TRExt Documentation

Release 0.1

Vathsala Achar

May 26, 2017

Contents

1 Contents **3**
- 1.1 TRExt . . . 3
- 1.2 Usage Instructions . . . 5
- 1.3 API Documentation . . . 7

2 Indices and tables **13**

Python Module Index **15**

TRExt is short for Tableau Refresh Extract (Externally).

TRExt provides an API to create a .tde extract from a database and publish to a Tableau Server, and this does not need the Tableau Desktop to set up.

This documentation contains instructions on how to best use TRExt with your database, as well as the API documentation if you want to get into the details.

The repository is available on GitHub if you wish to fork, contribute or just install and play around.

CHAPTER 1

Contents

TRExt

TRExt is short for Tableau Refresh Extract (Externally).

TRExt is a means to refresh a Tableau Extract (.tde files) externally so the Tableau Server can serve visual content without having to compete for resources while refreshing extracts internally.

Dependencies

The main dependencies are:

- Tableau SDK
- pyodbc

The repo also supports

- pyodbc wrapper such as EXASol Python SDK

Installation

You need *pip* to install TRExt.

You can install the latest version of the package straight from PyPI using:

```
$ pip install trext
```

You can also directly install from GitHub directly using:

```
$ pip install git+git@github.com:AtheonAnalytics/trext.git
```

or

```
$ pip install git+https://github.com/AtheonAnalytics/trext.git
```

Usage

Create an extract

```
>>> import trext
>>> tde = trext.Extract()
>>> connection_string = "appropriate db connection string"
>>> tde.create("db.schema.table", conn_string=connection_string, dbtype='exasol')
Created!
>>> tde.location
/temp/extract.tde
```

Publish to Tableau Server (overwrites existing extract)

```
>>> tableau_auth_details = ("username", "password")
>>> publish_details = ("site_content_url", "project_name")
>>> tde.publish("tableau server address", auth=tableau_auth_details, params=publish_
↪details)
Published!
```

Refreshing an extract is now replaced with creating and publishing an extract. You can use this is conjunction with TabAuto (not yet open source) or with Tableau's server-client-python library to get the datasource names that need refreshing.

Documentation

More detailed documentation is available at http://trext.rtfd.io

---

Disclaimer

TRExt is still a Work-in-Progress

I wrote most of this codebase when Tableau SDK was released for Tableau 8 and never got around to moving it from a POC/local copy to open source, so this a rough-and-ready type of library.

This is fair warning to anyone who uses this library: there will be bugs, bad documentation and no tests for a short while till I fix it up. So *please use with care* and if you find issues submit a bug report or a PR.

If you want to contribute and add tests, better documentation, new connectors, cleaner interface etc, *please do* and submit a PR.

Oh and don't forget to add yourself to AUTHORS

Note: I have tested TRExt only on a Linux distro, so if you find any issues on other Operating Systems please do create a bug report and I can try to fix it, but if you do know how to fix it please also submit a PR.

Usage Instructions

The current version of TRExt works with:

- Python 2.7
- pyodbc == 4.0.16
- Tableau SDK <= 10.1.4 (see *Notes*)
- EXASol Python SDK >=5.0.13 and <= 5.0.17

Installation of dependencies

You will need to install Tableau SDK and pyodbc as a minimum to use this library. The EXASol library will be needed if you want to connect to EXASol to create an extract.

pyodbc

PyODBC should be installed when you install TRExt from pip see *Installation*. But if you have any issues setting up pyodbc to connect to a database you might find the PyODBC wiki useful.

Tableau SDK Installation

The documentation for Tableau SDK is available at the Tableau Documentation.

The library I have used to develop and test with is available in the vendor folder and this is only for Linux Distribution. If you prefer to use versions for other Operating systems you can find them here.

The installation of the Linux version is as follows:

Download the correct *tar.gz* file for Python (32 or 64 bit depending on your OS) and then,

```
$ tar -xzvf Tableau-SDK-Python-Linux-xxBit-10-x-x.tar.gz
$ cd /Tableau*/
$ python setup.py install
```

EXASol SDK Installation

Skip to the next section if you do not use EXASol.

The EXASol Python SDK can be downloaded from their Download section. We use version 5.0.17 for this version of TRExt but I have also tested on some older versions.

Download from Packages and SDK the version you want, preferably in the *tar.gz* format and install as follows,

```
$ tar -xzvf EXASolution_SDK-5.0.xx.tar.gz
$ cd /EXASolution_SDK-5.0.xx/Python/
$ python setup.py install
```

Additional Setup

To connect to MSSQL you need to set up the MSSQL driver and driver manager.

Installation of the ODBC driver for Linux is available from Microsoft.

Instructions on setting up the driver manager to connect to MSSQL is available from pyodbc.

Once you have your connection information in *odbc.ini* you should be able to test the connection with the *DSN* parameter using pyodbc. You should get a pyodbc.Connection object if the connection was successful.

```
$ python -c 'import pyodbc; print(pyodbc.connect("DSN=MySQLServerDatabase;
↪UID=username;PWD=password"))'
<pyodbc.Connection object at 0x7f8597333200>
```

Usage of the api

Create a TRExt Extract

The TRExt Extract is the interface to the *create* and *publish* methods. Initialise the Extract,

```
>>> import trext
>>> tde = trext.Extract()
```

Create an extract for MSSQL

Assuming that you can connect to the MSSQL server, you can now create a .tde extract using the following,

```
>>> conn_string = "DSN=MySQLServerDatabase;UID=username;PWD=password"
>>> tde.create("db.schema.table", conn_string=conn_string)
Created!
```

Create an extract for EXASol

Note here that in the create api we use an extra argument called *dbtype* set to *'exasol'*. This is how TRExt extends to other databases. Currently only MSSQL and EXAsol have been tested.

```
>>> conn_string = "DSN=EXAServer"
>>> tde.create("db.schema.table", conn_string=conn_string, dbtype='exasol')
Created!
```

Location of the extract

Once you have created the extract and you want to know the location of your extract simply do,

```
>>> tde.location
/temp/location/of/extract.tde
```

Publish to Tableau Server

The default behaviour of publish is to overwrite the existing extract. This will be extended in the future versions.

```
>>> tableau_auth_details = ("username", "password")
>>> publish_details = ("site_content_url", "project_name")
>>> tde.publish("tableau server address", auth=tableau_auth_details, params=publish_
↪details)
Published!
```

Close the Extract

Once you are done creating and/or publishing an extract, perform the *close* operation,

```
>>> tde.close()
```

This api ensures that the tde created locally gets destroyed.

Publish existing .tde to Tableau Server

You can also use this api to publish a local .tde file to the Tableau Server, simply set the location of the TRExt extract to the path of the .tde you want to publish

```
>>> tde.location = "local/path/to/extract.tde"
>>> tde.publish("tableau server address", auth=tableau_auth_details, params=publish_
↪details)
Published!
```

Notes

(*) I have not tested to see if Tableau SDK still supports versions 8 and 9 but this code was based on Tableau SDK for Tableau 8; another area that needs testing and improvement for TRExt.

API Documentation

trext.api

trext.api.Extract

class `trext.api.`**`Extract`**(*is_temp=True*)

Bases: `object`

A Tableau Extract.

Provides the endpoint to create and/or publish .tde extracts. Usage:

Create an extract >>> import trext >>> tde = trext.Extract() >>> connection_string = “appropriate db connection string” >>> tde.create(“db.schema.table”, conn_string=connection_string) Created! >>> tde.location /temp/extract.tde

Publish to Tableau Server (overwrites existing extract)

```
>>> tableau_auth_details = ("username", "password")
>>> publish_details = ("site_content_url", "project_name")
>>> tde.publish("tableau server address", auth=tableau_auth_details,␣
↪params=publish_details)
Published!
```

Clean up after create and/or publish >>> tde.close()

close()
Clean up on exit. Delete the extract only if temporary flag is True

Returns if it is not a temporary extract

create(*view_or_table_name*, *conn_string*, *dbtype=None*)
Method to create an extract based on a view or a table on a database

Parameters

- **view_or_table_name** – view or table to create an extract from
- **conn_string** – connection string to the database
- **dbtype** – type of database to connect to

Returns *Created!* or *Failed!* message on creation

location

Returns location of the .tde file if it was created or set up

publish(*host_address*, *auth*, *params*)
Publish to Tableau Server (overwrites existing extract)

Parameters

- **host_address** – Address of the Tableau server to publish to
- **auth** – a tuple of username and password for authentication
- **params** – currently a typle of two parameters: site to publish to and project name

Returns Message on completing publishing

trext.db

The db package deals with connecting to the Database, pulling the column metadata and the data from the tables, and modifying the data to match the Tableau Type.

trext.db.conn

class trext.db.conn.**AnyDB**(*connection_string*, *dbtype=None*)
Bases: object

close()

get_cursor()

trext.db.consume

class trext.db.consume.**DBConsumer**(*cursor*, *view_or_table_name*, *dbtype*)
Bases: object

Pulls the view/table metadata and data from the database.

get_table_data()
Generator that returns the data row by row of the view/table.

Returns row of data

get_table_definition()
Generator that returns the column name, type and the column position of the view/table.

Returns column name, column position, column type

trext.db.fill

class trext.db.fill.**ExtractFiller**(*table*, *table_definition*, *column_metadata*)
Bases: object

Fills the extract skeleton with cleaned and formatted data.

insert_data_to_extract(*db_data_row*)
Inserts the data row by row into the tableau extract skeleton

Parameters **db_data_row** – row from the database

trext.db.typemap

trext.db.typemap.**get_type**(*db_type*)
Method to map the database column type to the Tableau SDK Type

Parameters **db_type** – types that may have length in the declaration

eg: CHAR(8) :return: Tableau SDK Type if it exists

trext.db.utils

trext.db.utils.**format_date**(*date_to_format*)
Formats the date value from database to the tableau format

Parameters **date_to_format** – date value from db

Returns formatted date

trext.db.utils.**format_datetime**(*datetime_to_format*)
Formats the datetime value from database to the tableau format

Parameters **datetime_to_format** – datetime value from db

Returns formatted datetime

trext.db.utils.**get_fake_date**()

trext.db.utils.**get_fake_datetime**()

trext.extract

The Extract package deals with creating, filling up and puslishing a .tde extract.

trext.extract.build

class `trext.extract.build.`**`ExtractBuilder`**
Bases: `object`

Builds the tableau Extract by creating a Tableau Extract, defines the table skeleton, adds the table to the extract and fills this table with the relevant data for the TDE.

`close`()

`connect_to_db`(*view_or_table_name*, *conn_string*, *dbtype=None*)
Connect to the view or table that needs to be turned into a .tde extract

Parameters

- **`view_or_table_name`** – View or Table that needs to be an extract
- **`conn_string`** – connection string to the database where the view or table exists
- **`dbtype`** – type of db so the right pyodbc wrapper is used to connect

`create_extract`()
Creates the extract by - connecting to the database, - building the tde path - initializing the extract - pulling data from db and filling the extract

Returns path to the created .tde extract

trext.extract.exceptions

exception `trext.extract.exceptions.`**`NotExtractPathError`**
Bases: `exceptions.Exception`

trext.extract.server

connection for tableau

class `trext.extract.server.`**`Tableau`**
Bases: `object`

Tableau server connection class

`close`()
Close connection to Tableau Server

`connect`(*host*, *username*, *password*, *site_content_url='Default'*)
Connect to the Tableau server

Parameters

- **`host`** – address of the Tableau server
- **`username`** – Tableau Server username
- **`password`** – Tableau Server password
- **`site_content_url`** – Site to publish to

publish(*tde_path*, *project_name='Default'*, *datasource_name=None*, *overwrite=True*)
Publishes an extract to the Tableau Server

Parameters

- **tde_path** – path of tde to publish
- **project_name** – name of project on the Tableau site to publish to
- **datasource_name** – the name of the .tde to publish as
- **overwrite** – boolean to flag if the .tde needs an overwrite when publishing

trext.extract.utils

trext.extract.utils.**get_db_components**(*db_table_or_view*)
Splits the db_table into 3 components - db, schema and table/view

Parameters **db_table_or_view** – of the form [DB].[SCHEMA].[TABLE]

Returns tuple of db, schema and table

trext.extract.utils.**get_extract_name**(*extract_path*)
Get the name of the extract in the format - name.tde

Parameters **extract_path** – Path where the extract sits

Returns

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

trext.api, 7
trext.db.conn, 8
trext.db.consume, 9
trext.db.fill, 9
trext.db.typemap, 9
trext.db.utils, 9
trext.extract.build, 10
trext.extract.exceptions, 10
trext.extract.server, 10
trext.extract.utils, 11

Index

A

AnyDB (class in trext.db.conn), 8

C

close() (trext.api.Extract method), 8
close() (trext.db.conn.AnyDB method), 8
close() (trext.extract.build.ExtractBuilder method), 10
close() (trext.extract.server.Tableau method), 10
connect() (trext.extract.server.Tableau method), 10
connect_to_db() (trext.extract.build.ExtractBuilder method), 10
create() (trext.api.Extract method), 8
create_extract() (trext.extract.build.ExtractBuilder method), 10

D

DBConsumer (class in trext.db.consume), 9

E

Extract (class in trext.api), 7
ExtractBuilder (class in trext.extract.build), 10
ExtractFiller (class in trext.db.fill), 9

F

format_date() (in module trext.db.utils), 9
format_datetime() (in module trext.db.utils), 9

G

get_cursor() (trext.db.conn.AnyDB method), 8
get_db_components() (in module trext.extract.utils), 11
get_extract_name() (in module trext.extract.utils), 11
get_fake_date() (in module trext.db.utils), 9
get_fake_datetime() (in module trext.db.utils), 9
get_table_data() (trext.db.consume.DBConsumer method), 9
get_table_definition() (trext.db.consume.DBConsumer method), 9
get_type() (in module trext.db.typemap), 9

I

insert_data_to_extract() (trext.db.fill.ExtractFiller method), 9

L

location (trext.api.Extract attribute), 8

N

NotExtractPathError, 10

P

publish() (trext.api.Extract method), 8
publish() (trext.extract.server.Tableau method), 10

T

Tableau (class in trext.extract.server), 10
trext.api (module), 7
trext.db.conn (module), 8
trext.db.consume (module), 9
trext.db.fill (module), 9
trext.db.typemap (module), 9
trext.db.utils (module), 9
trext.extract.build (module), 10
trext.extract.exceptions (module), 10
trext.extract.server (module), 10
trext.extract.utils (module), 11