Guidance on Potential Tool Updates

The software delivered to CARB can be accessed through the `ERG` branch of the `CARB-Facility_Matching-DesktopGUI` GitHub repository. ERG is aware that updates will be needed to this tool in the future. This document serves as a guide for CARB analysts and developers to makes these updates. Please email tyler.richman@erg.com if you have any questions or would like to contribute to the GitHub repository.

Incorporating an Alternative Facilities Database

The software deliverable from ERG to CARB incorporates a desktop database file containing the "test" golden master list of facilities into the tool. The database connection is established within the tool using the SQLite3 Python library. All columns of the "test" golden master database table are read into Python as a Pandas data frame and subsequently condensed and processed for facility matching.

Developers must do the following to update the database used in the tool:

1) Edit line **9** in `setup.py` to point to the new pathway or URL of the new database.

```
self.facility_database_loc = ""
```

2) Accessing the new database in Python may require a library other than SQLite3. It may not be included within the Python installation. If this is the case, copy and paste line **38** of `setup.py` into a new line.

```
subprocess.run([activate_venv_command, "&&", python_exe_location,
      pip_exe_location, "install", "pyinstaller==6.6.0"], check=True)
```

   In this **new line**, rename "`pyinstaller`" to the new library name and rename "`6.6.0`" to the version of the new library that you would like to install for the tool.

3) Open `config.ini` and replace the `Master_Facilities_Table_Name` variable with the name of the golden master facilities table in the new database.

```
Master_Facilities_Table_Name = Facilities
```

4) Update lines **212-214** in `algorithm.py` to establish a connection with the new database and convert the master facility table into a Pandas data frame.

```
conn = sqlite3.connect(db_loc)
df_master = pd.read_sql_query(f"SELECT * FROM {table_name}", conn)
conn.close()
```

5) Re-run `setup.py` to incorporate the changes into the executable file.

Adding Fields to Matching Algorithm

The software delivered from ERG to CARB conducts facility matching from 12 unique facility data elements: CO, AB, DIS, FACID, Name, Street, City, ZIP Code, SIC, NAICS, Latitude, and Longitude. These data elements make up or derive table fields used in the matching algorithm. The algorithm attempts a series of matches between the input table and master facilities table on a combination of the table fields. A match score indicates which fields and the level of stringiness involved in the match. For example, a match score of 1 matches the input table to the master facilities table on the "CO", "AB", "DIS", "FACID", "FNAME_STANDARDIZED", "FSTREET_STANDARDIZED", "FCITY", "FZIP", "FSIC", "LATITUDE_ROUND_5", "LONGITUDE_ROUND_5" fields. Whereas a match score of 30 matches the input table to the golden master facilities table on just "CO", "AB", "DIS", and "FACID".

Developers must do the following to incorporate a new data element into the tool:

1) Open `config.ini`. Lines **3**-**15** contain a list of field names within the master facilities database table.

```
[Database]
Master_Facilities_Table_Name = Facilities
ARBID_name = ARBID
CO_name = CO
AB_name = AB
DIS_name = DIS
FACID_name = FACID
FNAME_name = FNAME
FSTREET_name = FSTREET
FCITY_name = FCITY
FZIP_name = FZIP
FSIC_name = FSIC
FNAICS_name = FNAICS
LAT_name = LATITUDE
LON_name = LONGITUDE
```

   Add a new line to this with the new data element that follows the existing format used here.

2) Add a dropdown within the GUI to allow the tool user to configure the new field name with the input table. The code for this is found on lines **458**-**565** of `main.py`. Copy and paste an existing field configuration code to a new line and alter so that it is compatible with the new data element. Below is an example of existing field configuration code for a single data element.

```
## CO -- Create field configuration
selected_option_CO = tk.StringVar(field_selection_frame)
selected_option_CO.set(options[0])
label_CO = tk.Label(field_selection_frame, text="CO")
label_CO.config(font=(text_font, 10, "bold"))
dropdown_CO = tk.OptionMenu(field_selection_frame, selected_option_CO,
      *field_options)
dropdown_CO.config(width=50)
```

3) Add drop down options within the GUI to allow the user to select fields from the input table for field configuration. The code for this is found in lines **89-193** of `main.py`. There are three separate parts within this code that need to be updated. The first sets the initial field configuration option to `''`. Below is an existing example of this.

```
selected_option_CO.set('')
dropdown_CO['menu'].delete(0, 'end')
```

Copy and paste this onto a new line and alter the variable names so that it is compatible with the new data element. The second part adds the input table field names to the dropdown options. Below is an existing example of this.

```
dropdown_CO['menu'].add_command(label=choice,
      command=tk._setit(selected_option_CO, choice))
```

Copy and paste this onto a new line and alter the variable names so that it is compatible with the new data element. The third part implements these changes into the GUI. Below is an example of this.

```
label_CO.grid(row=1, column=0, padx=5, sticky='w')
dropdown_CO.grid(row=1, column=1)
```

Copy and paste this onto a new line and alter the variable names so that it is compatible with the new data element.

4) Update the `standardize_table` function within `algorithm.py` (starts on line **9**) to incorporate the new data element. This includes:
   - Adding a new input parameter to the function.
   - Adding the input parameter variable to the list of kept fields within the data frame (line **35**).
   - Adding a new item to the `dtype_mapping` dictionary.

5) Add a new input parameter to the `standardize_table` function within `main.py` (starts on line **229**) to incorporate the new data element. This will call upon the field name entered in by the user during the field configuration.

6) Update the `read_in_master_table` function within `algorithm.py` (starts on line **182**) to incorporate the new data element. This includes:
    - Adding a new input parameter to the function.
    - Adding a new item to the `col_rename_dict` dictionary.
    - Adding the input parameter variable to the list of kept fields within the data frame (line **233**).
    - Adding a new item to the `dtype_mapping` dictionary.

7) Add a new input parameter to the `read_in_master_table` function within `main.py` (starts on line **247**) to incorporate the new data element. This will call upon the field name used in `config.ini`.

8) Open `match_score_fields.json`. This contains a list of fields associated with each match score and is used directly within the tool for matching.


```
{
    "1": ["CO", "AB", "DIS", "FACID", "FNAME_STANDARDIZED", "FSTREET_STANDARDIZED", "FCITY", "FZIP", "FSIC", "LATITUDE_ROUND_5", "LONGITUDE_ROUND_5"],
    "2": ["CO", "AB", "DIS", "FACID", "FNAME_STANDARDIZED", "FSTREET_STANDARDIZED", "FCITY", "FZIP", "LATITUDE_ROUND_5", "LONGITUDE_ROUND_5"],
    "3": ["CO", "AB", "DIS", "FNAME_STANDARDIZED", "FSTREET_STANDARDIZED", "FCITY", "FZIP", "FSIC", "LATITUDE_ROUND_5", "LONGITUDE_ROUND_5"],
    "4": ["CO", "AB", "DIS", "FACID", "FNAME_STANDARDIZED", "FSTREET_STANDARDIZED", "FCITY", "FZIP", "FSIC", "Parcel_UID"],
    "5": ["CO", "AB", "DIS", "FACID", "FNAME_STANDARDIZED", "FSTREET_STANDARDIZED", "FCITY", "FZIP", "Parcel_UID"],
    "6": ["CO", "AB", "DIS", "FNAME_STANDARDIZED", "FSTREET_STANDARDIZED", "FCITY", "FZIP", "LATITUDE_ROUND_5", "LONGITUDE_ROUND_5"],
    "7": ["CO", "AB", "DIS", "FSTREET_STANDARDIZED", "FCITY", "FZIP", "FSIC", "LATITUDE_ROUND_5", "LONGITUDE_ROUND_5"],
    "8": ["CO", "AB", "DIS", "FSTREET_STANDARDIZED", "FCITY", "FZIP", "FSIC", "Parcel_UID"],
    "9": ["CO", "AB", "DIS", "FSTREET_STANDARDIZED", "FCITY", "FZIP", "Parcel_UID"],
    "20": ["CO", "AB", "DIS", "FNAME_STANDARDIZED", "FSIC", "FNAICS", "LATITUDE_ROUND_5", "LONGITUDE_ROUND_5"],
    "21": ["CO", "AB", "DIS", "FNAME_STANDARDIZED", "FSIC", "FNAICS", "Parcel_UID"],
    "30": ["CO", "AB", "DIS", "FACID"]
}
```


Add the field name of the data element to any of the existing match scores or you can create an entirely new match score with a unique list of data elements.

9) Update the Pandas data frame created in lines **302**-**318** of main.py. This data frame is implemented into the output table and informs users whether a particular data element was a match, a potential mismatch, or not considered at all within the matching algorithm. It should reflect the contents of `match_score_fields.json`.

10) Re-run setup.py to incorporate the changes into the executable file.

Updating Parcel Dataset

The software delivered from ERG to CARB incorporates parcel data for California which acts as a broader spatial classification than the coordinates rounded five decimal places. The parcel dataset was received by CARB as a GeoDatabase containing a feature class with close to 13 million parcels. Processing this into a GeoDataframe as very time consuming. The following steps were taken to speed up this process.

1) Remove all fields within the feature class. This greatly reduces the overall size of the dataset.
2) Project to NAD83. This is the coordinate reference system used in the master facilities table.

3) Add a unique identifier to each parcel.
4) Calculate the area of each parcel. This is done because some parcels overlap. The code combats this by assigning each facility to the largest parcel that it is located within. Any tiebreaks in area goes the parcel with the highest unique identifier.
5) The parcels dataset is then written into a parquet file. This format allows the parcels to be read into Python extremely quickly in comparison to GeoDatabase format.

All these steps where automated within an ArcGIS Python Toolbox called `CARB Facility Matching Support.pyt`. Users can execute this workflow by running the `Parcel Feature Class to Parquet` tool within this toolbox. This tool intakes a feature class of parcel data as well as a folder location in which the output `Parcel.pqt` file will be written to. Users then must list the `Parcel.pqt` file location in line **8** of `setup.py`. Users must also re-run `setup.py` to incorporate the changes into the executable file.