# The Problem

There are several companies that collect the donations that are received by Daniel’s Table (DT) and each company has a different way to inform DT about donations that are received. One system sends an email when a donation is given and requires the information to be downloaded from a secure site; others allow reports to be created from their web interfaces (GUIs). DT wants to record all the donations and donor info into their donor management system, Little Green Light (LGL).

LGL has a well written flexible installer that allows a user to manually map columns in a spreadsheet or CSV file to fields that it understands. It also has an excellent REST interface that allows donor data to be retrieved and written to the system. Today, a volunteer updates LGL based on the various input she receives from the donation sources. In addition to updating donation information, the volunteer also maintains the donor information such as the donor’s name, phone number(s), physical address, and email address.

This is a very time-consuming task and it is easy to make mistakes. In addition, the manual process does not guarantee that all sources will be handled consistently and it is difficult to train another person to assist or take over the task.

## Requirements

### General Requirements

* Enable data from different sources to be consolidated into a single CSV file where the columns to be imported names match the LGL data dictionary names.
* The program should allow the user to specify only the input file(s) they wish to import
* The program should allow the user to specify the name of the output file.
* Add a column for the LGL constituent ID. The ID must be looked up in LGL based on the name field(s) of the input data.
* Verify the mailing addresses in the input data match the mailing addresses in LGL if available.
* Verify the email addresses in the input data match the email addresses in LGL if available.
* Directly import donation data into LGL. New constituent accounts must be created before donations can be added to LGL.
  + The user must be able to validate that the information is correct before it is imported.
* Directly export donation data from sources for importing when possible.

### Stripe Requirements

Stripe is the backend collection system for most of the websites where donors make an on-line donation using a credit card. Its input file is created by exporting data for a time period in its web interface.

* Make sure only successful gifts are included. If a gift failed or was refunded, do not include that row.
* Other columns not needed: A, B (see below), C (see below), D, G, H, I , K, L, N, O, R, S, T, U, V, W, X, Y, Z, AA, AB, AC, AD, AE, AF, AI, AJ, AK, AL, AM,AN, AO, AP, AQ, AS, AT, AU, AV, AW, AX, AY, BA, BB, BC, BD, BE, BF, BH, BI, BJ, BK, BL, BM
  + B unless it says, “In Memory of”, “In Honor of” or “RoundUp: with person’s name. If it says “RoundUp”, the name must be carried over to the First Name and Last Name columns.
  + C unless it says anything other than “Payment Complete”.
* Columns to Keep: “Created (Gift Date), Amount (Gift Amount), Customer Email, First Name, Last Name, Campaign Name, Description (Gift Note).
  + Clear the description of these words, “Give Lively/Smart Donations” and “Invoice ###”.
* When a name is given, the first and last name can usually be found in the “Customer Description” column. If that column is blank, try the “user\_first\_name (metadata)” and “user\_last\_name (metadata)” columns.
* If the Campaign column is blank, compare amount and date to 1 month prior. If both amount and date are the same, the campaign should be “General ~ Recurring Stripe”. All others should be “General”.
* Columns to be imported to LGL: “LGL Constituent ID”, “Gift Date”, “Gift Amount”, “Gift Campaign”, “Gift Note”, and “External ID”.
  + LGL defaults: Gift Type = “Gift”, Category = “Donation”, Payment Type = “Credit Card Stripe”, Acknowledgement = “—do not acknowledge via LGL –“

### QB Requirements

QuickBooks is used by Daniel’s Table to track donations made directly by cash or check. Its input file is created by exporting data for a time period from its web interface.

* Note the beginning and end dates.
* Carry the date of deposit into all rows where there is a donor name or vendor for that deposit. Copy vendor names into the Donor column.
* Delete all rows with a C in the “Clr” column. Ignore the “Clr” column.
* Delete all rows with “Deposit” in the Transaction Type column. Delete all blank rows. Ignore the “Transaction Type” column.
* Ignore the Vendor column.
* Delete all rows that say, “External Deposit” or “Credit Interest” in the “Memo/Description” column.
* Clear the “Memo/Description” column of the single word, “donation”.
* Add a column for “Campaign”. Use anything still written in the “Memo/Description” column to map to the correct campaign description in LGL.
* QB Column mappings:
  + LGL Account # to LGL Constituent ID#
  + Date to Gift date
  + Num to Check/Reference No.
  + Memo/Description to Gift note
  + Amount to Gift amount
  + Campaign to Campaign
* Defaults:
  + Gift type to “Gift”
  + Category to “Donation”
  + Payment type to “Check”
  + Acknowledgment to “General Thank You Letter”

# The Solution

## Summary

A program will be built so that each source of data can be imported and transformed into a format consistent with LGL’s columns that will be in CSV format. All of the input files can be loaded into a single CSV file that can be directly imported into LGL. It is worth noting that each data source may have info that isn’t in other data sources. For instance, some data sources may have a full name in a single column, while others give first and last name in separate columns. When data isn’t available for a source, that column will be blank unless there is a default.

If a donor name does not match any LGL Constituent name (or spouse name), the name will be printed to the screen and the output file will not have a Constituent ID for that name.

In addition to consolidating the donation data into a consistent format and CSV file, additional checks will be made to verify that the physical and email addresses of each constituent is the same as the data in LGL. If an inconsistency is found, a second file will be created with the LGL Constituent ID, Constituent Name, and the changed address info from the input file. It will not contain the data from LGL.

### Phases

The code will be rolled out in phases as there is a lot of work and it would be good to know if some of the early work is operating correctly.

|  |  |
| --- | --- |
| **Phase** | **Delivery** |
| 1 | The code will import donor data and produce a consolidated CSV file to be imported based on one or more input files. |
|  |  |
| 2 | Checks for consistent email and physical address will be added. |
| 3 | in Stripe (donors who gave the same amount on the same day in the prior month)update the Campaign column with the comment, “General~Recurring Stripe”. |
| 4 | Enable the code to directly create new users in LGL. The user must be able to verify the data is correct before the import. This may require building a GUI to allow the user to verify the information before importing. |
| 5 | Enable the code to directly import donations to LGL. The user must be able to verify the data is correct before the import. This may require building a GUI to allow the user to verify the information before importing. |
| 6 | Start trying to collect information directly from the sources so the various input spreadsheets and CSV files don’t have to be created. This may not be possible for all sources. |

### User Interface

The planned user interface (UI) is code that will run at command line. It will be fairly simple to run with only a few options. It will be delivered as a single executable file, so it will be easy to load anyplace and run.

## Design Info

Below is a class diagram that basically describes the design of the modules. The main module is donor\_etl.py. It is invoked when the code is run and uses the DonorFileReaderFactory to read and choose the correct DonorFileReader class for each input file. It then runs the “map\_fields” method in DonorFileReader to correctly map the data in the files to the LGL columns. The DonorFileReader classes also use the ConstituentDataValidator class to compare the physical and email addresses from the input files to what is on record in LGL and record any variances in a separate file.

To make processing consistent, when the data is read from an input file, it is put into a standard format by the *initialize\_donor\_data* method of the DonorFileReader subclasses. That allows the *map\_data* method to be implemented in the DonorFileReader class instead of having to be rewritten for each subclass. The *get\_lgl\_constituent\_id* class needs to be implemented in the subclasses because each data set stores the donor name in different ways.

The comments in the code contain the formats of the various data objects as well as a description of the flow of the code. It is easier to maintain that information there so that it can be kept up to date as it changes.

There is a sample\_data.py module that is not included in the diagram as well. This is a module that contains some test data for the functional tests in the classes. This module was developed well after this project had started and is used inconsistently. Eventually, this will be corrected.

If you create a virtual environment, you should be able to install the requirements and run against the sample data.

The user can interact with the program in one of two ways. The user can execute the program from command line and use arguments specifying the name of the input file, output file, and variance file. The user can also specify no arguments and a GUI will be started that allows the user to give that information.

The entire GUI is in the donor\_gui (DonorGui class) module. It uses PySimpleGui as the underlying technology to create and manage the GUI. The GUI has two forms. The initial form collect the input from the user (input files, output file name, variance file name). As the program executes, various messages (errors and other warnings) are generated. These messages are saved by the DisplayData module (more on that in a moment) and displayed at the end in a dialog. The user can copy the messages if they want to save them someplace or can just dismiss the box.

The DisplayData is a singleton class that is used by modules that want to contribute messages to be displayed at the end of the program in the final dialog. It is basically a decorator for the Python logging functions. When you want to log a message to be saved, you simply wrap it in the DisplayData class. There are further notes and an example of this in the display\_data.py module.