**FamilyTreeSplitter**

A utility to split a GEDCOM format file.

**Overview**

This utility allows the user to extract a branch of a family tree, from a GEDCOM format file, and create a new GEDCOM file.

It allows the user to specify a family in params. It then processes the GEDCOM file, tagging all ancestors of the specified family, and all relatives of these ancestors.

It then produces a ‘New.ged’ file containing all the details from the source GEDCOM file, but only for the tagged individuals and families.

The process of creating the new GED file is separated from the process of tagging, to allow manual tagging to be done, for example to un-tag individuals that are still living, and their families, to protect their privacy if required.

Modules

* FamilyTreeSplitter.pyw

The main module. This module presents the user with a Windows user interface, allowing them to edit parameters, process a GEDCOM format file or create the new GEDCOM file.

* params.py

See the GEDFileProcessor utility for information on this module.

Parameters are:

|  |  |
| --- | --- |
| GED File | The name of the GEDCOM format file to be processed. The file should be in the location where the utility runs. |
| Initial Family | The family uses as the basis for the new GEDCOM file. To obtain this, first process a GEDCOM format file. This will produce Individuals.txt, Families.txt and Children.txt containing lists of individuals, families and children in the GEDCOM file. Then, in Individuals.txt, find the IDs of the husband and wife for the family, then, in Families.txt, find the ID of the family. |

* process\_ged\_file.py

See the GEDFileProcessor utility for information on this module.

* individuals.py

See the GEDFileProcessor utility for information on this module.

* families.py

See the GEDFileProcessor utility for information on this module.

* children.py

See the GEDFileProcessor utility for information on this module.

* tag\_records.py

This module contains the *tag\_ancestors\_families* subroutine that tags individuals and families in Individuals.txt, Families.txt, where they are ancestors of the initial family.

It first reads the Individuals.txt, Families.txt and Children.txt files into the ‘individuals’, ‘families’ and ‘children’ lists.

It then calls *tree\_walk* to tag all the families where they are ancestors of the initial family defined in params.

It then calls *tag\_family* for each family tagged in *tree\_walk* to tag the individuals that make up that family.

It then calls *tag\_additional\_families* to tag the families of individuals tagged by *tag\_family*, and also calls *tag\_family* for these individuals and does this repeatedly until no new individuals are tagged.

It first writes the ‘individuals’, ‘families’ and ‘children’ lists with new tags to Individuals.txt, Families.txt and Children.txt files.

*tree\_walk* tags the family (id passed as a parameter) in the ‘families’ list. Then, if the family has a husband, it gets the id of the family where the husband was a child and calls *tree\_walk* recursively with this family id. Then, if the family has a wife, if gets the family where the wife was a child and calls *tree\_walk* recursively with this family id.

*tag\_family* tags the husband, wife and children in the family (id passed as a parameter) and returns the number of individuals tagged that hadn’t been previously tagged.

*tag\_additional\_families* scans the ‘individuals’ list and, for each tagged individual, it searches the ‘families’ list for a family where the individual was a husband or wife. If found, it tags that family, then calls *tag\_family* to tag all the individuals in the family. It then returns the number of individuals newly tagged.

* write\_ged\_file.py

This module writes the new GEDCOM file with the tagged individuals and families.

It first calls *read\_individuals* and *read\_families* to get the tagging information. It then calls *load\_working* to get a ‘working’ taggable list of records contained in the old GEDCOM file. It then calls *tag\_working* to tag the individual, family and other records in the ‘working’ list that are required in the new GEDCOM file. It calls *write\_working*, to write the ‘working’ list to working.txt (purely to help in debugging), then calls *write\_new\_ged\_file* to write the records tagged in the ‘working’ list to the new GEDCOM file.

*load\_working* loads the old GEDCOM file into a ‘working’ list that identifies where a record relates to an individual or a family, and if so which individual or family and which allow records to be tagged.

*tag\_working* tags records in the ‘working’ list where they relate to a tagged individual or family, or where they are header/trailer records

*write\_working* writes the ‘working’ list to working.txt

*write\_new\_ged\_file* writes the records tagged in the ‘working’ list to the new GEDCOM file.