CIS 422 Project 2: *Freedge Tracker Prototype*
Programmer Documentation

Ginni Gallagher (gmg), Ellie Kobak (erk), Kalyn Koyanagi (kek),
Liza Richards (ljr), Madison Werries (mgw)
03-06-2022 - v2.01

Table of Contents

1. Programmer Documentation Revision History

Date	Author	Description
2-28-2022	ljr	Created the initial document and wrote first draft
3-01-2022	ljr	Updated sections 3 and 4.
3-05-2022	mgw	Proofread and reformatted document
3-06	mgw	Updated section 3.6.1 - Status Class

2. Introduction

This document details the components of the Freedge Tracker System Prototype in terms of its source code, including important classes and their member variables and methods. The documentation is based on the structure and content of the EyeDraw v3.0 Programmer's Documentation [1] and World Tax Planner Programmer's Documentation [2] examples provided in the CIS 422 course, as taught by Professor Anthony Hornof at the University of Oregon.

All code is developed using Pycharm, Sublime, and Vim, and is written to be runnable in Python 3.7 through 3.9. This version of the software was written for the 03/06/2022 release of the Freedge Tracker System Prototype.

This documentation assumes a strong grasp of the Python programming language in a MacOS environment.

3. Program Files

The following details each individual Python file and their classes, methods, and variables.

3.1. freedge_tracker.py

The freedge_tracker.py contains the main() function used to launch the Freedge Tracker system. Interacts with administrator_interface.py to create and run the program interface.

3.1(a) main()

The main function used to launch the Freedge Tracker System.

3.2 administrator_interface.py

This file contains the core driver of the Freedge Tracker System, as it is responsible for managing the system's GUI as well as the I/O between the administrator (the user) and the system. These I/O interactions allow the user to create, load, and interact with instances of the FreedgeDatabase class using buttons and interactable data tables on the screen.

3.2.1 AdministratorInterface — Overview:

A class responsible for maintaining the data of the currently running system, including the GUI elements, and the path to the currently open FreedgeDatabase.

3.2.2 AdministratorInterface — Attributes:

3.2.2(a) fdb_path → str

Used for storing the file path location of the currently loaded database file for later use.

3.2.2(b) fdb → FreedgeDatabase

Used for storing the instance of the FreedgeDatabase class which is currently loaded into the system.

3.2.2(c) root → Tk

The root of the tkinter interface. Used for any changes which need to be made to the GUI while the system is running.

3.2.2(d) notebook → Notebook

This attribute is the tkinter Notebook used to access and modify the two table display tabs any time the data tables need to be updated.

3.2.2(e) main_tab → Frame

Used for updating the display of the main tab which shows all the freedges currently loaded into the database.

3.2.2(f) ood_tab → Frame

Used for updating the display of the secondary tab which shows all the freedges currently loaded into the database whose statuses are considered to be "out-of-date."

3.2.2(g) info_label → Label

Used to update the information displayed in the box on the right hand side of the screen which shows additional information about the freedge the user has selected in the table.

3.2.2(h) ib_width → int

Used to maintain consistency across the width of the info box's labels as they are created, modified, and destroyed.

3.2.2(i) prompt_label → Label

Used to prompt the user with instructions when a database hasn't been loaded into the system yet, destroying it once a database has been loaded.

3.2.3 AdministratorInterface — Methods:

3.2.3(a) __init__()

Initializes a new instance of the AdministratorInterface class.

3.2.3(b) CreateDatabase(self)

Creates a new internal freedge database. Both the csv file to load and the location/name of the new db file to save to are chosen by the user via popup prompts. Calls SelectCSVFile(), SelectDBSave(), and OpenDatabase().

3.2.3(c) LoadDatabase(self, db_file_path)

Opens the database at the given path, loading the database into the system and updating the internal file which stores the most recently accessed database file. Called by SelectDatabaseToLoad() and the main driver module, freedge_tracker.py.

3.2.3(d) SelectCSV(self)

Prompts the user to choose a csv file to load during the creation of a new database. Returns the chosen path if one is selected successfully, or None otherwise. Called by CreateDatabase().

3.2.3(e) SelectDBSave(self)

Prompts the user to choose the name and location of the db file to save their newly created database to. Returns the chosen path if one is selected successfully, or None otherwise. Called by CreateDatabase().

3.2.3(f) SelectAndOpenDB(self)

Prompts the user to choose a preexisting SQL database file (.db) to load into the system, updating the display with the information in the chosen db file. Calls LoadDatabase() and UpdateFullDisplay(). Called by pressing the tkinter button labeled 'Load Database'.

3.2.3(g) UpdateDatabase(self)

Updates the currently loaded database file (db) with information from a new csv file that the user selects. Information about freedges which will be added to, removed from, and modified within the database is displayed to the user, allowing them to confirm or cancel the update.

3.2.3(h) GetSelected(self)

Returns the Freedge object corresponding to the user's currently selected entry in the display table, if any. If nothing is selected, the function returns None.

3.2.3(i) OnTableClick(self)

Called when the user clicks on an entry in the GUI table listing the freedge data entries. Calls GetSelected(), updating the info_box with the newly selected data, if any.

3.2.3(j) NotifyFreedge(self, freedge)

Sends a notification to the caretaker of the passed-in Freedge object.

3.2.3(k) NotifySelected(self)

Sends a notification to the caretaker of the freedge that is currently selected by the user in the display table, if any. Called when the user clicks on the GUI button 'Notify Selected'.

3.2.3(l) NotifyOutOfDate(self)

Sends a notification to each of the caretakers who have not been contacted in the amount of time defined by FIRST_UPDATE_THRESHOLD in the constants file (InternalData/database_constants.py). The system first prompts the user with information about the freedges which will be messaged, allowing them to confirm or cancel the mass-send.

3.2.3(m) UpdateFullDisplay(self)

Updates the GUI display to reflect the currently loaded database information. This is called on system startup, anytime a notification reply is received from a caretaker, as well as any time a new database is created/loaded.

3.2.3(n) UpdateTableDisplay(self, table, freedges: [Freedge])

Updates the information in a particular display table to contain the information stored in the array of Freedge objects passed in. This avoids duplicate code when updating the table tabs ('All', the main tab, and 'Out of Date', the tab showing only the out of date freedges).

3.2.3(o) BuildTable(self, tab)

Constructs the GUI elements necessary for a display table, using the passed-in tab argument as the parent object.

3.2.3(p) AddButton(self, parent, label, cmd, xpos, ypos)

Creates a new button on the GUI display. Sets the parent, the text label, the command (function) to call when pressing the button, and the position of the button based on the given parameters.

3.2.3(q) CreateDisplay(self)

Builds the full tkinter GUI display of the Administrator Interface.

3.2.3(r) exit_

Exits the program.

3.3 tkinter_style.py

A file containing the Tkinter style to be used in the system's GUI display, contained as a separate file for modularity and to maintain the readability of administrator_interface.py.

3.3(a) main():

This function starts the program, creating a new instance of AdministratorInterface to manage the interactions between the system and the user.

3.4 caretaker_info_parser.py

This module is responsible for reading in data from a csv file and parsing it for use by freedge_database.py.

3.4(a) remove_whitespace(field_name)

This function is used to remove any whitespace from a given field name.

3.4(b) _get_headers()

A private function with a purpose of obtaining all the column titles from the csv file.

3.4(c) _get_address_format ()

A private function with a purpose of formatting each freedge address from the imported csv file using constants from database_constants.py.

3.4(d) _parse_field(data)

A private function with a purpose of checking that a data field is not empty.

3.4(e) _parse_address(data)

A private function with a purpose of generating a list which has all of the address data for each freedge.

3.4(f). _parse_row(data)

A private function with a purpose of generating a list which has all of the general data for each freedge that is from a row in the csv file.

3.4(g) parse_freedge_data_file(filename)

A private function with a purpose of reading in information from a csv file.

3.5 freedge_database.py

This module is responsible for all of the functionality of any databases created, maintained, and used by the Freedge Tracker System. The module uses SQLite to store the system's freedge data. The database is constructed using the freedge data read from a csv file that has been parsed by caretaker_info_parser.

3.5.1 class FreedgeDatabase — Overview:

The file freedge_database.py contains a class, FreedgeDatabase, which has the purpose of creating an SQLite database connection so that database operations can be performed in order to create the database, update objects in the database, and return database objects in usable forms so that they may be used by other components in the system.

3.5.2 class FreedgeDatabase — Attributes:

3.5.2(a) db_location → str

Used to keep track of the file path to where the database is being stored.

3.5.3 class FreedgeDatabase — Methods:

3.5.3(a) __init__(self, db_location)

Creates a new instance of the class FreedgeDatabase, saving the location of the new database to the passed-in string *db_location*.

3.5.3(b) open_connection(db_location)

A function with the purpose of establishing the connection of the database.

3.5.3(c) create_table(conn, create_table_sql)

This function creates a new SQL table within the database in the open connection using the sql string passed into the function.

3.5.3(d) new_address(conn, address_data, temp)

This function inserts a new address into the SQL table containing the addresses. The variable temp determines whether or not to store the address in the main address table or the temporary address table, which is used only during the comparison of two databases.

3.5.3(e) new_freedge(conn, freedge_data, temp)

This function inserts a new freedge entry into the SQL database, using the list of fields passed in by the array of data elements, *freedge_data*. Same as *new_freedge()*, this function accepts an optional boolean flag, *temp*, denoting whether to save the new freedge into the main freedges SQL table or into the temporary table, used only for the comparison of two databases.

3.5.3(f) row_to_freedge(row)

This function converts an SQL query row into an instance of the Freedge class, which it then returns.

3.5.3(g) query_to_freedgelist(rows)

This function converts all the retrieved rows from an SQL query into a list of Freedge class objects, returning the created list.

3.5.3(h) get_freedges()

A function with the purpose of obtaining and returning a list of all the freedge objects in the database.

3.5.3(i) get_out_of_date()

A function with the purpose of obtaining and returning a list of all the freedge objects that are considered to be out of date.

3.5.3(j) update_freedge(f)

A function with the purpose of updating the freedge information in the database when changes are made to the freedge objects.

3.5.3(k) compare_databases(new_csv_data)

A function with the purpose of returning a tuple of all the freedges whose data has changed from the original upload.

3.5.3(l) exists_internal_database(db_path)

A function with the purpose of returning if a database exists given an inputted path or not.

3.5.3(m) load_internal_database(db_path)

A function with the purpose of loading in and returning a database at the inputted path.

3.5.3(n) new_database_from_csv(db_path, csv_file_path)

A function with the purpose of creating a new database at the inputted path from an inputted csv file.

3.6 freedge_data_entry.py:

The component responsible for retrieving all of the freedge information from the SQLite database so that it may be easily interacted with by other components.

3.6.1 class Status(Enum):

A class which inherits from the built-in Python class Enum, created with the purpose of defining fixed values for the possible status of a freedge.

Currently, the defined Status options are:

Status.Active
Status.Unknown
Status.SuspectedInactive
Status.ConfirmedInactive

3.6.1(a) Status.Active

Status.Active is used when the status of a freedge is *known* to be active. This means the freedge's status has been updated within the last {FIRST_UPDATE_THRESHOLD} days by the Freedge Tracker System or by an administrator.

3.6.1(a) Status.Unknown

Status.Unknown is assigned in the case when both the {ACTIVE_STATUS_KEY?} column and the {PERMISSION_TO_CONTACT_KEY} column have been left blank in the CSV file which is loaded in during the creation of a new database. It differs from 'SuspectedInactive' in that a freedge's status cannot reasonably be defined as 'SuspectedInactive' if a caretaker has not given consent to receive notifications, as a failure to respond to messages should not be regarded as the same as no input at all.

3.6.1(a) Status.SuspectedInactive

Status.SuspectedActive is used to indicate to the administrator user that the status of a freedge may need to be regarded as inactive. This happens when the freedge's status has not been updated within the last {FIRST_UPDATE_THRESHOLD} days, and the user has neither confirmed nor denied that their freedge is inactive.

3.6.1(a) Status.ConfirmedInactive

Status.ConfirmedInactive indicates that the status of the freedge is known to be inactive. This means that the freedge caretaker at some point notified the system that their freedge is inactive.

3.6.2 class ContactMethod(Enum):

A class which inherits from the built-in Python class Enum, created with the purpose of defining fixed values for the possible contact methods available to get in touch with a Freedge caretaker.

Currently, the defined ContactMethod options are:

ContactMethod.SMS
ContactMethod.Email

the Enum values of which correspond to the string values set by the user in the constants file, database_constants.py

3.6.3 class Freedge — Overview:

The class which is responsible for keeping track of the data for a particular freedge, including the caretaker's contact information, the id corresponding to the entry in the FreedgeDatabase, and how long it has been since we have updated the status of this particular freedge.

3.6.4 Freedge — Attributes:

3.6.4(a) freedge_id → int:

Used to keep track of the corresponding entry in the SQLite database so that the database may be updated as needed using the data stored in the Freedge object.

3.6.4(b) project_name → str

The name of the project (considered to be the name of the freedge itself by the Freedge organization and the caretakers).

3.6.4(c) network_name → str

The name of the network that the freedge was constructed under, if any.

3.6.4(d) caretaker_name → str

The name of the caretaker who has registered their freedge with the organization.

3.6.4(e) fridge_location → FreedgeAddress

Used to keep track of the address of the fridge itself.

3.6.4(f) date_installed → datetime.date

Used to keep track of when the fridge was installed. Stored as a Python date to allow for easier calculation of the number of days between this and other dates.

3.6.4(g) permission_to_notify → bool

Used to keep track of whether or not the registered caretaker of the freedge has given permission to be contacted/prompted/notified by the system to request updates about the status of their freedge.

3.6.4(h) preferred_contact_method → ContactMethod

The freedge caretaker's preferred method of being contacted, if any. Used to decide whether to send out an SMS text message or an email when the administrator requests to notify the given caretaker.

3.6.4(i) phone_number → str

Used to keep track of the freedge caretaker's phone number.

3.6.4(j) email_address → str

Used to keep track of the freedge caretaker's email address.

3.6.4(k) freedge_status → Status

Keeps track of the current status of the freedge, whether that be Active, SuspectedInactive, or ConfirmedInactive.

3.6.4(l) last_status_update → datetime.date

This keeps track of the last time the freedge's status was updated within the database, stored as a Python date to allow for easier calculation of how long it has been since the last update.

3.6.5 Freedge — Methods:

3.6.5(a) can_notify(self)

A function that returns a boolean regarding whether or not the freedge owner is allowed to be sent notifications

3.6.5(b) set_permission_to_notify(self)

A function that allows the caretaker to update their permission to notify status.

3.6.5(c) set_preferred_contact_method(self)

A function that returns which contact method the caretaker prefers to be notified through.

3.6.5(d) time_since_last_update(self)

A function with the purpose of returning the number of days that have passed since the last time the freedge activity status was updated.

3.6.5(e) update_status(self)

A function with the purpose of updating an individual freedge object's activity status.

3.6.5(f) reset_last_update(self)

A function with the purpose of changing the time since the last update to be the current date.

3.6.5(g) comparison_string(self, field_name, old, new)

A function with the purpose of comparing a specified freedge field between the old and new freedge data.

3.6.5(h) compare_freedges(self, freedge)

A function with the purpose of comparing the fields between two freedge objects and returning which fields differ between them.

3.6.5(i) ToString(self, freedge)

A function with the purpose of converting freedge data into a string containing all of the freedge data.

3.6.6 class FreedgeAddress — Overview

A class with the purpose of storing the address of a freedge. It is initialized with the freedge's street address, city, state, zip code, and country.

3.6.7 class FreedgeAddress — Attributes:

3.6.7(a) street_address → str

Stores the street address of the given FreedgeAddress.

3.6.7(b) self.city → str

Used to keep track of the city a Freedge is located in.

3.6.7(c) self.state_province → str

Used to keep track of the state or province a Freedge is located in.

3.6.7(d) self.zip_code → str

Used to keep track of the zip code where a Freedge is located.

3.6.7(e) self.country → str

Used to keep track of the country a Freedge is located in.

3.6.8 class FreedgeAddress — Methods:

3.6.8(a) ShortString()

A function with the purpose of returning a shorter version of the freedge's address

3.6.8(b) ToString()

A function with the purpose of changing the street address to be in a single string.

3.6.8(c) ToDisplayString()

A function with the purpose of returning a full string of the freedge's address information in the desired display format.

3.7 database_constants.py:

This file contains all of the constants that are used in the management of the Freedge Tracker system. These constants are able to be edited by the user to suit their needs.

This includes managing all the necessary constants that will be used by many files like the time limits between updates, the path to the database and the path to the csv. It also includes the keys

for the project name, network name, street address, city, state, zip code, country, installation date, contact name, phone number, email, permission to contact, preferred contact method, and activity status. Lastly, it has the strings for when the user specifies if they prefer to be contacted by text or email.

3.8 notificationMgmt.py

The component responsible for sending messages to and receiving messages from the caretaker's whose fridge's are out of date, communicating with administrator_interface.py to update data according to the caretaker's replies.

3.8.1 class NotificationMgmt — Overview

A class with the purpose of managing the notification system.

3.8.2 class NotificationMgmt — Attributes

3.8.2(a) root→ Tk

The tkinter root for the GUI display. Used to maintain consistency between the main GUI in administrator_interface.py and notificationGUI.py, as there can only be one instance of Tk() in-use at a time.

3.8.3 class NotificationMgmt — Methods

3.8.3(a) get_fridge_info_message()

A function with the purpose of obtaining all of the freedges that are out of date. It then collects the associated freedge information to be used in the notification message for the user to view.

3.8.3(b) notify_and_update(freedge_database, freedge)

A function with the purpose of calling the notification GUI and obtaining a user response that will then be used to update the freedge database.

3.9 notificationGUI.py

This file is responsible for the GUI of the notification system. It is a prototype component, taking the place of a real SMS or email message. It interacts with notificationMgmt.py to obtain the fridge caretaker's information and prompts for a response from the "caretaker" regarding the status of their freedge.

3.9.1 class PopUp — Overview

A class with the purpose of managing the notification pop up window. This window will display the notification message sent to each fridge caretaker, and will have two buttons to select whether or not the community fridge is still active. There will also be an exit button to exit the window.

3.9.2 class PopUp — Attributes

3.9.2(a) root→ Tk

The tkinter root for the GUI display. Used to maintain consistency between the main GUI in administrator_interface.py and notificationGUI.py, as there can only be one instance of Tk() in-use at a time.

3.9.2(b) response_received → BooleanVar

Used to track whether or not the user has responded to the popup. When the popup is created by a NotificationMgmt object, it is set as a wait_variable (a function built-into Tkinter), meaning that the system will only proceed once the value of said variable has changed. This allows us to pause the system when the popup appears, rather than continuing in the background, allowing us to show only one popup at a time and to parse their responses individually.

3.9.2(c) response_value → Status

This is the recorded response from the user. It is updated when they click on one of the bubble options on the popup window, or if the user force-closes the window by hitting 'X', in which case it is set to None.

3.9.2(d) pop_up_win → tkinter TopLevel()

Used to create the actual GUI element which shows the tkinter popup menu. It is constructed using TopLevel to ensure that even after one of the pop ups is closed, the others stay in the foreground of the application.

3.9.2(e) message → str

This attribute is used to construct the message to display in the pretend SMS/email message to the user.

3.9.3 class PopUp — Methods

3.9.3(a) __init__(self, root: Tk, freedge: Freedge)

Creates a new instance of the PopUp class, setting self.root=root and building a message to display via a popup window about the freedge object which was passed in.

3.9.3(b) false_button(self)

A function with the purpose of processing the user input as False in terms of whether or not the fridge is still active.

3.9(c) true_button(self)

A function with the purpose of processing the user input as True in terms of whether or not the fridge is still active.

3.9(d) get_response(self)

A function with the purpose of updating the status of the button based on which one the user selected.

3.9(e) on_close(self)

A function with the purpose of managing the case where a user closes the notification pop up window and records this action as a non-response.

3.9(f) exit_(self)

A function with the purpose of exiting the notification window.

3.10 notificationMgmtEmail.py:

A child module of notificationMgmt, this is a file containing skeleton code to be updated with Twilio or another method in order to send notifications via email.

3.11 notificationMgmt.SMS.py:

A child module of notificationMgmt, this is a file containing skeleton code to be updated with Twilio or another method in order to send notifications via SMS.

4. Known Bugs

4.1 No check for csv file format

With the current system, there is no check for correct csv file format. If the headers of an imported csv file do not match the headers required by the system, the user will still be able to upload the csv file. However, the information from the csv file will not be displayed on the screen, and there will be errors present on the terminal window. In a future version of this software, it would be useful to have a warning message appear to notify the user that their csv file format does not match the required format, and to try uploading a different csv file.