**Postgres database**

**IP**: 4.36.57.13 (must be on ERG VPN)

**Port**: 5432

**Database**: UGSTank

**User**: ugst\_ergone

**Password**: Sunday1234

**Github repo**

<https://github.com/Eastern-Research-Group/UST>

Contains Python scripts that help with data processing and performing various analyses. The State Processing folder includes scripts for finding bad and duplicated data, performing QA, creating templates, exporting data for state review, exporting data to make a geocoding request (which Paul Dziemiela handles), importing geocoded data, etc., however, because the database structure was very recently changed to align with the new UST template, ***very few of these scripts will work right now***. I’ll be re-writing them as we get new data in, and feel free to contribute your own scripts.

I use an Anaconda environment that is set up with Python 3.10, but newer versions should work too. I just looked at the requirements.txt and was surprised how much was in there! I think a lot of it was installed by the arcgis library. The main libraries you will probably use are: psycopg2, pandas, and openpyxl. I’d probably just install those three libraries and then if you encounter an error when running a script because something isn’t installed, install it then.

**Database schemas**

**public** – Main data schema, containing the tables that correspond to the UST and UST Release templates, lookup tables, and control tables.

**State \_ust and \_release schemas** – These are named with the state code as a prefix and contain the raw state data, which is uploaded from whatever format the state submits it (or in whatever format we’ve downloaded it from a state website). (Note TrUSTD (the Tribal database) data is technically in an Oracle database on an EPA server that can be accessed through VDI – I exported only the relevant tables from that database into the trustd\_ust and trustd\_release schemas, so that data is a snapshot from the time I processed their data and should not be considered current (plus TrUSTD is changing the structure of their database, which is what the new UST template is based on)).

**archive** – This schema contains the db objects that used to be in the public schema prior to the recent template change/db restructure and is currently maintained for reference.

**oust** – This schema was used to perform an analysis comparing performance measure data to our data and is not used for regular UST processing.

**ust\_finder** and **ust\_finder\_prod** – These schemas were used for transforming the UST and LUST (now UST Release) data into the production UST Finder format in conjunction with the geospatial data. Not used for regular UST processing.

**Processing UST and Release data**

1. Import data into the schema. The schema name is the xx\_ust for UST data and xx\_release for UST Release (a.k.a. LUST) data, where xx is the state/organization code. You can do this manually using any database tools you prefer, or you can use a Python script. To use the Python importer to import flat file(s), save the files to a local directory and open the *UST/ust/python/state\_processing/import\_data\_from\_files.py* script. This script saves all .xlsx and .cvs files found in the directories specified to the correct state schema. The script will create the appropriate schema if it doesn’t exist and convert the name of the file into a database-friendly table name. Set the state, ust\_path, and release\_path, and overwrite\_table variables as necessary. If you are processing Release data only, set ust\_path to None, or if processing UST data only, set release\_path to None. The overwrite\_table variable defaults to False, which means if a table with the same name already exists in the database it will skip it. Set this variable to True to overwrite the existing table instead. If the state has supplied a database, import the data into the postgres database by whatever means is convenient.
2. Populate table **ust\_control**  or **release\_control** to uniquely identify the submission. You can do this manually in the database, or use script *UST/ust/python/state\_processing/insert\_control.py*, updating the variables at the top of the script as necessary. Be sure to include the source of the data.
3. As you analyze and process the data, you may come across inconsistent and/or irrelevant data in the state database. Unless there is a compelling reason to do so, do **not** alter the state’s raw data in any way. This is so we can refer to the original data easily if questions arise, and so we can redo the processing later if necessary without re-importing the data. Instead, write views or create new tables (keep them in the state schema, and prefix the names of any new tables with “erg\_” so it is obvious they are not from the state) that perform any needed data manipulation.
4. Going through the UST template field by field, examine the state data to determine how to map the state’s data to EPA’s format. You can see the field names with these queries:
   1. **Facility**:

select database\_column\_name

from ust\_elements a join ust\_elements\_tables b on a.element\_id = b.element\_id

where table\_name = 'ust\_facility' order by sort\_order;

* 1. **Tanks**:

select database\_column\_name

from ust\_elements a join ust\_elements\_tables b on a.element\_id = b.element\_id

where table\_name = 'ust\_tank' order by sort\_order;

* 1. **Compartment**:

select database\_column\_name

from ust\_elements a join ust\_elements\_tables b on a.element\_id = b.element\_id

where table\_name = 'ust\_compartment' order by sort\_order;

* 1. **Piping**:

select database\_column\_name

from ust\_elements a join ust\_elements\_tables b on a.element\_id = b.element\_id

where table\_name = 'ust\_piping' order by sort\_order;

The EPA lookup tables are in the public schema; their names are obvious. Do the best you can to map the state values to EPA values. Create a record of the mapping by populating tables **ust\_element\_mapping** and **ust\_element\_value\_mapping**. See the versions of these tables in the archive schema for examples of their use.

* 1. **ust\_element\_mapping/release\_element\_mapping**
     1. **ust\_element\_mapping\_id /release\_element\_mapping\_id** – Auto-generated PK.
     2. **ust\_control\_id/release\_control\_id** – Required, FK to **ust\_control/release\_control**
     3. **mapping\_date** – Required but defaults to today’s date if not specified.
     4. **epa\_table\_name** – Required. Table name related to the EPA template structure, e.g. ust\_facility, ust\_tank, ust\_compartment, or ust\_piping for UST or ust\_release, ust\_release\_substance, ust\_release\_source, ust\_release\_cause, or ust\_release\_corrective\_action\_strategry for Releases.
     5. **epa\_column\_name** – Required. Column name related to the EPA template structure, i.e., a column in one of the tables listed above. See information\_schema.columns where table\_schema = ‘public’ and table\_name = [epa\_table\_name] for allowed values. *For elements that have a lookup table,* ***enter the data table name and the column name from the data table*** *(e.g. ust\_compartment and substance\_id instead of substances and substance).*
     6. **organization\_table\_name** – Required. The table name in the state schema (e.g. “nc\_ust”) containing the raw state table. If the column is related to a lookup table, enter the main data table here and put the lookup table name in the organization\_join\_table column.
     7. **organization\_column\_name** – Required. The column name in the organization table above. If the column is related to a lookup table, enter the main data table column here and put the lookup column name in the organization\_join\_column column.
     8. **organization\_join\_table** – Optional. If the organization value is in a lookup table, enter the lookup table name here.
     9. **organization\_join\_column** – Optional. If organization\_join\_table is populated, this is the column name in organization\_join\_table that is used in the join to the organization\_table\_name.
     10. **programmer\_comments** - Optional. Any information you think is relevant.
     11. **organization\_comments** – Optional. Any additional information the organization provides.
     12. **deagg\_table\_name** – Optional. If the raw state data contains multiple values for a single entity in a single column (e.g. a comma-separated list of codes), we usually need to deaggregate those values and map them separately. There is a python script that will do this for you and create a “deagg” table in the state schema. Enter the name of this generated table in this column.
     13. **deagg\_column\_name** – Optional. The name of the “description” column of the generated “deagg” table described above.
  2. **ust\_element\_value\_mapping**
     1. **ust\_element\_value\_mapping\_id** – Auto-generated PK.
     2. **ust\_element\_mapping\_id** – Required. FK to **ust\_element\_mapping**. ust\_element\_mapping\_id
     3. **organization\_value** – Required. Value from the state’s data.
     4. **epa\_value** – Optional. Value from the EPA template. This can be null because you should enter ALL the organization values, even if a) you don’t know how to map them (you want the row to appear in the mapping tab of the exported template later) or b) the state value is not one that should be included in UST Finder. If the state value is one that should be excluded from UST Finder (such as a non-federally regulated substance), leave this column null, but set the exclude\_from\_query column to Y (see below). There will also be cases where the state lookup table contains a larger universe of values than the EPA element you are mapping to. For example, the state lookup table might include domain values for multiple EPA elements. If the state value from their domain table is not relevant to the EPA element being mapping, enter the state value in organization\_value, leave the epa\_value null, and leave exclude\_from\_query null (exclude\_from\_query should only be used if the tank itself should not be included in the EPA template/UST Finder).
     5. **epa\_approved** – Y/N flag indicating EPA has approved the mapping. You can leave it null until EPA has approved it.
     6. **programmer\_comments** – Optional. Any information you think is relevant. I tend to populate this with a note like “NEED MAPPING” if I’m completely unable to even guess at the mapping; that way, when the mapping data is exported to the review template, this comment will be included and it will be obvious we need input from the state or EPA. Remember to update the comments columns after review as necessary.
     7. **epa\_comments** – Optional. Any additional information from EPA. This column is usually populated during a mapping review with EPA (they’ll either write it into the template during their review period, or they may supply it verbally during a meeting and the developer should capture it and update the database).
     8. **organization\_comments** – Optional. Any additional information from the state.
     9. **exclude\_from\_query** – Optional. Y/N flag indicating that this state value is one that should be excluded from EPA’s UST template. A value of Y means the value is one that should not be included in the EPA’s UST Finder data. Examples of cases where you may set a value to Y are Substances that are not federally regulated (such as 100% biodiesel and Diesel Exhaust Fluid), or Tank Statuses such as “Never Installed”. *Set this flag to Y only if you want to exclude the related tank from UST Finder.*

***Note*** *that occasionally the state data is structured in such a way that it is difficult or impossible to create an entry for the mapping in the* ***ust\_element\_mapping*** *table* (for example, you may need to use multiple columns in the state data to map to a single EPA column). In these cases, you can write views or create new tables in the state’s schema that transform the data into EPA’s format. If you create new tables, prefix their name with “erg\_” so it is obvious which tables we created versus what was supplied by the state. You can then enter the column names from these views or tables into the ust\_element\_mapping and ust\_element\_value\_mapping tables to ensure the QA/QC and automatic template export scripts work; just be sure to enter programmer comments as necessary to describe the steps you took to build the view. The ust\_element\_mapping and ust\_element\_value\_mapping tables exist solely for our benefit as the developers, to make it easy to a) export the mapping choices for EPA review and b) maintain a record of the mapping decisions – both it’s easy to do the mapping for the same organization the next time they submit data, AND to see what mapping decisions were previously made for various state values when working on data for other organizations. Over time, mapping should go faster and faster as we build up a history of mappings. *See the* ***ust\_element\_db\_mapping*** *and* ***ust\_element\_value\_mappings*** *tables or the* ***v\_ust\_element\_mapping*** *view in the archive schema for existing mappings from the old template structure.*

1. Make sure that all Yes/No and Yes/No/Unknown columns contain valid values. These columns don’t have explicit lookup tables but must conform to the template requirements.
2. Write a query that populates table **ust\_facility** using the state data and the mapping performed in step 3. You can save this as a view in the state data schema to refer to later.
3. Write a query that populates table **ust\_tank** using the state data and the mapping performed in step 3. You can save this as a view in the state data schema to refer to later.
4. Write a query that populates table **ust\_compartment** using the state data and the mapping performed in step 3. You can save this as a view in the state data schema to refer to later.
5. Write a query that populates table **ust\_piping** using the state data and the mapping performed in step 3. You can save this as a view in the state data schema to refer to later.
6. **Dispenser data** may be at the facility, tank, or compartment level depending on the organization. The three columns that contain dispenser data are therefore found in all three tables. If the organization has dispenser data, insert it into the table that makes sense.
7. Export the new data to the template form. The simplest way to do this is by running the Python script *UST/ust/python/state\_processing/export\_template\_ust.py*. Set the input variables at the top of the script as follows; usually you can get away with passing only the ust\_control\_id.
   1. **ust\_control\_id**: *Integer*. Primary key from the ust\_control table for the data set you wish to export. If ust\_control\_id is *None* and organization\_id is passed, the script will query the database to find the most recent ust\_control\_id for the organization and will set this variable for you.
   2. **organization\_id**: *String*. Can set to *None* if you are passing an ust\_control\_id because the script will query the database and set the organization\_id for you in that case.
   3. **data\_only**: *Boolean*. Defaults to *False*. Set to *True* to export the organization’s data without the Reference, Lookup, and Mapping tabs.
   4. **template\_only**: *Boolean*. Defaults to *False*. Set to *True* to create an empty UST template with Reference, Lookup, and blank data tabs.
   5. **export\_file\_path**: *String*. Defaults to *None*. Full path including file name where you’d like the template to be exported. If *None*, and export\_file\_dir and export\_file\_name are also *None* (and template\_only is *False*), the script will set these variables for you and the template will be created in the exports directory of the repo. (The script will output the file name and path.)
   6. **export\_file\_dir**: *String*. Defaults to *None*. The directory path (without file name) you’d like the template to be exported to. Must be set in conjunction with **export\_file\_name**. If *None*, and export\_file\_name and export\_file\_path are also *None* (and template\_only is *False*), the script will set these variables for you and the template will be created in the exports directory of the repo. (The script will output the file name and path.)
   7. **export\_file\_name**: *String*. Defaults to *None*. The file name (without the directory path) you’d like the template to be exported to. Must be set in conjunction with **export\_file\_dir**. If *None*, and export\_file\_dir and export\_file\_path are also *None* (and template\_only is *False*), the script will set these variables for you and the template will be created in the exports directory of the repo. (The script will output the file name and path.)
8. Send the template to EPA for approval. (Adam or Victoria will probably do this so send it to them first.)
9. When you receive EPA’s comments on the mapping, make any changes necessary to table **ust\_element\_value\_mapping**, including updating the epa\_comments column if any are available.
10. If there is a meeting or correspondence with the state regarding the mapping, update tables **ust\_element\_mapping** and **ust\_element\_value\_mapping** as necessary, including the organization\_comments column if appropriate.
11. Geocoding – coming soon
    1. The next step will be to export the locational data to a specially-formatted spreadsheet and ask Paul to perform the geocoding, if necessary. I haven’t had time to update this part yet.