str | None = None*, *priority: Literal[-2, -1, 0, 1, 2, 'VERY LOW', 'LOW', 'NORMAL', 'HIGH', 'VERY HIGH'] | None = None*, *retry_limit: int | None = None*, *type: str = 'hive'*, ***kwargs: Any*) → *Job*

Run a query on specified database table.

Parameters

- **db_name** (*str*) – name of a database
- **q** (*str*) – a query string
- **result_url** (*str*) – result output URL. e.g., `postgresql://<username>:<password>@<hostname>:<port>/<database>/<table>`
- **priority** (*int or str*) – priority (e.g. "NORMAL", "HIGH", etc.)
- **retry_limit** (*int*) – retry limit
- **type** (*str*) – name of a query engine

Returns
tdclient.models.Job

Raises
ValueError – if unknown query type has been specified

remove_apikey(*name: str*, *apikey: str*) → bool

Parameters

- **name** (*str*) – name of the user
- **apikey** (*str*) – an API key to remove

Returns
True if success

remove_user(*name: str*) → bool

Remove a user

Parameters
name (*str*) – name of the user

Returns
True if success

results() → list[*Result*]

Get the list of all the available authentications.

Returns
a list of *tdclient.models.Result*

run_schedule(*name: str*, *time: int*, *num: int | None = None*) → list[*ScheduledJob*]

Execute the specified query.

Parameters

- **name** (*str*) – Target scheduled query name.
- **time** (*int*) – Time in Unix epoch format that would be set as TD_SCHEDULED_TIME
- **num** (*int*) – Indicates how many times the query will be executed. Value should be 9 or less.

Returns
[*tdclient.models.ScheduledJob*]

schedules() → list[*Schedule*]

Get the list of all the scheduled queries.

Returns
[*tdclient.models.Schedule*]

server_status() → str

Returns
a string represents current server status.

swap_table(*db_name: str*, *table_name1: str*, *table_name2: str*) → bool

Parameters

- **db_name** (*str*) – name of a database
- **table_name1** (*str*) – original table name
- **table_name2** (*str*) – table name you want to rename to

Returns
True if success

table(*db_name: str*, *table_name: str*) → *Table*

Parameters

- **db_name** (*str*) – name of a database
- **table_name** (*str*) – name of a table

Returns
tdclient.models.Table

Raises
tdclient.api.NotFoundError – if the table doesn't exist

tables(*db_name: str*) → list[*Table*]

List existing tables

Parameters
db_name (*str*) – name of a database

Returns
a list of *tdclient.models.Table*

tail(*db_name: str*, *table_name: str*, *count: int*, *to: None = None*, *_from: None = None*, *block: None = None*) → list[dict[str, Any]]

Get the contents of the table in reverse order based on the registered time (last data first).

Parameters

- **db_name** (*str*) – Target database name.
- **table_name** (*str*) – Target table name.
- **count** (*int*) – Number for record to show up from the end.
- **to** – Deprecated parameter.
- **_from** – Deprecated parameter.
- **block** – Deprecated parameter.

Returns
Contents of the table.

Return type
[dict]

unfreeze_bulk_import(*name: str*) → bool

Unfreeze a bulk import session

Parameters
name (*str*) – name of a bulk import session

Returns
True if success

update_expire(*db_name: str*, *table_name: str*, *expire_days: int*) → bool

Set expiration date to a table

Parameters

- **db_name** (*str*) – name of a database
- **table_name** (*str*) – name of a table
- **epire_days** (*int*) – expiration date in days from today

Returns
True if success

update_schedule(*name: str*, *params: ScheduleParams | None = None*) → None

Update the scheduled query.

Parameters

- **name** (*str*) – Target scheduled query name.
- **params** (*dict*) – Extra parameters.

- **type (str):**
  Query type. {"presto", "hive"}. Default: "hive"
- **database (str):**
  Target database name.
- **timezone (str):**
  Scheduled query's timezone. e.g. "UTC" For details, see also: https://gist.github.com/frsyuki/4533752
- **cron (str, optional):**
  Schedule of the query. {`"@daily"`, `"@hourly"`, `"10 * * * *"` (custom cron)} See also: https://docs.treasuredata.com/articles/#!pd/Scheduling-Jobs-Using-TD-Console
- **delay (int, optional):**
  A delay ensures all buffered events are imported before running the query. Default: 0
- **query (str):**
  Is a language used to retrieve, insert, update and modify data. See also: https://docs.treasuredata.com/articles/#!pd/SQL-Examples-of-Scheduled-Queries
- **priority (int, optional):**
  Priority of the query. Range is from -2 (very low) to 2 (very high). Default: 0
- **retry_limit (int, optional):**
  Automatic retry count. Default: 0
- **engine_version (str, optional):**
  Engine version to be used. If none is specified, the account's default engine version would be set. {"stable", "experimental"}
- **pool_name (str, optional):**
  For Presto only. Pool name to be used, if not specified, default pool would be used.
- **result (str, optional):**
  Location where to store the result of the query. e.g. 'tableau://user:password@host.com:1234/datasource'

`update_schema`(*db_name: str*, *table_name: str*, *schema: list[list[str]]*) → bool

Updates the schema of a table

Parameters

- **db_name** (*str*) – name of a database
- **table_name** (*str*) – name of a table
- **schema** (*list*) – a dictionary object represents the schema definition (will be converted to JSON) e.g.

```
[
    ["member_id", # column name
     "string", # data type
     "mem_id", # alias of the column name
    ],
    ["row_index", "long", "row_ind"],
    ...
]
```

Returns
True if success

users()
List users

Returns
a list of *tdclient.models.User*

property api: *API*
an instance of *tdclient.api.API*

property apikey: **str | None**
API key string.

tdclient.client.**job_from_dict**(*client:* Client, *dd: dict[str, Any]*, ***values: Any*) → *Job*

6.2.2 DB API

tdclient

tdclient.**Binary**(*string: bytes*) → bytes

tdclient.**DateFromTicks**(*ticks: float*) → date

tdclient.**TimeFromTicks**(*ticks: float*) → time

tdclient.**TimestampFromTicks**(*ticks: float*) → datetime

tdclient.**connect**(**args: Any*, ***kwargs: Any*) → *Connection*
Returns a DBAPI compatible connection object

Parameters
- **type** (*str*) – query engine type. “hive” by default.
- **db** (*str*) – the name of database on Treasure Data
- **result_url** (*str*) – result output URL
- **priority** (*str*) – job priority
- **retry_limit** (*int*) – job retry limit
- **wait_interval** (*int*) – job wait interval to check status
- **wait_callback** (*callable*) – a callback to be called on every ticks of job wait

Returns
tdclient.connection.Connection

tdclient.connection

class tdclient.connection.**Connection**(*type: str | None = None*, *db: str | None = None*, *result_url: str | None = None*, *priority: Literal[-2, -1, 0, 1, 2, 'VERY LOW', 'LOW', 'NORMAL', 'HIGH', 'VERY HIGH'] | None = None*, *retry_limit: int | None = None*, *wait_interval: int | None = None*, *wait_callback: Callable[[*Cursor*], None] | None = None*, ***kwargs: Any*)

Bases: object

close() → None

commit() → None

cursor() → *Cursor*

rollback() → None

property api: *API*

tdclient.cursor

class tdclient.cursor.**Cursor**(*api: API*, *wait_interval: int = 5*, *wait_callback: Callable[[Cursor], None] | None = None*, ***kwargs: Any*)

Bases: `object`

callproc(*procname: str*, **parameters: Any*) → None

close() → None

execute(*query: str*, *args: dict[str, Any] | None = None*) → str | None

executemany(*operation: str*, *seq_of_parameters: list[dict[str, Any]]*) → list[str | None]

fetchall() → list[Any]

Fetch all (remaining) rows of a query result, returning them as a sequence of sequences (e.g. a list of tuples). Note that the cursor's arraysize attribute can affect the performance of this operation.

fetchmany(*size: int | None = None*) → list[Any]

Fetch the next set of rows of a query result, returning a sequence of sequences (e.g. a list of tuples). An empty sequence is returned when no more rows are available.

fetchone() → Any | None

Fetch the next row of a query result set, returning a single sequence, or *None* when no more data is available.

job_result() → list[dict[str, Any]]

Fetch job results

Returns
Job result in `list`

job_status() → str

Show job status

Returns
The status information of the given job id at last execution.

nextset() → None

setinputsizes(*sizes: Any*) → None

setoutputsize(*size: Any*, *column: Any = None*) → None

show_job() → dict[str, Any]

Returns detailed information of a Job

Returns
Detailed information of a job

Return type
`dict`

property api: *API*

property description: list[Any]

property rowcount: int

6.2.3 Model

Some methods of *tdclient.client.Client* returns model object which represents results from REST API.

tdclient.model

class tdclient.model.**Model**(*client:* Client)

Bases: object

property client: *Client*

a *tdclient.client.Client* instance

Type

Returns

tdclient.models

tdclient.models.**BulkImport = <class 'tdclient.bulk_import_model.BulkImport'>**

Bulk-import session on Treasure Data Service

tdclient.models.**Database = <class 'tdclient.database_model.Database'>**

Database on Treasure Data Service

tdclient.models.**Schema = <class 'tdclient.job_model.Schema'>**

Schema of a database table on Treasure Data Service

tdclient.models.**Job = <class 'tdclient.job_model.Job'>**

Job on Treasure Data Service

tdclient.models.**Result = <class 'tdclient.result_model.Result'>**

Result on Treasure Data Service

tdclient.models.**ScheduledJob = <class 'tdclient.schedule_model.ScheduledJob'>**

Scheduled job on Treasure Data Service

tdclient.models.**Schedule = <class 'tdclient.schedule_model.Schedule'>**

Schedule on Treasure Data Service

tdclient.models.**Table = <class 'tdclient.table_model.Table'>**

Database table on Treasure Data Service

tdclient.models.**User = <class 'tdclient.user_model.User'>**

User on Treasure Data Service

tdclient.bulk_import_model

class tdclient.bulk_import_model.**BulkImport**(*client:* Client, ***kwargs: Any*)

Bases: *Model*

Bulk-import session on Treasure Data Service

commit(*wait: bool = False*, *wait_interval: int = 5*, *timeout: float | None = None*) → bool

Commit bulk import

delete() → bool

Delete bulk import

delete_part(*part_name: str*) → bool

Delete a part of a Bulk Import session

Parameters
part_name (*str*) – name of a part of the bulk import session

Returns
True if succeeded.

error_record_items() → Iterator[dict[str, Any]]

Fetch error record rows.

Yields
Error record

freeze() → bool

Freeze bulk import

list_parts() → list[str]

Return the list of available parts uploaded through `bulk_import_upload_part()`.

Returns
The list of bulk import part name.

Return type
[str]

perform(*wait: bool = False*, *wait_interval: int = 5*, *wait_callback: Callable[[Job], None] | None = None*, *timeout: float | None = None*) → *Job*

Perform bulk import

Parameters

- **wait** (*bool, optional*) – Flag for wait bulk import job. Default *False*
- **wait_interval** (*int, optional*) – wait interval in second. Default *5*.
- **wait_callback** (*callable, optional*) – A callable to be called on every tick of wait interval.
- **timeout** (*int, optional*) – Timeout in seconds. No timeout by default.

unfreeze() → bool

Unfreeze bulk import

update() → None

upload_file(*part_name: str*, *fmt: Literal['msgpack', 'msgpack.gz', 'json', 'json.gz', 'csv', 'csv.gz', 'tsv', 'tsv.gz']*, *file_like: str | bytes | IO[bytes]*, ***kwargs: Any*) → None

Upload a part to Bulk Import session, from an existing file on filesystem.

Parameters

- **part_name** (*str*) – name of a part of the bulk import session
- **fmt** (*str*) – format of data type (e.g. “msgpack”, “json”, “csv”, “tsv”)

- **file_like** (*str or file-like*) – the name of a file, or a file-like object, containing the data
- ****kwargs** – extra arguments.

There is more documentation on *fmt*, *file_like* and ***kwargs* at file import parameters.

In particular, for "csv" and "tsv" data, you can change how data columns are parsed using the `dtypes` and `converters` arguments.

- `dtypes` is a dictionary used to specify a datatype for individual columns, for instance `{"col1": "int"}`. The available datatypes are `"bool"`, `"float"`, `"int"`, `"str"` and `"guess"`. If a column is also mentioned in `converters`, then the function will be used, NOT the datatype.
- `converters` is a dictionary used to specify a function that will be used to parse individual columns, for instance `{"col1", int}`.

The default behaviour is `"guess"`, which makes a best-effort to decide the column datatype. See file import parameters for more details.

upload_part(*part_name: str*, *bytes_or_stream: bytes | bytearray | IO[bytes]*, *size: int*) → None

Upload a part to bulk import session

Parameters

- **part_name** (*str*) – name of a part of the bulk import session
- **bytes_or_stream** (*file-like*) – a file-like object contains the part
- **size** (*int*) – the size of the part

STATUS_COMMITTED = 'committed'

STATUS_COMMITTING = 'committing'

STATUS_PERFORMING = 'performing'

STATUS_READY = 'ready'

STATUS_UPLOADING = 'uploading'

property database: str | None

A database name in a string which the bulk import session is working on

property error_parts: int | None

The number of error parts.

property error_records: int | None

The number of error records.

property job_id: str | None

Job ID

property name: str

A name of the bulk import session

property status: str | None

The status of the bulk import session in a string

property table: str | None

A table name in a string which the bulk import session is working on

property upload_frozen: bool | None

The number of upload frozen.

property valid_parts: int | None

The number of valid parts.

property valid_records: int | None

The number of valid records.

tdclient.database_model

class tdclient.database_model.Database(*client: Client*, *db_name: str*, ***kwargs: Any*)

Bases: *Model*

Database on Treasure Data Service

create_log_table(*name: str*) → bool

Parameters
name (*str*) – name of new log table

Returns
tdclient.model.Table

delete() → bool

Delete the database

Returns
True if success

query(*q: str*, ***kwargs: Any*) → *Job*

Run a query on the database

Parameters
q (*str*) – a query string

Returns
tdclient.model.Job

table(*table_name: str*) → *Table*

Parameters
table_name (*str*) – name of a table

Returns
tdclient.model.Table

tables() → list[*Table*]

Returns
a list of tdclient.model.Table

PERMISSIONS = ['administrator', 'full_access', 'import_only', 'query_only']

PERMISSION_LIST_TABLES = ['administrator', 'full_access']

property count: int | None

Total record counts in a database.

Type
int

property created_at: datetime | None

datetime.datetime

property name: str

a name of the database

Type
str

property org_name: str | None

organization name

Type
str

property permission: str | None

permission for the database (e.g. "administrator", "full_access", etc.)

Type
str

property updated_at: datetime | None

datetime.datetime

tdclient.job_model

class tdclient.job_model.**Job**(*client:* Client, *job_id: str*, *type: str*, *query: str | None*, ***kwargs: Any*)

Bases: *Model*

Job on Treasure Data Service

error() → bool

Returns
True if the job has been finished in error

finished() → bool

Returns
True if the job has been finished in success, error or killed

kill() → str | None

Kill the job

Returns
a string represents the status of killed job ("queued", "running")

killed() → bool

Returns
True if the job has been finished in killed

queued() → bool

Returns
True if the job is queued

result() → Iterator[dict[str, Any]]

Yields
an iterator of rows in result set

result_format(*fmt: ResultFormat*, *store_tmpfile: bool = False*, *num_threads: int = 4*) → Iterator[dict[str, Any]]

Parameters

- **fmt** (*str*) – output format of result set
- **store_tmpfile** (*bool, optional*) – store result to a temporary file. Works only when fmt is "msgpack". Default is False.
- **num_threads** (*int, optional*) – number of threads to download result. Works only when store_tmpfile is True. Default is 4.

Yields
an iterator of rows in result set

running() → bool

Returns
True if the job is running

status() → str | None

Returns
a string represents the status of the job ("success", "error", "killed", "queued", "running")

Return type
str

success() → bool

Returns
True if the job has been finished in success

update() → None

Update all fields of the job

wait(*timeout: float | None = None*, *wait_interval: int = 5*, *wait_callback: Callable[[Job], None] | None = None*) → None

Sleep until the job has been finished

Parameters

- **timeout** (*int, optional*) – Timeout in seconds. No timeout by default.
- **wait_interval** (*int, optional*) – wait interval in second. Default 5 seconds.
- **wait_callback** (*callable, optional*) – A callable to be called on every tick of wait interval.

FINISHED_STATUS = ['success', 'error', 'killed']

JOB_PRIORITY = {-2: 'VERY LOW', -1: 'LOW', 0: 'NORMAL', 1: 'HIGH', 2: 'VERY HIGH'}

STATUS_BOOTING = 'booting'

STATUS_ERROR = 'error'

STATUS_KILLED = 'killed'

STATUS_QUEUED = 'queued'

STATUS_RUNNING = 'running'

STATUS_SUCCESS = 'success'

property database: str | None
a string represents the name of a database that job is running on

property debug: dict[str, Any] | None
a dict of debug output (e.g. "cmdout", "stderr")

property id: str
a string represents the identifier of the job

property job_id: str
a string represents the identifier of the job

property linked_result_export_job_id: str | None
Linked result export job ID from query job

property num_records: int | None
the number of records of job result

property org_name: str | None
organization name

property priority: str
a string represents the priority of the job (e.g. "NORMAL", "HIGH", etc.)

property query: str | None
a string represents the query string of the job

property result_export_target_job_id: str | None
Associated query job ID from result export job ID

property result_schema: list[list[str]] | None
an array of array represents the type of result columns (Hive specific) (e.g. [["_c1", "string"], ["_c2", "bigint"]])

property result_size: int | None
the length of job result

property result_url: str | None
a string of URL of the result on Treasure Data Service

property retry_limit: int | None
a number for automatic retry count

property type: str
a string represents the engine type of the job (e.g. "hive", "presto", etc.)

property url: str | None
a string of URL of the job on Treasure Data Service

property user_name: str | None
executing user name

class tdclient.job_model.**Schema**(*fields: list[Field] | None = None*)

Bases: object

Schema of a database table on Treasure Data Service

class **Field**(*name: str*, *type: str*)

Bases: object

property **name: str**

add docstring

Type
TODO

property **type: str**

add docstring

Type
TODO

add_field(*name: str*, *type: str*) → None

TODO: add docstring

property **fields: list[*Field*]**

add docstring

Type
TODO

tdclient.result_model

class tdclient.result_model.**Result**(*client: Client*, *name: str*, *url: str*, *org_name: str | None*)

Bases: *Model*

Result on Treasure Data Service

property **name: str**

a name for a authentication

Type
str

property **org_name: str | None**

organization name

Type
str

property **url: str**

a result output URL

Type
str

tdclient.schedule_model

class tdclient.schedule_model.**Schedule**(*client: Client*, **args: Any*, ***kwargs: Any*)

Bases: *Model*

Schedule on Treasure Data Service

run(*time: int*, *num: int | None = None*) → list[*ScheduledJob*]

Run a scheduled job

Parameters

- **time** (*int*) – Time in Unix epoch format that would be set as TD_SCHEDULED_TIME
- **num** (*int*) – Indicates how many times the query will be executed. Value should be 9 or less.

Returns

[*tdclient.models.ScheduledJob*]

property created_at: datetime | None

Create date

Type

datetime.datetime

property cron: str | None

The configured schedule of a scheduled job.

Returns a string represents the schedule in cron form, or *None* if the job is not scheduled to run (saved query)

property database: str | None

The target database of a scheduled job

property delay: int | None

A delay ensures all buffered events are imported before running the query.

property name: str | None

The name of a scheduled job

property next_time: datetime | None

Schedule for next run

Type

datetime.datetime

property org_name: str | None

add docstring

Type

TODO

property priority: str

The priority of a scheduled job

property query: str | None

The query string of a scheduled job

property result_url: str | None

The result output configuration in URL form of a scheduled job

property retry_limit: int | None

Automatic retry count.

property timezone: str | None

The time zone of a scheduled job

property type: str | None

Query type. {"presto", "hive"}.

property user_name: str | None

User name of a scheduled job

class tdclient.schedule_model.**ScheduledJob**(*client: Client*, *scheduled_at: datetime*, *job_id: str*, *type: str*, *query: str | None*, ***kwargs: Any*)

Bases: *Job*

Scheduled job on Treasure Data Service

property scheduled_at: datetime

a `datetime.datetime` represents the schedule of next invocation of the job

tdclient.table_model

class tdclient.table_model.**Table**(**args: Any*, ***kwargs: Any*)

Bases: *Model*

Database table on Treasure Data Service

delete() → str

a string represents the type of deleted table

export_data(*storage_type: str*, ***kwargs: Any*) → *Job*

Export data from Treasure Data Service

Parameters

- **storage_type** (*str*) – type of the storage
- ****kwargs** (*dict*) – optional parameters. Assuming the following keys:
  - **access_key_id (str):**
    ID to access the information to be exported.
  - **secret_access_key (str):**
    Password for the *access_key_id*.
  - **file_prefix (str, optional):**
    Filename of exported file. Default: "<database_name>/<table_name>"
  - **file_format (str, optional):**
    File format of the information to be exported. {"jsonl.gz", "tsv.gz", "json.gz"}
  - **from (int, optional):**
    From Time of the data to be exported in Unix epoch format.
  - **to (int, optional):**
    End Time of the data to be exported in Unix epoch format.
  - **assume_role (str, optional):**
    Assume role.
  - **bucket (str):**
    Name of bucket to be used.

- **domain_key (str, optional):**
  Job domain key.
- **pool_name (str, optional):**
  For Presto only. Pool name to be used, if not specified, default pool would be used.

Returns
tdclient.models.Job

import_data(*format: Literal['msgpack', 'msgpack.gz', 'json', 'json.gz', 'csv', 'csv.gz', 'tsv', 'tsv.gz']*, *bytes_or_stream: bytes | bytearray | IO[bytes]*, *size: int*, *unique_id: str | None = None*) → float

Import data into Treasure Data Service

Parameters

- **format** (*str*) – format of data type (e.g. "msgpack.gz")
- **bytes_or_stream** (*str or file-like*) – a byte string or a file-like object contains the data
- **size** (*int*) – the length of the data
- **unique_id** (*str*) – a unique identifier of the data

Returns
second in float represents elapsed time to import data

import_file(*format: Literal['msgpack', 'msgpack.gz', 'json', 'json.gz', 'csv', 'csv.gz', 'tsv', 'tsv.gz']*, *file: str | bytes | IO[bytes]*, *unique_id: str | None = None*) → float

Import data into Treasure Data Service, from an existing file on filesystem.

This method will decompress/deserialize records from given file, and then convert it into format acceptable from Treasure Data Service ("msgpack.gz").

Parameters

- **file** (*str or file-like*) – a name of a file, or a file-like object contains the data
- **unique_id** (*str*) – a unique identifier of the data

Returns
float represents the elapsed time to import data

tail(*count: int*, *to: int | None = None*, *_from: int | None = None*) → list[dict[str, Any]]

Parameters

- **count** (*int*) – Number for record to show up from the end.
- **to** – Deprecated parameter.
- **_from** – Deprecated parameter.

Returns
the contents of the table in reverse order based on the registered time (last data first).

property count: int | None

total number of the table

Type
int

property created_at: datetime | None

Created datetime

Type
datetime.datetime

property database_name: str

a string represents the name of the database

property db_name: str

a string represents the name of the database

property estimated_storage_size: int | None

estimated storage size

property estimated_storage_size_string: str

a string represents estimated size of the table in human-readable format

property expire_days: int | None

an int represents the days until expiration

property identifier: str

a string identifier of the table

property last_import: datetime | None

datetime.datetime

property last_log_timestamp: datetime | None

datetime.datetime

property name: str

a string represents the name of the table

property permission: str | None

permission for the database (e.g. "administrator", "full_access", etc.)

Type
str

property primary_key: str | None

add docstring

Type
TODO

property primary_key_type: str | None

add docstring

Type
TODO

property schema: list[tuple[str, str, str]] | None

str, alias:str]]: The list of a schema

Type
[[column_name

Type
str, column_type

property table_name: str

a string represents the name of the table

property type: str | None

a string represents the type of the table

property updated_at: datetime | None

Updated datetime

Type
datetime.datetime

6.2.4 API

tdclient.api.API class is an internal class represents API.

tdclient.api

class tdclient.api.**API**(*apikey: str | None = None*, *user_agent: str | None = None*, *endpoint: str | None = None*, *headers: dict[str, str] | None = None*, *retry_post_requests: bool = False*, *max_cumul_retry_delay: int = 600*, *http_proxy: str | None = None*, ***kwargs: Any*)

Bases: *BulkImportAPI*, *ConnectorAPI*, *DatabaseAPI*, *ExportAPI*, *ImportAPI*, *JobAPI*, *ResultAPI*, *ScheduleAPI*, *ServerStatusAPI*, *TableAPI*, UserAPI

Internal API class

Parameters

- **apikey** (*str*) – the API key of Treasure Data Service. If *None* is given, *TD_API_KEY* will be used if available.
- **user_agent** (*str*) – custom User-Agent.
- **endpoint** (*str*) – custom endpoint URL. If *None* is given, *TD_API_SERVER* will be used if available.
- **headers** (*dict*) – custom HTTP headers.
- **retry_post_requests** (*bool*) – Specify whether allowing API client to retry POST requests. *False* by default.
- **max_cumul_retry_delay** (*int*) – maximum retry limit in seconds. 600 seconds by default.
- **http_proxy** (*str*) – HTTP proxy setting. if *None* is given, *HTTP_PROXY* will be used if available.

build_request(*path: str | None = None*, *headers: dict[str, str] | None = None*, *endpoint: str | None = None*) → tuple[str, dict[str, str]]

checked_json(*body: bytes*, *required: list[str]*) → dict[str, Any]

close() → None

delete(*path: str*, *params: dict[str, Any] | None = None*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

get(*path: str*, *params: dict[str, Any] | None = None*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

post(*path: str*, *params: dict[str, Any] | bytes | None = None*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

put(*path: str*, *bytes_or_stream: bytes | bytearray | IO[bytes]*, *size: int*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

raise_error(*msg: str*, *res: BaseHTTPResponse*, *body: bytes | str*) → None

send_request(*method: str*, *url: str*, *fields: dict[str, Any] | None = None*, *body: bytes | bytearray | memoryview | array[int] | IO[bytes] | None = None*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → BaseHTTPResponse

DEFAULT_ENDPOINT = 'https://api.treasuredata.com/'

DEFAULT_IMPORT_ENDPOINT = 'https://api-import.treasuredata.com/'

property apikey: str | None

property endpoint: str

tdclient.bulk_import_api

class tdclient.bulk_import_api.**BulkImportAPI**

Bases: object

Enable bulk importing of data to the targeted database and table.

This class is inherited by *tdclient.api.API*.

static validate_part_name(*part_name: str*) → None

Make sure the part_name is valid

Parameters
part_name (*str*) – The part name the user is trying to use

bulk_import_delete_part(*name: str*, *part_name: str*, *params: dict[str, Any] | None = None*) → bool

Delete the imported information with the specified name.

Parameters

- **name** (*str*) – Bulk import name.
- **part_name** (*str*) – Bulk import part name.
- **params** (*dict, optional*) – Extra parameters.

Returns
True if succeeded.

bulk_import_error_records(*name: str*, *params: dict[str, Any] | None = None*) → Iterator[dict[str, Any]]

List the records that have errors under the specified bulk import name.

Parameters

- **name** (*str*) – Bulk import name.
- **params** (*dict, optional*) – Extra parameters.

Yields
Row of the data

bulk_import_upload_file(*name: str*, *part_name: str*, *format: Literal['msgpack', 'msgpack.gz', 'json', 'json.gz', 'csv', 'csv.gz', 'tsv', 'tsv.gz']*, *file: str | bytes | IO[bytes]*, ***kwargs: Any*) → None

Upload a file with bulk import having the specified name.

Parameters

- **name** (*str*) – Bulk import name.
- **part_name** (*str*) – Bulk import part name.
- **format** (*str*) – Format name. {msgpack, json, csv, tsv}
- **file** (*str or file-like*) – the name of a file, or a file-like object, containing the data
- ****kwargs** – Extra arguments.

There is more documentation on *format*, *file* and ***kwargs* at file import parameters.

In particular, for "csv" and "tsv" data, you can change how data columns are parsed using the `dtypes` and `converters` arguments.

- `dtypes` is a dictionary used to specify a datatype for individual columns, for instance `{"col1": "int"}`. The available datatypes are `"bool"`, `"float"`, `"int"`, `"str"` and `"guess"`. If a column is also mentioned in `converters`, then the function will be used, NOT the datatype.
- `converters` is a dictionary used to specify a function that will be used to parse individual columns, for instance `{"col1", int}`.

The default behaviour is `"guess"`, which makes a best-effort to decide the column datatype. See file import parameters for more details.

bulk_import_upload_part(*name: str*, *part_name: str*, *stream: bytes | bytearray | IO[bytes]*, *size: int*) → None

Upload bulk import having the specified name and part in the path.

Parameters

- **name** (*str*) – Bulk import name.
- **part_name** (*str*) – Bulk import part name.
- **stream** (*str or file-like*) – Byte string or file-like object contains the data
- **size** (*int*) – The length of the data.

checked_json(*body: bytes*, *required: list[str]*) → dict[str, Any]

commit_bulk_import(*name: str*, *params: dict[str, Any] | None = None*) → bool

Commit the bulk import information having the specified name.

Parameters

- **name** (*str*) – Bulk import name.
- **params** (*dict, optional*) – Extra parameters.

Returns

True if succeeded.

create_bulk_import(*name: str*, *db: str*, *table: str*, *params: BulkImportParams | None = None*) → bool

Enable bulk importing of data to the targeted database and table and stores it in the default resource pool. Default expiration for bulk import is 30days.

Parameters

- **name** (*str*) – Name of the bulk import.
- **db** (*str*) – Name of target database.
- **table** (*str*) – Name of target table.
- **params** (*dict, optional*) – Extra parameters.

Returns
True if succeeded

delete_bulk_import(*name: str*, *params: dict[str, Any] | None = None*) → bool

Delete the imported information with the specified name

Parameters

- **name** (*str*) – Name of bulk import.
- **params** (*dict, optional*) – Extra parameters.

Returns
True if succeeded

freeze_bulk_import(*name: str*, *params: dict[str, Any] | None = None*) → bool

Freeze the bulk import with the specified name.

Parameters

- **name** (*str*) – Bulk import name.
- **params** (*dict, optional*) – Extra parameters.

Returns
True if succeeded.

get(*path: str*, *params: dict[str, Any] | None = None*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

list_bulk_import_parts(*name: str*, *params: dict[str, Any] | None = None*) → list[str]

Return the list of available parts uploaded through *bulk_import_upload_part()*.

Parameters

- **name** (*str*) – Name of bulk import.
- **params** (*dict, optional*) – Extra parameters.

Returns
The list of bulk import part name.

Return type
[str]

list_bulk_imports(*params: dict[str, Any] | None = None*) → list[dict[str, Any]]

Return the list of available bulk imports :param params: Extra parameters. :type params: dict, optional

Returns
The list of available bulk import details.

Return type
[dict]

perform_bulk_import(*name: str*, *params: dict[str, Any] | None = None*) → str

Execute a job to perform bulk import with the indicated priority using the resource pool if indicated, else it will use the account's default.

Parameters

- **name** (*str*) – Bulk import name.
- **params** (*dict, optional*) – Extra parameters.

Returns
Job ID

Return type
str

post(*path: str*, *params: dict[str, Any] | bytes | None = None*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

put(*path: str*, *bytes_or_stream: bytes | bytearray | IO[bytes]*, *size: int*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

raise_error(*msg: str*, *res: BaseHTTPResponse*, *body: bytes | str*) → None

show_bulk_import(*name: str*) → dict[str, Any]

Show the details of the bulk import with the specified name

Parameters
name (*str*) – Name of bulk import.

Returns
Detailed information of the bulk import.

Return type
dict

unfreeze_bulk_import(*name: str*, *params: dict[str, Any] | None = None*) → bool

Unfreeze bulk_import with the specified name.

Parameters

- **name** (*str*) – Bulk import name.
- **params** (*dict, optional*) – Extra parameters.

Returns
True if succeeded.

tdclient.connector_api

class tdclient.connector_api.**ConnectorAPI**

Bases: object

Access Data Connector API which handles Data Connector.

This class is inherited by *tdclient.api.API*.

checked_json(*body: bytes*, *required: list[str]*) → dict[str, Any]

connector_create(*name: str*, *database: str*, *table: str*, *job: dict[str, Any]*, *params: dict[str, Any] | None = None*) → dict[str, Any]

Create a Data Connector session.

Parameters

- **name** (*str*) – name of the connector job
- **database** (*str*) – name of the database to perform connector job
- **table** (*str*) – name of the table to perform connector job
- **job** (*dict*) – dict representation of *load.yml*
- **params** (*dict, optional*) – Extra parameters
  - **config (str):**
    Embulk configuration as JSON format. See also https://www.embulk.org/docs/built-in.html#embulk-configuration-file-format
  - **cron (str, optional):**
    Schedule of the query. {"@daily", "@hourly", "10 * * * *" (custom cron)} See also: https://docs.treasuredata.com/articles/#!pd/Scheduling-Jobs-Using-TD-Console
  - **delay (int, optional):**
    A delay ensures all buffered events are imported before running the query. Default: 0
  - **database (str):**
    Target database for the Data Connector session
  - **name (str):**
    Name of the Data Connector session
  - **table (str):**
    Target table for the Data Connector session
  - **time_column (str, optional):**
    Column in the table for registering config.out.time
  - **timezone (str):**
    Timezone for scheduled Data Connector session. See here for list of supported timezones https://gist.github.com/frsyuki/4533752

Returns
dict

connector_delete(*name: str*) → dict[str, Any]

Delete a Data Connector session.

Parameters
name (*str*) – name of the connector job

Returns
dict

connector_guess(*job: dict[str, Any] | bytes*) → dict[str, Any]

Guess the Data Connector configuration

Parameters
job (*dict*) – dict representation of *seed.yml* See Also: https://www.embulk.org/docs/built-in.html#guess-executor

Returns
The configuration of the Data Connector.

Return type
dict

Examples

```
>>> config = {
...     "in": {
...         "type": "s3",
...         "bucket": "your-bucket",
...         "path_prefix": "logs/csv-",
...         "access_key_id": "YOUR-AWS-ACCESS-KEY",
...         "secret_access_key": "YOUR-AWS-SECRET-KEY"
...     },
...     "out": {"mode": "append"},
...     "exec": {"guess_plugins": ["json", "query_string"]},
... }
>>> td.api.connector_guess(config)
{'config': {'in': {'type': 's3',
   'bucket': 'your-bucket',
   'path_prefix': 'logs/csv-',
   'access_key_id': 'YOUR-AWS-ACCESS-KEY',
   'secret_access_key': 'YOU-AWS-SECRET-KEY',
   'parser': {'charset': 'UTF-8',
    'newline': 'LF',
    'type': 'csv',
    'delimiter': ',',
    'quote': '"',
    'escape': '"',
    'trim_if_not_quoted': False,
    'skip_header_lines': 1,
    'allow_extra_columns': False,
    'allow_optional_columns': False,
    'columns': [{'name': 'sepal.length', 'type': 'double'},
     {'name': 'sepal.width', 'type': 'double'},
     {'name': 'petal.length', 'type': 'double'},
     {'name': 'petal.width', 'type': 'string'},
     {'name': 'variety', 'type': 'string'}]}},
  'out': {'mode': 'append'},
  'exec': {'guess_plugin': ['json', 'query_string']},
  'filters': [{'rules': [{'rule': 'upper_to_lower'},
     {'pass_types': ['a-z', '0-9'],
      'pass_characters': '_',
      'replace': '_',
      'rule': 'character_types'},
     {'pass_types': ['a-z'],
      'pass_characters': '_',
      'prefix': '_',
      'rule': 'first_character_types'},
     {'rule': 'unique_number_suffix', 'max_length': 128}],
    'type': 'rename'},
   {'from_value': {'mode': 'upload_time'},
    'to_column': {'name': 'time'},
    'type': 'add_time'}]}}
```

connector_history(*name: str*) → list[dict[str, Any]]

Show the list of the executed jobs information for the Data Connector job.

Parameters
name (*str*) – name of the connector job

Returns
list

connector_issue(*db: str*, *table: str*, *job: dict[str, Any]*) → str

Create a Data Connector job.

Parameters

- **db** (*str*) – name of the database to perform connector job
- **table** (*str*) – name of the table to perform connector job
- **job** (*dict*) – dict representation of *load.yml*

Returns
job Id

Return type
str

connector_list() → list[dict[str, Any]]

Show the list of available Data Connector sessions.

Returns
list

connector_preview(*job: dict[str, Any]*) → dict[str, Any]

Show the preview of the Data Connector job.

Parameters
job (*dict*) – dict representation of *load.yml*

Returns
dict

connector_run(*name: str*, ***kwargs: Any*) → dict[str, Any]

Create a job to execute Data Connector session.

Parameters

- **name** (*str*) – name of the connector job
- ****kwargs** (*optional*) – Extra parameters.
  - **scheduled_time (int):**
    Time in Unix epoch format that would be set as *TD_SCHEDULED_TIME*.
  - **domain_key (str):**
    Job domain key which is assigned to a single job.

Returns
dict

connector_show(*name: str*) → dict[str, Any]

Show a specific Data Connector session information.

Parameters
name (*str*) – name of the connector job

Returns
dict

connector_update(*name: str*, *job: dict[str, Any]*) → dict[str, Any]

Update a specific Data Connector session.

Parameters

- **name** (*str*) – name of the connector job
- **job** (*dict*) – dict representation of *load.yml*. For detailed format, see also: https://www.embulk.org/docs/built-in.html#embulk-configuration-file-format

Returns
dict

delete(*path: str*, *params: dict[str, Any] | None = None*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

get(*path: str*, *params: dict[str, Any] | None = None*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

post(*path: str*, *params: dict[str, Any] | bytes | None = None*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

put(*path: str*, *bytes_or_stream: bytes | bytearray | IO[bytes]*, *size: int*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

raise_error(*msg: str*, *res: BaseHTTPResponse*, *body: bytes*) → None

tdclient.database_api

class tdclient.database_api.**DatabaseAPI**

Bases: object

Access to Database of Treasure Data Service.

This class is inherited by *tdclient.api.API*.

checked_json(*body: bytes*, *required: list[str]*) → dict[str, Any]

create_database(*db: str*, *params: dict[str, Any] | None = None*) → bool

Create a new database with the given name.

Parameters

- **db** (*str*) – Target database name.
- **params** (*dict*) – Extra parameters.

Returns
True if succeeded.

Return type
bool

delete_database(*db: str*) → bool

Delete a database.

Parameters
db (*str*) – Target database name.

Returns
True if succeeded.

Return type
bool

`get`(*path: str*, *params: dict[str, Any] | None = None*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

`list_databases`() → dict[str, Any]

Get the list of all the databases of the account.

Returns
Detailed database information. Each key of the dict is database name.

Return type
dict

`post`(*path: str*, *params: dict[str, Any] | bytes | None = None*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

`raise_error`(*msg: str*, *res: BaseHTTPResponse*, *body: bytes*) → None

tdclient.export_api

`class` tdclient.export_api.**ExportAPI**

Bases: `object`

Access to Export API.

This class is inherited by *tdclient.api.API*.

`checked_json`(*body: bytes*, *required: list[str]*) → dict[str, Any]

`export_data`(*db: str*, *table: str*, *storage_type: str*, *params: ExportParams | None = None*) → str

Creates a job to export the contents from the specified database and table names.

Parameters

- **db** (*str*) – Target database name.
- **table** (*str*) – Target table name.
- **storage_type** (*str*) – Name of storage type. e.g. "s3"
- **params** (*dict*) – Extra parameters. Assuming the following keys:
  - **access_key_id (str):**
    ID to access the information to be exported.
  - **secret_access_key (str):**
    Password for the *access_key_id*.
  - **file_prefix (str, optional):**
    Filename of exported file. Default: "<database_name>/<table_name>"
  - **file_format (str, optional):**
    File format of the information to be exported. {"jsonl.gz", "tsv.gz", "json.gz"}
  - **from (int, optional):**
    From Time of the data to be exported in Unix epoch format.
  - **to (int, optional):**
    End Time of the data to be exported in Unix epoch format.

- **assume_role (str, optional):**
  Assume role.
- **bucket (str):**
  Name of bucket to be used.
- **domain_key (str, optional):**
  Job domain key.
- **pool_name (str, optional):**
  For Presto only. Pool name to be used, if not specified, default pool would be used.

Returns
Job ID.

Return type
str

post(*path: str*, *params: dict[str, Any] | bytes | None = None*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

raise_error(*msg: str*, *res: BaseHTTPResponse*, *body: bytes | str*) → None

tdclient.import_api

class tdclient.import_api.**ImportAPI**

Bases: object

Import data into Treasure Data Service.

This class is inherited by *tdclient.api.API*.

checked_json(*body: bytes*, *required: list[str]*) → dict[str, Any]

import_data(*db: str*, *table: str*, *format: Literal['msgpack', 'msgpack.gz', 'json', 'json.gz', 'csv', 'csv.gz', 'tsv', 'tsv.gz']*, *bytes_or_stream: bytes | bytearray | IO[bytes]*, *size: int*, *unique_id: str | None = None*) → float

Import data into Treasure Data Service

This method expects data from a file-like object formatted with "msgpack.gz".

Parameters

- **db** (*str*) – name of a database
- **table** (*str*) – name of a table
- **format** (*str*) – format of data type (e.g. "msgpack.gz")
- **bytes_or_stream** (*str or file-like*) – a byte string or a file-like object contains the data
- **size** (*int*) – the length of the data
- **unique_id** (*str*) – a unique identifier of the data

Returns
float represents the elapsed time to import data

import_file(*db: str*, *table: str*, *format: Literal['msgpack', 'msgpack.gz', 'json', 'json.gz', 'csv', 'csv.gz', 'tsv', 'tsv.gz']*, *file: str | bytes | IO[bytes]*, *unique_id: str | None = None*, ***kwargs: Any*) → float

Import data into Treasure Data Service, from an existing file on filesystem.

This method will decompress/deserialize records from given file, and then convert it into format acceptable from Treasure Data Service ("msgpack.gz"). This method is a wrapper function to *import_data*.

Parameters

- **db** (*str*) – name of a database
- **table** (*str*) – name of a table
- **format** (*str*) – format of data type (e.g. "msgpack", "json")
- **file** (*str or file-like*) – a name of a file, or a file-like object contains the data
- **unique_id** (*str*) – a unique identifier of the data

Returns

float represents the elapsed time to import data

put(*path: str*, *bytes_or_stream: bytes | bytearray | IO[bytes]*, *size: int*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

raise_error(*msg: str*, *res: BaseHTTPResponse*, *body: bytes*) → None

tdclient.job_api

class tdclient.job_api.**JobAPI**

Bases: object

Access to Job API

This class is inherited by *tdclient.api.API*.

checked_json(*body: bytes*, *required: list[str]*) → dict[str, Any]

download_job_result(*job_id: str*, *path: str*, *num_threads: int = 4*) → bool

Download the job result to the specified path.

Parameters

- **job_id** (*int*) – Job ID
- **path** (*str*) – Path to save the job result
- **num_threads** (*int*) – Number of threads to download the job result. Default is 4.

get(*path: str*, *params: dict[str, Any] | None = None*, *headers: dict[str, str] | None = None*) → AbstractContextManager[BaseHTTPResponse]

job_result(*job_id: str*) → list[dict[str, Any]]

Return the job result.

Parameters

job_id (*int*) – Job ID

Returns

Job result in list

job_result_each(*job_id: str*) → Iterator[dict[str, Any]]

Yield a row of the job result.

Parameters
job_id (*int*) – Job ID

Yields
Row in a result

job_result_format(*job_id: str*, *format: str*, *header: bool = False*) → list[dict[str, Any]]

Return the job result with specified format.

Parameters

- **job_id** (*int*) – Job ID
- **format** (*str*) – Output format of the job result information. "json" or "msgpack"
- **header** (*boolean*) – Includes Header or not. False or True

Returns
The query result of the specified job in.

job_result_format_each(*job_id: str*, *format: str*, *header: bool = False*, *store_tmpfile: bool = False*, *num_threads: int = 4*) → Iterator[dict[str, Any]]

Yield a row of the job result with specified format.

Parameters

- **job_id** (*int*) – job ID
- **format** (*str*) – Output format of the job result information. "json" or "msgpack"
- **header** (*bool*) – Include Header info or not "True" or "False"
- **store_tmpfile** (*bool*) – Download job result as a temporary file or not. Default is False. It works only when format is "msgpack". "True" or "False"
- **num_threads** (*int*) – Number of threads to download the job result when store_tmpfile is True. Default is 4.

Yields
The query result of the specified job in.

job_status(*job_id: str*) → str

Show job status :param job_id: job ID :type job_id: str

Returns
The status information of the given job id at last execution.

kill(*job_id: str*) → str | None

Stop the specific job if it is running.

Parameters
job_id (*str*) – Job Id to kill

Returns
Job status before killing

list_jobs(*_from: int = 0*, *to: int | None = None*, *status: str | None = None*, *conditions: dict[str, Any] | None = None*) → list[dict[str, Any]]

Show the list of Jobs.

Parameters

- **_from** (*int*) – Gets the Job from the nth index in the list. Default: 0
- **to** (*int, optional*) – Gets the Job up to the nth index in the list. By default, the first 20 jobs in the list are displayed
- **status** (*str, optional*) – Filter by given status. {"queued", "running", "success", "error"}
- **conditions** (*dict[str, Any], optional*) – Condition for `TIMESTAMPDIFF()` to search for slow queries. Avoid using this parameter as it can be dangerous.

Returns
a list of `dict` which represents a job

post(*path: str*, *params: dict[str, Any] | bytes | None = None*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

query(*q: str*, *type: Literal['hive', 'presto', 'trino', 'bulkload'] = 'hive'*, *db: str | None = None*, *result_url: str | None = None*, *priority: Literal[-2, -1, 0, 1, 2, 'VERY LOW', 'LOW', 'NORMAL', 'HIGH', 'VERY HIGH'] | None = None*, *retry_limit: int | None = None*, ***kwargs: Any*) → str

Create a job for given query.

Parameters

- **q** (*str*) – Query string.
- **type** (*str*) – Query type. *hive*, *presto*, *trino*, *bulkload*. Default: *hive*
- **db** (*str*) – Database name.
- **result_url** (*str*) – Result output URL. e.g., `postgresql://<username>:<password>@<hostname>:<port>/<database>/<table>`
- **priority** (*int or str*) – Job priority. In str, "Normal", "Very low", "Low", "High", "Very high". In int, the number in the range of -2 to 2.
- **retry_limit** (*int*) – Automatic retry count.
- ****kwargs** – Extra options.

Returns
Job ID issued for the query

Return type
str

raise_error(*msg: str*, *res: BaseHTTPResponse*, *body: bytes | str*) → None

show_job(*job_id: str*) → dict[str, Any]

Return detailed information of a Job.

Parameters
job_id (*str*) – job ID

Returns
Detailed information of a job

Return type
`dict`

JOB_PRIORITY: dict[str, int] = {'HIGH': 1, 'LOW': -1, 'NORM': 0, 'NORMAL': 0, 'VERY HIGH': 2, 'VERY LOW': -2, 'VERY-HIGH': 2, 'VERY-LOW': -2, 'VERY_HIGH': 2, 'VERY_LOW': -2}

tdclient.result_api

class tdclient.result_api.**ResultAPI**

Bases: `object`

Access to Result API.

This class is inherited by *tdclient.api.API*.

checked_json(*body: bytes*, *required: list[str]*) → dict[str, Any]

create_result(*name: str*, *url: str*, *params: ResultParams | None = None*) → bool

Create a new authentication with the specified name.

Parameters

- **name** (*str*) – Authentication name.
- **url** (*str*) – Url of the authentication to be created. e.g. "ftp://test.com/"
- **params** (*dict, optional*) – Extra parameters.

Returns
True if succeeded.

Return type
bool

delete_result(*name: str*) → bool

Delete the authentication having the specified name.

Parameters
name (*str*) – Authentication name.

Returns
True if succeeded.

Return type
bool

get(*path: str*, *params: dict[str, Any] | None = None*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

list_result() → list[tuple[str, str, None]]

Get the list of all the available authentications.

Returns

The list of tuple of name, Result output url, and
organization name (always *None* for api compatibility).

Return type
[(str, str, None)]

post(*path: str*, *params: dict[str, Any] | bytes | None = None*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

raise_error(*msg: str*, *res: BaseHTTPResponse*, *body: bytes*) → None

tdclient.schedule_api

class tdclient.schedule_api.**ScheduleAPI**

Bases: object

Access to Schedule API

This class is inherited by *tdclient.api.API*.

checked_json(*body: bytes*, *required: list[str]*) → dict[str, Any]

create_schedule(*name: str*, *params: ScheduleParams | None = None*) → datetime | None

Create a new scheduled query with the specified name.

Parameters

- **name** (*str*) – Scheduled query name.
- **params** (*dict, optional*) – Extra parameters.
  - **type (str):**
    Query type. {"presto", "hive"}. Default: "hive"
  - **database (str):**
    Target database name.
  - **timezone (str):**
    Scheduled query's timezone. e.g. "UTC" For details, see also: https://gist.github.com/frsyuki/4533752
  - **cron (str, optional):**
    Schedule of the query. {"@daily", "@hourly", "10 * * * *" (custom cron)} See also: https://docs.treasuredata.com/articles/#!pd/Scheduling-Jobs-Using-TD-Console
  - **delay (int, optional):**
    A delay ensures all buffered events are imported before running the query. Default: 0
  - **query (str):**
    Is a language used to retrieve, insert, update and modify data. See also: https://docs.treasuredata.com/articles/#!pd/SQL-Examples-of-Scheduled-Queries
  - **priority (int, optional):**
    Priority of the query. Range is from -2 (very low) to 2 (very high). Default: 0
  - **retry_limit (int, optional):**
    Automatic retry count. Default: 0
  - **engine_version (str, optional):**
    Engine version to be used. If none is specified, the account's default engine version would be set. {"stable", "experimental"}
  - **pool_name (str, optional):**
    For Presto only. Pool name to be used, if not specified, default pool would be used.
  - **result (str, optional):**
    Location where to store the result of the query. e.g. 'tableau://user:password@host.com:1234/datasource'

Returns

Start date time.

Return type
datetime.datetime

delete_schedule(*name: str*) → tuple[str, str]

Delete the scheduled query with the specified name.

Parameters
name (*str*) – Target scheduled query name.

Returns
Tuple of cron and query.

Return type
(str, str)

get(*path: str*, *params: dict[str, Any] | None = None*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

history(*name: str*, *_from: int = 0*, *to: int | None = None*) → list[tuple[Any, ...]]

Get the history details of the saved query for the past 90days.

Parameters

- **name** (*str*) – Target name of the scheduled query.
- **_from** (*int, optional*) – Indicates from which nth record in the run history would be fetched. Default: 0. Note: Count starts from zero. This means that the first record in the list has a count of zero.
- **to** (*int, optional*) – Indicates up to which nth record in the run history would be fetched. Default: 20

Returns
History of the scheduled query.

Return type
dict

list_schedules() → list[dict[str, Any]]

Get the list of all the scheduled queries.

Returns
str, cron:str, query:str, database:str, result_url:str)]

Return type
[(*name*

post(*path: str*, *params: dict[str, Any] | bytes | None = None*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

raise_error(*msg: str*, *res: BaseHTTPResponse*, *body: bytes*) → None

run_schedule(*name: str*, *time: int*, *num: int | None = None*) → list[tuple[Any, Any, datetime | None]]

Execute the specified query.

Parameters

- **name** (*str*) – Target scheduled query name.
- **time** (*int*) – Time in Unix epoch format that would be set as TD_SCHEDULED_TIME

- **num** (*int, optional*) – Indicates how many times the query will be executed. Value should be 9 or less.

Returns
[(job_id:int, type:str, scheduled_at:str)]

Return type
list of tuple

update_schedule(*name: str*, *params: ScheduleParams | None = None*) → datetime | None

Update the scheduled query.

Parameters

- **name** (*str*) – Target scheduled query name.
- **params** (*ScheduleParams | None*) – Extra parameters.
  - **type (str):**
    Query type. {"presto", "hive"}. Default: "hive"
  - **database (str):**
    Target database name.
  - **timezone (str):**
    Scheduled query's timezone. e.g. "UTC" For details, see also: https://gist.github.com/frsyuki/4533752
  - **cron (str, optional):**
    Schedule of the query. {"@daily", "@hourly", "10 * * * *" (custom cron)} See also: https://docs.treasuredata.com/articles/#!pd/Scheduling-Jobs-Using-TD-Console
  - **delay (int, optional):**
    A delay ensures all buffered events are imported before running the query. Default: 0
  - **query (str):**
    Is a language used to retrieve, insert, update and modify data. See also: https://docs.treasuredata.com/articles/#!pd/SQL-Examples-of-Scheduled-Queries
  - **priority (int, optional):**
    Priority of the query. Range is from -2 (very low) to 2 (very high). Default: 0
  - **retry_limit (int, optional):**
    Automatic retry count. Default: 0
  - **engine_version (str, optional):**
    Engine version to be used. If none is specified, the account's default engine version would be set. {"stable", "experimental"}
  - **pool_name (str, optional):**
    For Presto only. Pool name to be used, if not specified, default pool would be used.
  - **result (str, optional):**
    Location where to store the result of the query. e.g. 'tableau://user:password@host.com:1234/datasource'

tdclient.schedule_api.**history_to_tuple**(*m: dict[str, Any]*) → tuple[datetime | None, Any, str, Any, Any, datetime | None, datetime | None, Any, Any, Any]

tdclient.schedule_api.**job_to_tuple**(*m: dict[str, Any]*) → tuple[str | None, str, datetime | None]

tdclient.schedule_api.**schedule_to_tuple**(*m: dict[str, Any]*) → dict[str, Any]

tdclient.server_status_api

class tdclient.server_status_api.**ServerStatusAPI**

Bases: object

Access to Server Status API

This class is inherited by *tdclient.api.API*.

checked_json(*body: bytes*, *required: list[str]*) → dict[str, Any]

get(*path: str*, *params: dict[str, Any] | None = None*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

server_status() → str

Show the status of Treasure Data

Returns
status

Return type
str

tdclient.table_api

class tdclient.table_api.**TableAPI**

Bases: object

Access to Table API

This class is inherited by *tdclient.api.API*.

change_database(*db: str*, *table: str*, *dest_db: str*) → bool

Move a target table from it's original database to new destination database.

Parameters

- **db** (*str*) – Target database name.
- **table** (*str*) – Target table name.
- **dest_db** (*str*) – Destination database name.

Returns
True if succeeded

Return type
bool

checked_json(*body: bytes*, *required: list[str]*) → dict[str, Any]

create_log_table(*db: str*, *table: str*) → bool

Create a new table in the database and registers it in PlazmaDB.

Parameters

- **db** (*str*) – Target database name.
- **table** (*str*) – Target table name.

Returns
True if succeeded.

Return type
bool

delete_table(*db: str*, *table: str*) → str

Delete the specified table.

Parameters

- **db** (*str*) – Target database name.
- **table** (*str*) – Target table name.

Returns
Type information of the table (e.g. "log").

Return type
str

get(*path: str*, *params: dict[str, Any] | None = None*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

list_tables(*db: str*) → dict[str, Any]

Gets the list of table in the database.

Parameters
db (*str*) – Target database name.

Returns
Detailed table information.

Return type
dict

Examples

```
>>> td.api.list_tables("my_db")
{ 'iris': {'id': 21039862,
  'name': 'iris',
  'estimated_storage_size': 1236,
  'counter_updated_at': '2019-09-18T07:14:28Z',
  'last_log_timestamp': datetime.datetime(2019, 1, 30, 5, 34, 42,␣
↪tzinfo=tzutc()),
  'delete_protected': False,
  'created_at': datetime.datetime(2019, 1, 30, 5, 34, 42, tzinfo=tzutc()),
  'updated_at': datetime.datetime(2019, 1, 30, 5, 34, 46, tzinfo=tzutc()),
  'type': 'log',
  'count': 150,
  'schema': [['sepal_length', 'double', 'sepal_length'],
   ['sepal_width', 'double', 'sepal_width'],
   ['petal_length', 'double', 'petal_length'],
   ['petal_width', 'double', 'petal_width'],
   ['species', 'string', 'species']],
  'expire_days': None,
  'last_import': datetime.datetime(2019, 9, 18, 7, 14, 28, tzinfo=tzutc())},
}
```

post(*path: str*, *params: dict[str, Any] | bytes | None = None*, *headers: dict[str, str] | None = None*, ***kwargs: Any*) → AbstractContextManager[BaseHTTPResponse]

raise_error(*msg: str*, *res: BaseHTTPResponse*, *body: bytes | str*) → None

swap_table(*db: str*, *table1: str*, *table2: str*) → bool

Swap the two specified tables with each other belonging to the same database and basically exchanges their names.

Parameters

- **db** (*str*) – Target database name
- **table1** (*str*) – First target table for the swap.
- **table2** (*str*) – Second target table for the swap.

Returns
True if succeeded.

Return type
bool

tail(*db: str*, *table: str*, *count: int*, *to: Any = None*, *_from: Any = None*, *block: Any = None*) → list[dict[str, Any]]

Get the contents of the table in reverse order based on the registered time (last data first).

Parameters

- **db** (*str*) – Target database name.
- **table** (*str*) – Target table name.
- **count** (*int*) – Number for record to show up from the end.
- **to** – Deprecated parameter.
- **_from** – Deprecated parameter.
- **block** – Deprecated parameter.

Returns
Contents of the table.

Return type
[dict]

update_expire(*db: str*, *table: str*, *expire_days: int*) → bool

Update the expire days for the specified table

Parameters

- **db** (*str*) – Target database name.
- **table** (*str*) – Target table name.
- **expire_days** (*int*) – Number of days where the contents of the specified table would expire.

Returns
True if succeeded.

Return type
bool

update_schema(*db: str*, *table: str*, *schema_json: str*) → bool

Update the table schema.

Parameters

- **db** (*str*) – Target database name.
- **table** (*str*) – Target table name.
- **schema_json** (*str*) – Schema format JSON string. See also: ~`Client.update_schema` e.g. '[["sep_len", "long", "sep_len"], ["sep_wid", "long", "sep_wid"]]'

Returns

True if succeeded.

Return type

bool

6.2.5 Misc

tdclient.errors

exception tdclient.errors.**APIError**

Bases: Exception

Base exception for API-related errors.

exception tdclient.errors.**AlreadyExistsError**

Bases: *APIError*

Exception raised when a resource already exists (HTTP 409).

exception tdclient.errors.**AuthError**

Bases: *APIError*

Exception raised for authentication errors (HTTP 401).

exception tdclient.errors.**DataError**

Bases: *DatabaseError*

Exception for errors due to problems with the processed data (PEP 249).

exception tdclient.errors.**DatabaseError**

Bases: *Error*

Exception for errors related to the database (PEP 249).

exception tdclient.errors.**Error**

Bases: Exception

Base class for database-related errors (PEP 249).

exception tdclient.errors.**ForbiddenError**

Bases: *APIError*

Exception raised for forbidden access errors (HTTP 403).

exception tdclient.errors.**IntegrityError**

Bases: *DatabaseError*

Exception for errors related to relational integrity (PEP 249).

exception tdclient.errors.**InterfaceError**

Bases: *Error*

Exception for errors related to the database interface (PEP 249).

exception tdclient.errors.**InternalError**

Bases: *DatabaseError*

Exception for internal database errors (PEP 249).

exception tdclient.errors.**NotFoundError**

Bases: *APIError*

Exception raised when a resource is not found (HTTP 404).

exception tdclient.errors.**NotSupportedError**

Bases: *DatabaseError*

Exception for unsupported operations (PEP 249).

exception tdclient.errors.**OperationalError**

Bases: *DatabaseError*

Exception for errors related to database operation (PEP 249).

exception tdclient.errors.**ParameterValidationError**

Bases: Exception

Exception raised when parameter validation fails.

exception tdclient.errors.**ProgrammingError**

Bases: *DatabaseError*

Exception for programming errors (PEP 249).

tdclient.util

tdclient.util.**create_msgpack**(*items: list[dict[str, Any]]*) → bytes

Create msgpack streaming bytes from list

Parameters

items (*list of dict*) – target list

Returns

Converted msgpack streaming (bytes)

Examples

```
>>> t1 = int(time.time())
>>> l1 = [{"a": 1, "b": 2, "time": t1}, {"a":3, "b": 6, "time": t1}]
>>> create_msgpack(l1)
b'\x83\xa1a\x01\xa1b\x02\xa4time\xce]\xa5X\xa1\x83\xa1a\x03\xa1b\x06\xa4time\xce]\
↪xa5X\xa1'
```

tdclient.util.**create_url**(*tmpl: str*, ***values: Any*) → str

Create url with values

Parameters

- **tmpl** (*str*) – url template

- **values** (*dict*) – values for url

tdclient.util.**csv_dict_record_reader**(*file_like: BinaryIO*, *encoding: str*, *dialect: str | type[Dialect]*) → Iterator[dict[str, str]]

Yield records from a CSV input using csv.DictReader.

This is a reader suitable for use by **`tdclient.util.read_csv_records`_**.

It is used to read CSV data when the column names are read from the first row in the CSV data.

Parameters

- **file_like** – acts like an instance of io.BufferedIOBase. Reading from it returns bytes.
- **encoding** (*str*) – the name of the encoding to use when turning those bytes into strings.
- **dialect** (*str | type[csv.Dialect]*) – the name of the CSV dialect to use, or a Dialect class.

Yields

For each row of CSV data read from *file_like*, yields a dictionary whose keys are column names (determined from the first row in the CSV data) and whose values are the column values.

tdclient.util.**csv_text_record_reader**(*file_like: BinaryIO*, *encoding: str*, *dialect: str | type[Dialect]*, *columns: list[str]*) → Iterator[dict[str, str]]

Yield records from a CSV input using csv.reader and explicit column names.

This is a reader suitable for use by **`tdclient.util.read_csv_records`_**.

It is used to read CSV data when the column names are supplied as an explicit *columns* parameter.

Parameters

- **file_like** – acts like an instance of io.BufferedIOBase. Reading from it returns bytes.
- **encoding** (*str*) – the name of the encoding to use when turning those bytes into strings.
- **dialect** (*str | type[csv.Dialect]*) – the name of the CSV dialect to use, or a Dialect class.

Yields

For each row of CSV data read from *file_like*, yields a dictionary whose keys are column names (determined by *columns*) and whose values are the column values.

tdclient.util.**get_or_else**(*hashmap: dict[str, str]*, *key: str*, *default_value: str | None = None*) → str | None

Get value or default value

It differs from the standard dict `get` method in its behaviour when *key* is present but has a value that is an empty string or a string of only spaces.

Parameters

- **hashmap** (*dict*) – target dictionary with string values
- **key** (*str*) – key to look up
- **default_value** (*str | None*) – default value to return if key is missing or value is empty/whitespace

Example

```
>>> get_or_else({'k': 'nonspace'}, 'k', 'default')
'nonspace'
>>> get_or_else({'k': ''}, 'k', 'default')
'default'
>>> get_or_else({'k': '    '}, 'k', 'default')
'default'
```

Returns
The value of *key* or *default_value*

tdclient.util.**guess_csv_value**(*s: str*) → int | float | str | bool | None

Determine the most appropriate type for *s* and return it.

Tries to interpret *s* as a more specific datatype, in the following order, and returns the first that succeeds:

1. As an integer
2. As a floating point value
3. If it is “false” or “true” (case insensitive), then as a boolean
4. If it is “” or “none” or “null” (case insensitive), then as None
5. As the string itself, unaltered

Parameters
s (*str*) – a string value, assumed to have been read from a CSV file.

Returns
A good guess at a more specific value (int, float, str, bool or None)

tdclient.util.**merge_dtypes_and_converters**(*dtypes: dict[str, str] | None = None*, *converters: dict[str, Callable[[str], Any]] | None = None*) → dict[str, Callable[[str], Any]]

Generate a merged dictionary from those given.

Parameters

- **dtypes** (*optional dict*) – A dictionary mapping column name to “dtype” (datatype), where “dtype” may be any of the strings ‘bool’, ‘float’, ‘int’, ‘str’ or ‘guess’.
- **converters** (*optional dict*) – A dictionary mapping column name to a callable. The callable should take a string as its single argument, and return the result of parsing that string.

Internally, the *dtypes* dictionary is converted to a temporary dictionary of the same form as *converters* - that is, mapping column names to callables. The “data type” string values in *dtypes* are converted to the Python builtins of the same name, and the value *“guess”* is converted to the **`tdclient.util.guess_csv_value`_** callable.

Example

```
>>> merge_dtypes_and_converters(
...     dtypes={'col1': 'int', 'col2': 'float'},
...     converters={'col2': int},
... )
{'col1': int, 'col2': int}
```

Returns
(dict) A dictionary which maps column names to callables. If a column name occurs in both input dictionaries, the callable specified in *converters* is used.

tdclient.util.**normalize_connector_config**(*config: dict[str, Any]*) → dict[str, Any]

Normalize connector config

This is porting of TD CLI's ConnectorConfigNormalizer#normalized_config. see also: https://github.com/treasure-data/td/blob/15495f12d8645a7b3f6804098f8f8aca72de90b9/lib/td/connector_config_normalizer.rb#L7-L30

Parameters
config (*dict*) – A config to be normalized

Returns
Normalized configuration

Return type
dict

Examples

Only with `in` key in a config. `>>>` config = {"in": {"type": "s3"}} `>>>` normalize_connector_config(config) {'in': {'type': 's3'}, 'out': {}, 'exec': {}, 'filters': []}

With `in`, `out`, `exec`, and `filters` in a config. `>>>` config = { ... "in": {"type": "s3"}, ... "out": {"mode": "append"}, ... "exec": {"guess_plugins": ["json"]}, ... "filters": [{"type": "speedometer"}], ... } `>>>` normalize_connector_config(config) {'in': {'type': 's3'}, 'out': {'mode': 'append'}, 'exec': {'guess_plugins': ['json']}, 'filters': [{'type': 'speedometer'}]}

tdclient.util.**normalized_msgpack**(*value: Any*) → Any

Recursively convert int to str if the int "overflows".

Parameters
value (*list, dict, int, float, str, bool or None*) – value to be normalized

If *value* is a list, then all elements in the list are (recursively) normalized.

If *value* is a dictionary, then all the dictionary keys and values are (recursively) normalized.

If *value* is an integer, and outside the range `-(1 << 63)` to `(1 << 64)`, then it is converted to a string.

Otherwise, *value* is returned unchanged.

Returns
Normalized value

tdclient.util.**parse_csv_value**(*k: str*, *s: str*, *converters: dict[str, Callable[[str], Any]] | None = None*) → Any

Given a CSV (string) value, work out an actual value.

Parameters

- **k** (*str*) – The name of the column that the value belongs to.
- **s** (*str*) – The value as read from the CSV input.
- **converters** (*optional dict*) – A dictionary mapping column name to callable.

If *converters* is given, and there is a key matching *k* in *converters*, then `converters[k](s)` will be called to work out the return value. Otherwise, **`tdclient.util.guess_csv_value`_** will be called with *s* as its argument.

> **Warning**
>
> No attempt is made to cope with any errors occurring in a callable from the *converters* dictionary. So if `int` is called on the string `"not-an-int"` the resulting `ValueError` is not caught.

Example

```
>>> repr(parse_csv_value('col1', 'A string'))
'A string'
>>> repr(parse_csv_value('col1', '10'))
10
>>> repr(parse_csv_value('col1', '10', {'col1': float, 'col2': int}))
10.0
```

Returns
The value for the CSV column, after parsing by a callable from *converters*, or after parsing by `tdclient.util.guess_csv_value`_.

`tdclient.util.`**`parse_date`**(*s: str | None*) → datetime | None

Parse date from str to datetime

TODO: parse datetime using an optional format string

For now, this does not use a format string since API may return date in ambiguous format :(

Parameters
s (*str | None*) – target str, or None

Returns
datetime or None

`tdclient.util.`**`read_csv_records`**(*csv_reader: Iterator[dict[str, str]]*, *dtypes: dict[str, str] | None = None*, *converters: dict[str, Callable[[str], Any]] | None = None*, ***kwargs: Any*) → Iterator[dict[str, Any]]

Read records using csv_reader and yield the results.

`tdclient.util.`**`validate_record`**(*record: dict[str, Any]*) → bool

Check that *record* contains a key called “time”.

Parameters

- **record** (*dict*) – a dictionary representing a data record, where the
- **"columns".** (*keys name the*)

Returns
True if there is a key called “time” (it actually checks for `"time"` (a string) and `b"time"` (a binary)). False if there is no key called “time”.

6.3 Version History

6.3.1 v1.7.0 (2026-01-29)

- Set minimum version of dependencies

6.3.2 v1.6.0 (2025-11-14)

- Support Python 3.13 and 3.14, drop 3.8 and 3.9
- Support type hints

6.3.3 v1.5.0 (2025-08-14)

- Added timeout parameter to BulkImport.perform() method

6.3.4 v1.4.0 (2025-08-05)

- Support trino type query
- Remove partial_delete api

6.3.5 v1.2.1 (2020-07-01)

- Ensure `endpoint` starts with `'https://'` if scheme not given (#96)

6.3.6 v1.2.0 (2019-12-05)

- Add new (optional) parameters to `ImportApi.import_files`, `BulkImportApi.bulk_import_upload_file` and `BulkImport.upload_file`. (#85) The `dtypes` and `converters` parameters allow better control of the import of CSV data (#83). This is modelled on the approach taken by pandas.
- Ensure `config` key for `ConnectorAPI.connector_guess` (#84)

6.3.7 v1.1.0 (2019-10-16)

- Move `normalized_msgpack()` from `tdclient.api` to `tdclient.util` module (#79)
- Add `tdclient.util.create_msgpack()` to support creating msgpack streaming from list (#79)

6.3.8 v1.0.1 (2019-10-10)

- Fix `wait_interval` handling for `BulkImport.perform` appropriately (#74)
- Use `io.TextIOWrapper` to prevent `"x85"` issue creating None (#77)

6.3.9 v1.0.0 (2019-09-27)

- Drop Python 2 support (#60)
- Remove deprecated functions as follows (#76):
  - `TableAPI.create_item_table`
  - `UserAPI.change_email`, `UserAPI.change_password`, and `UserAPI.change_my_password`
  - `JobAPI.hive_query`, and `JobAPI.pig_query`
- Support `TableAPI.tail` and `TableAPI.change_database` (#64, #71)
- Introduce documentation site (#65, #66, #70, #72)

6.3.10 v0.14.0 (2019-07-11)

- Remove ACL and account APIs (#56, #58)
- Fix PyOpenSSL issue which causes pandas-td error (#59)

6.3.11 v0.13.0 (2019-03-29)

- Change msgpack-python to msgpack (#50)
- Dropped 3.3 support as it has already been EOL'd (#52)
- Set urllib3 minimum version as v1.24.1 (#51)

6.3.12 v0.12.0 (2018-05-31)

- Avoided to declare library dependencies too tightly within this project since this is a library project (#42)
- Got rid of all configurations for Python 2.6 completely (#42)

6.3.13 v0.11.1 (2018-05-21)

- Added 3.6 as test target. No functional changes have applied since 0.11.0 (#41)

6.3.14 v0.11.0 (2018-05-21)

- Support missing parameters in JOB API (#39, #40)

6.3.15 v0.10.0 (2017-11-01)

- Ignore empty string in job's `start_at` and `end_at` (#35, #36)

6.3.16 v0.9.0 (2017-02-27)

- Add validation to part names for bulk upload

6.3.17 v0.8.0 (2016-12-22)

- Fix unicode encoding issues on Python 2.x (#27, #28, #29)

6.3.18 v0.7.0 (2016-12-06)

- Fix for tdclient tables data not populating
- `TableAPI.list_tables` now returns a dictionary instead of a tuple

6.3.19 v0.6.0 (2016-09-27)

- Generate universal wheel by default since there's no binary in this package
- Add missing support for `created_time` and `user_name` from `/v3/schedule/list` API (#20, #21)
- Use keyword arguments for initializing model attributes (#22)

6.3.20 v0.5.0 (2016-06-10)

- Prevent retry after PUT request failures. This is the same behavior as https://github.com/treasure-data/td-client-ruby (#16)
- Support HTTP proxy authentication (#17)

6.3.21 v0.4.2 (2016-03-15)

- Catch exceptions on parsing date time string

6.3.22 v0.4.1 (2016-01-19)

- Fix Data Connector APIs based on latest td-client-ruby's implementation (#14)

6.3.23 v0.4.0 (2015-12-14)

- Avoid an exception raised when a `start` is not set for a schedule (#12)
- Fix getting database names of job objects (#13)
- Add Data Connector APIs
- Add deprecation warnings on the usage of "item tables"
- Show `cumul_retry_delay` in retry messages

6.3.24 v0.3.2 (2015-08-01)

- Fix bugs in `ScheduledJob` and `Schedule` models

6.3.25 v0.3.1 (2015-07-10)

- Fix `OverflowError` on importing integer value longer than 64 bit length which is not supported by msgpack specification. Those values will be converted into string.

6.3.26 v0.3.0 (2015-07-03)

- Add Python Database API (PEP 0249) compatible connection and cursor.
- Add varidation to the part name of a bulk import. It should not contain '/'.
- Changed default wait interval of job models from 1 second to 5 seconds.
- Fix many potential problems/warnings found by landscape.io.

6.3.27 v0.2.1 (2015-06-20)

- Set default timeout of API client as 60 seconds.
- Change the timeout of API client from `sum(connect_timeout, read_timeout, send_timeout)` to `max(connect_timeout, read_timeout, send_timeout)`
- Change default user-agent of client from `TD-Client-Python:{version}` to `TD-Client-Python/{version}` to comply RFC2616

6.3.28 v0.2.0 (2015-05-28)

- Improve the job model. Now it retrieves the job values automatically after the invocation of `wait`, `result` and `kill`.
- Add a property `result_schema` to `Job` model to provide the schema of job result
- Improve the bulk import model. Add a convenient method named `upload_file` to upload a part from file-like object.
- Support CSV/TSV format on both streaming import and bulk import
- Change module name; `tdclient.model` -> `tdclient.models`

6.3.29 v0.1.11 (2015-05-17)

- Fix API client to retry POST requests properly if `retry_post_requests` is set to `True` (#5)
- Show warnings if imported data don't have `time` column

6.3.30 v0.1.10 (2015-03-30)

- Fixed a JSON parse error in `job.result_format("json")` with multiple result rows (#4)
- Refactored model classes and tests

6.3.31 v0.1.9 (2015-02-26)

- Stopped using syntax added in recent Python releases

6.3.32 v0.1.8 (2015-02-26)

- Fix SSL verification errors on Python 2.7 on Windows environment. Now it uses `certifi` to verify SSL certificates if it is available.

6.3.33 v0.1.7 (2015-02-26)

- Fix support for Windows environments
- Fix byte encoding problem in `tdclient.api.API#import_file` on Python 3.x

6.3.34 v0.1.6 (2015-02-12)

- Support specifying job priority in its name (e.g. "NORMAL", "HIGH", etc.)
- Convert job priority number to its name (e.g. 0 => "NORMAL", 1 => "HIGH", etc.)
- Fix a broken behavior in `tdclient.model.Job#wait` when specifying timeout
- Fix broken `tdclient.client.Client#database()` which is used from `tdclient.model.Table#permission()`
- Fix broken `tdclient.Client.Client#results()`

6.3.35 v0.1.5 (2015-02-10)

- Fix local variable scope problem in `tdclient.api.show_job` (#2)
- Fix broken multiple assignment in `tdclient.model.Job#_update_status` (#3)

6.3.36 v0.1.4 (2015-02-06)

- Add new data import function of `tdclient.api.import_file` to allow importing data from file-like object or an existing file on filesystem.
- Fix an encoding error in `tdclient.api.import_data` on Python 2.x
- Add missing import to fix broken `tdclient.model.Job#wait`
- Use `td.api.DEFAULT_ENDPOINT` for all requests

6.3.37 v0.1.3 (2015-01-24)

- Support PEP 343 in `tdclient.Client` and remove `contextlib` from example
- Add deprecation warnings to `hive_query` and `pig_query` of `tdclient.api.API`
- Add `tdclient.model.Job#id` as an alias of `tdclient.model.Job#job_id`
- Parse datatime properly returned from `tdclient.Client#create_schedule`
- Changed `tdclient.model.Job#query` as a property since it won't be modified during the execution
- Allow specifying query options from `tdclient.model.Database#query`

6.3.38 v0.1.2 (2015-01-21)

- Fix broken PyPI identifiers
- Update documentation

6.3.39 v0.1.1 (2015-01-21)

- Improve the verification of SSL certificates on RedHat and variants
- Implement `wait` and `kill` in `tdclient.model.Job`
- Change the "Development Status" from Alpha to Beta

6.3.40 v0.1.0 (2015-01-15)

- Initial public release

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W