eldonationtracker

Release 4.1.1

Eric Mesa

Mar 02, 2020

CONTENTS:

ELDonation Tracker is used to provide donation information and updates when streaming or recording a VOD in OBS or XSplit. For a video explaining how to use this program, visit: http://djotaku.github.io/ELDonationTracker/

Note: Starting 20200227 this project will strictly follow Semantic Versioning as laid out at https://semver.org/

That means:

Given a version number MAJOR.MINOR.PATCH, increment the:

1. MAJOR version when you make incompatible API changes,
2. MINOR version when you add functionality in a backwards compatible manner, and
3. PATCH version when you make backwards compatible bug fixes.

CHAPTER ONE

INSTALLATION

1.1 Via PyPi

The first thing to decide is whether you want to install ELDonationTracker to your system packages, user packages, or a virtual environment.

1.1.1 Virtual Environment (recommended)

Create the folder you want to work in and cd into it.

```
python3 -m venv .
source ./bin/activate
# when you are done using the program you can type deactivate
# first, make sure you have the latest pip, I've had trouble installing with old␣
↪versions
pip install --upgrade pip
pip install eldonationtracker
# on Windows you may need to type python -m pip install eldonationtracker
# Grab participant.conf from git repo or create based on documentation
# Place participant.conf in persistent location, see the page in documentation
```

1.1.2 System Packages

Todo: Add instructions and note that this could potentially cause problems with system packages.

1.1.3 User Install

Todo: Add Instructions

1.2 Via Github

Here you have a few options depending on what you want to do.

1.2.1 GUI Single Executable Users

Go to the latest release and download the file that ends in “For Windows”. Unzip it to the location you want to use and then proceed to *Usage*.

Pyinstaller images will no longer be made for Linux as there are issues with the version of libC it links to as well as other side effects from the VM used by the Github CI system. Linux users can still use the GUI via PyPi, Source Code download, or git clone.

Changed in version 4.0.1.

1.2.2 Commandline Users and/or Developers

Two options:

1. Go to the latest release and click on “Source Code (zip)” or “Source Code (tar.gz)” depending on whether you’re using on Windows or Linux. Then proceed to *Usage*.
2. Go to the main Github page and click on “Clone or Download” and click the button to copy the URL to your clipboard. Then run:

```
git clone https://github.com/djotaku/ELDonationTracker.git
```

And any time you want to get up to the latest version you can just go to that folder and type:

```
git pull
```

The master branch is always equivalent to the latest release (except maybe with more up-to-date documentation) so you should always end up with a working version of ELDonationTracker if you do a git pull. (As long as you’re not changing any files. For that reason you may want to move your participant.conf to the persistent location - see *participant.conf* for that location) Then proceed to *Usage*.

CHAPTER TWO

USAGE

2.1 GUI Single Executable users

This refers to you if you downloaded a file like Extra Life Donation Tracker for Windows v3.4.zip.

2.1.1 Launching

Go into the folder you unzipped. Find the file called gui.exe and double-click it. If Windows or your anti-virus software throws a warning, click through to allow it to run.

Note: In a future version, this may change to just be one executable instead of a folder full of files.

2.1.2 Using

Todo: Add documentation on how to use it, include images.

2.2 Commandline users (PyPi)

Go to the folder you created in *Installation*. If you don't have the virtual environment activated, start with that:

```
python3 -m venv .
source ./bin/activate
# to check for upgrades
pip install --upgrade eldonationtracker
```

2.2.1 GUI

Make sure you have the *participant.conf* in the persistent location. You can grab the one in the Github repo or create your own by looking at the example there. Once the GUI has actually started, you can easily modify the config file via the GUI. To start the GUI:

```
python -m eldonationtracker.gui
```

That should work just fine. Keep an eye on the commandline for any errors or messages from eldonationtracker. The benefit you get from using the GUI is that once the GUI comes up you can click "tracker" to get a window that will display an image and text when a donation is registered.

eg:

You can also edit the settings in a GUI rather than on the commandline.

2.2.2 Commandline Only (No GUI)

Make sure you have the *participant.conf* in the persistent location. You can grab the one in the Github repo or create your own by looking at the example there. To start the commandline only version:

```
python -m python -m eldonationtracker.extralifedonations
```

Of course, you can import the modules into your own scripts and modify how you use the code I've written. In that case, you may be interested in the module index to get a good look at the API available to your program.

2.3 Commandline users (non-PyPi)

If you downloaded a zip or tar file, unzip it first, then cd into that directory. If you did a git clone, cd in to that directory. Afterwards, follow along below to create a virtual environment (so as not to mess with your Python installation), grab the required packages, and run the program. (For information on what you should put into participant.conf, see *participant.conf*.

```
python3 -m venv .
source ./bin/activate
# when you are done using the program you can type deactivate
pip install -r requirements.txt
# on Windows you may need to type python -m pip install -r requirements.txt
# edit participant.conf
cd eldonationtracker
# for the GUI:
python gui.py
# for the commandline only
python extralifedonations.py
```

The benefit you get from using the GUI is that once the GUI comes up you can click "tracker" to get a window that will display an image and text when a donation is registered.

eg:

You can also edit the settings in a GUI rather than on the commandline. Once the settings are configured, hit the run button. You should get the same output on the commandline as you would if you weren't running the GUI. Check there for any errors or messages from the program.

CHAPTER

THREE

PARTICIPANT.CONF

This is the configuration file for the program.

3.1 Example

Here's an example from my config file in 2020:

```
{
"Version" : "1.0",

"extralife_id" : "401280",

"text_folder" : "/home/ermesa/Dropbox/ELtracker/",

"currency_symbol" : "$",

"team_id" : "50394",

"tracker_image": "Engineer.png",

"donation_sound": "/home/ermesa/Programming Projects/python/extralife/Donation.mp3",

"donors_to_display": "5"

}
```

If you are using the GUI with the program, you can edit the file with the GUI and it should always turn out OK. If you are editing it in a text editor, the important thing to remember is the quotation marks.

3.2 Locations

The program will look in 3 locations.

First, it will look in the XDG official locations. If you use "save persistent settings" in the GUI, it will save to this location. On Linux this will be in $HOME/.config/extralifedonationtracker. On Windows it will be C:UsersusernameAppDataRoamingconfigextralifedonationtracker where you replace username with your Windows username.

Second, it will look up two directory levels. If you cloned from Github or grabbed a Source Code.zip or .tar.gz, this would be in the first directory you cd into - the one that has requirements.txt, README.md, etc.

Third, it will look in the directory where it is being run. If you are using the GUI single executable, this means the same folder as gui.exe (on Windows) or gui (on Linux).

As long as it's in one of those directories, it will be fine. Otherwise it will error out.

3.3 Config Values

- Version: this should be left alone. It's to know if a persistent config needs updating
- extralife_id: this should be your extralife id, the end of your campaign URL
- text_folder: this is where the text files generated by the program will be placed
- currency_symbol: the prefix for your money
- team_id: if you're on a team, grab from team page URL. If not, put None without quotation marks
- tracker_image: if you're using the GUI, what you want to appear when you get a donation
- donation_sound: if you're using GUI, what you want to hear when there's a donation
- donors_to_display: controls how many donors/donations appear in files that have more than one

CHAPTER

FOUR

CALL_SETTINGS

Contains the programming logic for the settings window in the GUI.

class eldonationtracker.call_settings.**MyForm**(*participant_conf*)

Bases: PyQt5.QtWidgets.QDialog

Class for the settings Window.

persistent_save()

Use xdg_config, saves a persistent config to the XDG spot.

reload_config()

Reload the values from the config file.

Called by gui.py before loading the settings window.

revert()

Revert the values in the settings window.

If the user made mistakes while editing the config this will take them back to the values since they opened the window.

save()

Save the values in the window to participant.conf.

Calls the write_config method from extralife_IO.ParticipantConf.

eldonationtracker.call_settings.**main**(*participant_conf*)

Launch the window.

CHAPTER FIVE

CALL_TRACKER

A window that displays the last donation. Useful during streaming.

class `eldonationtracker.call_tracker.`**`MyForm`**(*participant_conf*)

Bases: `PyQt5.QtWidgets.QDialog`

The class to hold the tracker window.

`loadAndUnloadTest`()

Trigger the tracker functionality.

This causes the image and sound to load so that the user can test to see how it's going to look on their OBS or XSplit screen as well as to make sure they can hear the sound. Called by gui.py.

`eldonationtracker.call_tracker.`**`main`**(*participant_conf*)

Launch the window.

CHAPTER

SIX

DONATION

A class to hold the Donation attributes and methods.

class eldonationtracker.donation.**Donation**(*name*, *message*, *amount*)

Bases: `object`

Donation Attributes.

Class exists to provide attributes for a donation based on what comes in from the JSON so that it doesn't have to be traversed each time a donor action needs to be taken.

Parameters

- **name** (*str*) – the name of the donor for this donation. If the donor wished to stay anonymous, the variable is set to "Anonymous"
- **message** (*str*) – the message associated with the donation.
- **amount** (*int*) – the amount of the donation. If they blocked it from showing it is set to 0.

CHAPTER SEVEN

DONOR

A donor.

class `eldonationtracker.donor.`**`Donor`**(*json*)

Bases: `object`

Donor Attributes.

Class exists to provide attributes for a donor based on what comes in from the JSON so that it doesn't have to be traversed each time a donor action needs to be taken.

Parameters

- **`json`** (*`json`*) – JSON attributes from the API
- **`name`** (*`str`*) – donor's name if provided, else Anonymous
- **`donor_id`** (*`str`*) – the ID assigned by the API (currently not used)
- **`image_url`** (*`str`*) – the URL for the donor's avatar (currently not used)
- **`amount`** – the sum of all donations the donor has made this campaign number_of_dononations: the number of donations the donor has made this campaign

`json_to_attributes`(*json*)

Convert API JSON values to Donor attributes.

May be overwritten by child classes.

Parameters **`json`** (*`json`*) – JSON attributes from the API

CHAPTER

EIGHT

EXTRALIFEDONATIONS

Grabs donor JSON data and outputs to files.

class eldonationtracker.extralifedonations.**Participant**(*participant_conf*)

Bases: `object`

Owns all the attributes under the participant API.

Also owns the results of any calculated data.

Participant.conf variables:

Parameters

- **ExtraLifeID** (*int*) – the participant's extra life ID
- **textFolder** (*str*) – where the output txt files will be written on disk
- **CurrencySymbol** (*str*) – for the output txt files
- **donors_to_display** (*int*) – for txt files that display multiple donors (or donations), the number of them that should be written to the text file.

Donor Drive API api info at https://github.com/DonorDrive/PublicAPI

Donor Drive Variables:

Parameters

- **participant_url** (*str*) – API info for participant
- **donorURL** (*str*) – donation API info (should be renamed to donationURL)
- **participant_donor_URL** (*str*) – API info for donors. Useful for calculating top donor.
- **participantinfo** (*dict*) – a dictionary holding data from participantURL:
  - totalRaised: total money raised
  - numDonations: number of donations
  - averageDonation: this doesn't come from the API, it's calculated in this class.
  - goal: the participant's fundraising goal
- **myteam** (*cls: eldonationtracker.team*) – An instantiation of a team class for the participant's team.
- **donationlist** (*list*) – a list of Donation class ojects made of donations to this participant

Helper Variables:

Parameters

- **`donorcalcs`** (*`dict`*) – a dictionary holding values for txt ouput:
  - **LastDonationNameAmnt: most recent donation,** donor name, amount of donation
  - TopDonorNameAmnt: top donor name and sum of donations
  - **lastNDonationNameAmts: based on value of donors_to_display** above, a list of the last N donor names and donation amounts
  - lastNDonationNameAmtsMessage: same with messages
  - lastNDonationNameAmtsMessageHorizontal: same, but horizontal
  - lastNDonationNameAmtsHorizontal: same, but no message
- **`loop`** (*`bool`*) – set to true on init, it's there so that the GUI can stop the loop.(if the GUI is being used. Otherwise, no big deal)

`run`()

Run loop to get participant data.

This should run getParticipantJSON, getDonors, the calculations methnods, and the methods to write to text files.

Warning: This will be changed in a future version to no longer be a loop and instead the loop will be in the if __name__=__main__ part. This will make it more consistent with the way team.py works and will enable some better efficiencies with the GUI.

`stop`()

Stop the loop.

`write_text_files`(*dictionary*)

Write OBS/XSplit display info to text files.

It uses the helper function extralife_IO.write_text_files to handle the task.

Parameters **`dictionary`** (*`dict`*) – Dictionary containing values to write to text files . The key will become the filename. The value will be written to the file.

CHAPTER NINE

EXTRALIFE_IO

Holds all the file and internet input and output.

class eldonationtracker.extralife_IO.**ParticipantConf**

Bases: object

Holds Participant Configuaration info.

Parameters

- **participant_conf_version** – version of participant.conf
- **version_mismatch** – Initialized to False.If true, the user has a different version of the participant.conf.
- **fields** – A dictionary initialed to None for all fields.
- **xdg** – By using the PedanticPackage, the directory will be created if it doesn't already exist.
- **participantconf** – holds a dictionary of the user's config file.

fields: dict = {'currency_symbol': None, 'donation_sound': None, 'donors_to_display

get_CLI_values() → Tuple[str, int]

Return data required for a CLI-only run.

Returns A tuple of strings with config values needed if only running on the commandline.

get_GUI_values() → Tuple[str, int]

Return values needed for the GUI.

Returns A tuple of strings with config values needed if only running the GUI

get_if_in_team() → bool

Return True if participant is in a team.

Returns True if the participant is in a team.

get_text_folder_only() → str

Return text folder data.

Returns A string with the text folder location.

get_tracker_image() → str

Return the tracker image location on disk.

Returns Location of tracker image on disk.

get_tracker_sound() → str

Return the donation sound image location on disk.

Returns location of the donation sound on disk.

`get_version``()`
Return version.

`get_version_mismatch``()` → bool
Return bool of whether there is a version mismatch.0

Returns True if the version the user is running is not the same as what this program has as its version.

`load_JSON``()` → dict
Load in the config file.

Checks in a variety of locations for the participant.conf file. First, it checks in the persistent settings location (XDG_CONFIG_HOME) . Then it checks the current directory and one level up. Otherwise it raises an exception and the program stops.

Returns A dictionary representing the JSON config file.

Raises FileNotFoundError

`participant_conf_version: str = '1.0'`

`reload_JSON``()`
Reload JSON and update the fields.

`update_fields``()`
Update fields variable with data from JSON.

`version_mismatch: bool = False`

`write_config``(`*config: dict*, *default: bool*`)`
Write config to file.

Only called from GUI. Commandline user is expected to edit participant.conf manually. Afterward it triggers self.reloadJSON to have the program run with the updated config.

Parameters

- **`config`** – A dictionary holding the config values.
- **`default`** – If True, will save the file in the current directory.

`eldonationtracker.extralife_IO.`**`get_JSON`**`(`*url: str*, *order_by_donations: bool = False*`)` → dict
Grab JSON from server.

Connects to server and grabs JSON data from the specified URL. The API server should return JSON with the donation data.

Parameters

- **`url`** – API URL for the specific json API point.
- **`order_by_donations`** – If true, the url param has has text appended that will cause the API to return the data in descending order of the sum of donations.

Returns JSON as dictionary with API data.

Raises HTTPError, URLError

`eldonationtracker.extralife_IO.`**`multiple_format`**`(`*donors*, *message: bool*, *horizontal: bool*, *currency_symbol: str*, *how_many: int*`)` → str
Format string for output to text file.

This function is for when there is are multiple donors or donations. For example, for the text file holding the Top 5 donors.

Parameters

- **`donors`** – An iterable of Donors or Donations class objects.
- **`message`** – If True, the message (if any) from the donor object :param horizontal: If True, format the message horizontally. Else vertically.
- **`currency_symbol`** – The currency symbol to append to the return string.
- **`how_many`** – A number for how many donors/donations to append to the string.

Returns A string containing the inputs formatted.

Horizontal example:

John Doe - $100.00 - Thanks for raising money for the kids. | Jane Doe - $25.00 - Hurray!

Vertical example:

John Doe - $200.00 - a message

Jane Doe - $75.00 - another message

`eldonationtracker.extralife_IO.`**`single_format`**(*donor*, *message: bool*, *currency_symbol: str*) → str

Format string for output to text file.

This function is for when there is only one donor or donation. For example, for the text file holding the most recent donation.

Parameters

- **`donor`** – Donor or Donation class object.
- **`message`** – If True, the message (if any) from the donor object :param currency_symbol: The currency symbol to append to the return string.

Returns A string containing the inputs formatted. For example:

John Doe - $100.00 - Thanks for raising money for the kids.

`eldonationtracker.extralife_IO.`**`write_text_files`**(*dictionary: dict*, *text_folder: str*)

Write info to text files.

The dictionary key will become the filename and the values will be the content of the files.

Parameters

- **`dictionary`** – The dictionary with items to output.
- **`text_folder`** – The directory to write the text files.

CHAPTER TEN

GUI

The main GUI window.

class eldonationtracker.gui.**ELDonationGUI**

Bases: PyQt5.QtWidgets.QMainWindow, eldonationtracker.design.Ui_MainWindow

The main gui Window.

callSettings()

callTracker()

check_for_update()

deadbuton()

getsomeText()

load_documentation()

quit()

Quit the application.

First need to stop the thread running the CLI code.

readFiles(*folders*, *files*)

runbutton()

show_about()

stopbutton()

testAlert()

version_check()

class eldonationtracker.gui.**donationGrabber**(*participant_conf*)

Bases: threading.Thread

counter = 0

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

stop()

eldonationtracker.gui.**main**()

CHAPTER ELEVEN

IPC

Writes to an IPC value to allow different modules to pass information.

This is used to allow the tracker to know that new data needs to be read in because a new donation has ocurred.

`eldonationtracker.ipc.`**`writeIPC`**`(`*folder: str*, *value: str*`)`

Write to the IPC file.

This is used to let the call_tracker module know that a new donation has been made.

Parameters

- **`folder`** – The location of trackerIPC.txt
- **`value`** – The value to write to the file.

Raises IOError

CHAPTER TWELVE

TEAM

Contains classes pertaining to teams.

class eldonationtracker.team.**Team**(*team_id: str*, *folder: str*, *currency_symbol: str*)

Bases: object

Hold Team API Data.

API Variables:

param team_url URL to the team JSON API

param team_participant_url URL to the JSON api for participants in the team.

param team_info a dictionary to hold the following:

- Team_goal: fundraising goal
- Team_captain: team captain's name
- Team_totalRaised: total amount raised by team
- Team_numDonations: total number of donations to the team

param participant_list a list of the most recent participants

param top_5_participant_list a list of the top 5 team participants by amount donated.

Helper Variables:

param output_folder the folder that will contain the output txt files

param currency_symbol for formatting text

param participant_calculation_dict dictionary holding output for txt files:

- Team_Top5Participants: top 5 participants by donation amount
- Team_Top5ParticipantsHorizontal: same, but horizontal
- Team_TopParticipantNameAmnt: Top participant and amount

get_participants()
Get team participant info from API.

get_team_json()
Get team info from JSON api.

get_top_5_participants()
Get team participants.

participant_run()
Get and calculate team partipant info.

team_run()
Get team info from API.

write_text_files(*dictionary: dict*)
Write info to text files.

Parameters **dictionary** – The dictionary containing the values to write out to text files.

class eldonationtracker.team.**TeamParticipant**(*json*)
Bases: *eldonationtracker.donor.Donor*

Participant Attributes.

Inherits from the donor class, but over-rides the json_to_attributes function.

API variables:

Parameters

- **name** – participant's name or Anonymous
- **amount** – the sum of all donations by this participant
- **number_of_donations** – number of all donations by this participant
- **image_url** – the url of the participant's avatar image (not used)

json_to_attributes(*json*)
Convert JSON to Team Participant attributes.

Parameters **json** – JSON attributes from API

CHAPTER THIRTEEN

UPDATE_AVAILABLE

Return true if there is an update available.

An update is available if the version on PyPi is higher than the version the user has on their machine.

`eldonationtracker.utils.update_available.`**`get_pypi_version`**`()` → str

Use PyPi JSON API to get latest version.

Returns A string with the version number.

`eldonationtracker.utils.update_available.`**`main`**`()`

`eldonationtracker.utils.update_available.`**`update_available`**`(`*pypi_version: str*, *current_version: str*`)` → bool

Use semver module to calculate whether there is a newer version on PyPi.

Returns True if the PyPi version is higer than the version being run. Returns false if the version being compared to PyPi is equal or greater than the PyPi version.

CHAPTER FOURTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX