GEDCOM Utilities Documentation

Release 0.5.1

Andy Salnikov

May 02, 2021

CONTENTS

Contents:

CHAPTER

ONE

GEDCOM UTILITIES

Tools for converting GEDCOM data into document formats.

- Free software: MIT license
- Documentation: https://ged2doc.readthedocs.io.

1.1 Features

- Conversion of GEDCOM files into nicely formatted HTML or ODT (OpenDocument Text) format
- Output includes most textual information, images/photos, an segments of genealogical tree
- Support for English and Russian output translations, more translations can be added easily
- Can be used as standalone command-line application or as a Python API

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install ged2doc, run this command in your terminal:

```
$ pip install ged2doc
```

This is the preferred method to install GEDCOM Utilities, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for ged2doc can be downloaded from the Github repo.

You can either clone the repository:

```
$ git clone git://github.com/andy-z/ged2doc
```

Or download the tarball:

```
$ curl  -OL https://github.com/andy-z/ged2doc/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

2.3 Binary install on Windows

Both previous variants can be used on Windows platform if you install Python first, any reasonable Python distribution can work. Alternative variant that does not require Python installation is to install ged2doc as a separate Windows application which includes all necessary components. Download an installer (EXE file) from latest release (or any specific release) and execute it on your computer. After installation Program Menu and Desktop will show a `ged2doc` shortcut which starts a command prompt with access to `ged2doc` command line application.

CHAPTER

THREE

USAGE

ged2doc package can be used either as a standalone application (command line tool) with the same name ged2doc or as a Python library/package that can be imported by other Python code. ged2doc main function is to parse GEDCOM data and convert it into printable/browsable document format. For GEDCOM parsing ged2doc uses ged4py package, if you only need to parse GEDCOM file from Python code and do not need to produce output document then ged4py package is a good place to start.

3.1 Command line tool

ged2doc application is the main user interface, it reads and parses GEDCOM file and produces document in one of the supported formats (currently HTML and OpenDocument Text/OTD).

ged2doc application is a command line tool which is run from a terminal, it has a large number of command line options which control contents and appearance of the produced output document. To get a brief description of all options run the command with `--help` or `-h` option:

```
% ged2doc --help
usage: ged2doc [-h] [-v] [--log PATH] [--version] [-i PATH] [-p PATTERN]
               [--encoding ENCODING] [--encoding-errors MODE] [-t {html,odt}]
               [-l LANG_CODE] [-d FMT] [-s ORDER] [--locale LOCALE]
               [--no-missing-date] [--no-image] [--no-toc] [--no-stat]
               [-w NUMBER] [--name-surname-first] [--name-comma]
               [--name-maiden] [--name-maiden-only] [--name-capital]
               [--html-page-width SIZE] [--html-image-width SIZE]
               [--html-image-height SIZE] [-u] [--odt-page-width SIZE]
               [--odt-page-height SIZE] [--odt-margin-left SIZE]
               [--odt-margin-right SIZE] [--odt-margin-top SIZE]
               [--odt-margin-bottom SIZE] [--odt-image-width SIZE]
               [--odt-image-height SIZE] [--first-page NUMBER]
               [--odt-tree-type FORMAT]
               input output

Convert GEDCOM file into document.

positional arguments:
  input                 Location of input file, input file can be either
                        GEDCOM file or ZIP archive which can also include
                        images.
  output                Location of output file.

optional arguments:
  -h, --help            show this help message and exit
```

(continues on next page)

(continued from previous page)

```
  -v, --verbose         Print some info to standard output, -vv prints debug
                        info.
  --log PATH            Produces log file with debugging information.
  --version             Print version information and exit

... description of all other options ...
```

Application takes two required positional arguments – the name of input and output files – and a bunch of optional arguments. Sections below provide more detailed description of the positional and optional arguments that this command accepts. Optional arguments with value can be specified either as `--option value` or `--option=value`.

3.1.1 Input files

First positional argument is an input file which can be either a GEDCOM file or a ZIP archive. If ZIP archive is specified as an argument then it should contain GEDCOM file and it may also contain image files referenced from GEDCOM file. `ged2doc` can only process one GEDCOM file at a time even if ZIP archive contains multiple GEDCOM files. Application tries to guess which file in ZIP archive is a GEDCOM file based on pattern match, by default `*.ged*` pattern is used which matches names like `file.ged` or `file.gedcom`. File in archive can be located in any sub-folder, whole ZIP archive is searched for matching files.

If GEDCOM file in archive has different extension or if there is more than one file matching default pattern then option `-p PATTERN` should be used to provide more exact match, e.g.:

```
$ unzip -l archive.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  2017-10-30 22:28   folder1/
    77279  2017-10-30 22:28   folder1/file1.ged
        0  2017-10-30 22:28   folder2/
    79999  2017-10-30 22:28   folder2/file2.ged

$ ged2doc -p file1.ged archive.zip output.html
```

GEDCOM file can contain references to images which can be included into resulting document (in current implementation only one main image is included per person). `ged2doc` tries to locate image file by first using unchanged path directly from GEDCOM file (if the path is relative then it is resolved relative to current working directory). If image file cannot be found and `-i PATH` is given then `ged2doc` tries to locate image using only base name of the image path by recursively scanning `PATH` given in `-i` option. If input file is an archive then archive is searched for images first using base name of the path from GEDCOM file, if image is not found in archive then above logic is used to search for image on disk.

3.1.2 Output file

`ged2doc` saves output document in a file given as a second positional argument. Currently `ged2doc` can generate either single-page HTML with embedded images or OpenDocument Text (ODT) format. HTML format is better suitable for online browsing as it includes navigation, ODT format is better for printing (possibly after modification in OpenOffice/LibreOffice).

The type of the produced document is determined by default from the file extension – if extension is `.htm` or `.html` then HTML format is produced, if extension is `.odt` then ODT format is saved. If file has other extension then one has to use `-t` or `--type` option to specify document format:

```
$ ged2doc -t html archive.zip output.txt
$ ged2doc --type=odt archive.zip output.opendoc
```

3.1.3 GEDCOM file encoding

Properly constructed GEDCOM file should have enough information in it for `ged2doc` to determine its encoding. In some cases it may be necessary to specify file encoding explicitly or to change how decoding errors are handled. By default `ged2doc` tries to determine file encoding from file contents and it terminates for any encoding-related errors. You can use `--encoding` option to force it to use different encoding and `--encoding-errors` option to control error handling. The argument to `--encoding` option is the name of the encoding such as `utf-8`, `iso-8859-1`, etc. The argument to `--encoding-errors` option is one of the keywords:

`strict` Default behavior, application aborts in case of errors

`ignore` Application removes problematic encoded characters

`replace` Application replaces problematic encoded characters with special replacement character ()

Here is an example of a command which forces utf-8 encoding but replaces incorrectly encoded data:

```
$ ged2doc --encoding=utf-8 --encoding-errors=replace file.ged out.html
```

3.1.4 Common output options

Languages

`ged2doc` can produce output document in different languages (currently supporting English, Russian, Polish, and Czech). By default the language is determined from system configuration which may not always work reliably. To specify output language explicitly use `-l CODE` option, `CODE` is the language code (`en` for English, `ru` for Russian, `pl` for Polish, `cz` for Czech).

Date Format

GEDCOM data can include dates in that can be either precise or approximate. `ged2doc` tries to represent all possible dates in output document in a reasonable way according to locale. Default date format in the output document is determined by the document language but it can also be changed via `-d FMT` (or `--date-format=FMT`) option, `FMT` can be one of:

`YMD` Space-separated year, month name, and day, e.g.: `2000 Dec 31`; `2017 Dec`; `2017`

`MDY` Space-separated month name, day, and year, e.g.: `Dec 31 2000`; `Dec 2017`; `2017`

`DMY` Space-separated day, month name, and year, e.g.: `31 Dec 2000`; `Dec 2017`; `2017`

`Y-M-D` Dash-separated year, month name, and day, e.g.: `2000-Dec-31`; `2017-Dec`; `2017`

`D-M-Y` Dash-separated day, month name, and year, e.g.: `31-Dec-2000`; `Dec-2017`; `2017`

`Y/M/D` Slash-separated year, month number, and day, e.g.: `2000/12/31`; `2017/12`; `2017`

`M/D/Y` Slash-separated month number, day, and year, e.g.: `12/31/2000`; `12/2017`; `2017`.

`Y.M.D` Dot-separated year, month number, and day, e.g.: `2000.12.31`; `2017.12`; `2017`

`D.M.Y` Dot-separated day, month number, and year, e.g.: `31.12.2000`; `12.2017`; `2017`. This is default for `ru` language.

MD,Y Comma after day, month number, year, e.g.: `Dec 31, 2000`; `Dec 2017`; `2017`. This is default for `en` language.

Person ordering

Ordering of persons in output document is controlled by `--sort-order=ORDER` option, `ORDER` is one of:

last+first Persons are ordered according to family (married) name and given name, this is default ordering.

first+last Persons are ordered according to given name and family (married) name.

maiden+first Persons are ordered according to family (maiden) name and given name.

first+maiden Persons are ordered according to given name and family (maiden) name.

By default ordering of the names is performed according to collation rules of the current system locale. If system locale does not correspond to the language of the document one can specify different locale using `--locale=LOCALE` option. `LOCALE` is the name of the locale and it is usually system dependent, e.g. the name can be `Russian` or `Czech` on Windows host or `ru_RU.UTF-8` or `cs_CZ.UTF-8` on Linux host. On Linux it is also possible to change locale by using `LC_ALL` or `LC_COLLATE` environment variables. Check system documentation for how to install and enable locales.

Events without dates

By default `ged2doc` outputs all events including those events that do not have associated dates (events are prefixed with "Date unknown"). To disable printing of those events use `--no-missing-date` option.

Images

By default `ged2doc` adds an image for each person (if it can find it on disk), one can disable this by using `--no-image` option which disables all images in output file.

TOC

Table of Contents is added by default to each document, `--no-toc` option can be used to disable generation of TOC.

Statistics

Some statistical info is normally added to each document (e.g. name frequency), `--no-stat` option can be used to disable it.

Tree Width

For each person `ged2doc` adds a small inline graphical representation of ancestor tree, by default four generations are represented in the tree. Option `-w NUMBER` (`--tree-width NUMBER`) can be used to change the number of generations in this tree.

3.1.5 Name formatting options

Different locales use different name formatting rules which may be quite complicated. By default `ged2doc` represents person names as given name followed by family (married) name (e.g. `Jane Smith`) but there are also multiple options that can change this representation:

--name-surname-first Format names with surname in leading position, e.g. `Smith Jane`

--name-comma Format names with surname followed by comma (only if surname is in leading position), e.g. `Smith, Jane`

--name-maiden Format names with surname followed by maiden name in parentheses, e.g. `Jane Smith (Ivanova)`

--name-maiden-only Format names with maiden name for surname, e.g. `Jane Ivanova`

--name-capital Format names with surname and maiden name in all capital, e.g. `Jane SMITH`

Combining these options should produce expected effect, e.g. `--name-surname-first --name-comma --name-capital` would produce something like `SMITH (IVANOVA), Jane`.

3.1.6 Specifying size in options

Few options below take size as a value, size can be specified in different units. Units can be screen-based (pixels) or print-based (inches/points/mm). You can specify sizes in any form, output document format determines actual type of units to use. When `ged2doc` needs to convert units of one type into another it uses a fixed conversion factor of 96 DPI (dots/pixels per inch).

Supported units are:

px Size is given in pixels, typically used for on-screen dimensions, probably most useful for HTML output. Example: `100px`.

pt Size is given in points, typically used for print dimensions, one point is 1/72 of inch. Example: `100pt`.

in Size is given in inches, typically used for print dimensions. Example: `6.5in`.

mm Size is given in millimeters, typically used for print dimensions. 1 in = 25.4 mm. Example: `100mm`.

cm Size is given in centimeters, typically used for print dimensions. 1 in = 2.54 cm. Example: `10cm`.

Options that accept size as value have default unit type, if option default unit is pixels then giving it value of `300` is the same as giving `300px`.

3.1.7 HTML Options

There are few options that are specific to HTML output:

--html-page-width SIZE HTML page width, default unit is pixels; default value: `800px`

--html-image-width SIZE Image width, default unit is pixels; default value: `300px`

--html-image-height SIZE Image height, default unit is pixels; default value: `300px`

-u, --html-image-upscale Re-scale images which are smaller than size given by the options above. Without this option small images will be displayed in their actual size without re-scaling.

3.1.8 ODT Options

Options specific to ODT output:

--odt-page-width SIZE Page width, default unit is inches; default value: `6in`

--odt-page-height SIZE Page height, default unit is inches; default value: `9in`

--odt-margin-left SIZE Page left margin, default unit is inches; default value: `0.5in`

--odt-margin-right SIZE Page right margin, default unit is inches; default value: `0.5in`

--odt-margin-top SIZE Page top margin, default unit is inches; default value: `0.5in`

--odt-margin-bottom SIZE Page bottom margin, default unit is inches; default value: `0.25in`

--odt-image-width SIZE Image width, default unit is inches; default value: `2in`

--odt-image-height SIZE Image height, default unit is inches; default value: `2in`

--first-page NUMBER Number of the first page; default: `1`. Can be changed to something different if you plan to add extra pages at the beginning when printing the final document.

--odt-tree-type FORMAT Type of image format for ancestor tree, one of emf, svg; default: emf

3.1.9 Logging

In case application crashes or produces incorrect or unexpected output it would be helpful to produce log file with debug information and forward it to author (see *Contributing* for how to report bugs). To produce log file use `--log` option, e.g.:

```
$ ged2doc --log=log.txt input.ged page.html
```

which will create `log.txt` file in a current working directory.

3.1.10 Examples

To produce HTML page from GEDCOM file with default settings:

```
$ ged2doc input.ged page.html
```

To also include images that are referenced from GEDCOM file (assuming UNIX-style file names):

```
$ ged2doc -i /home/joe/gedcom_images input.ged page.html
```

Same but produce OpenDocument Text format:

```
$ ged2doc -i /home/joe/gedcom_images input.ged output.odt
```

If GEDCOM is named `gedcom.dump` is in ZIP archive together with all images:

```
$ ged2doc -p gedcom.dump input.zip page.html
```

If you need to specify different output language:

```
$ ged2doc -l ru input.zip page.html
```

To change date representation:

```
$ ged2doc -d Y-M-D input.zip page.html
```

To change how person name is printed:

```
$ ged2doc --name-surname-first --name-comma --name-maiden input.zip page.html
```

To change page size of ODT document:

```
$ ged2doc --odt-page-width=8.5in --odt-page-height=11in input.zip page.odt
```

3.2 Using Python modules

`ged2doc` package can be used from other Python code to perform the same conversion of GEDCOM file as command line tool does. There are three basic objects that are needed to run conversion from Python:

- file locator instance
- language translator instance
- writer/converter instance

3.2.1 File locator

File locator is an object responsible for finding/opening input files, both GEDCOM and images. It abstracts operations with filesystem and ZIP archives so that remaining code does not need to know details of file storage.

Factory method *`make_file_locator()`* is used to instantiate file locator and it takes few parameters:

```
from ged2doc.input import make_file_locator

input_file = "archive.zip"    # or you can pass GEDCOM file here
file_name_pattern = "*.ged*"
image_path = r"C:\Users\joe\Documents\gedcom_images"
flocator = make_file_locator(input_file, file_name_pattern, image_path)
```

3.2.2 Language translator

This object is responsible for translating output document into desired language. *`ged2doc.i18n.I18N`* implements this translation and it needs to be installed with couple of parameters:

```
from ged2doc.i18n import I18N

lang = "ru"          # language code, "en" or "ru"
date_fmt = "D.M.Y"   # one of the formats described above
tr = I18N(lang, date_fmt)
```

3.2.3 Conversion

Converter instance is made by instantiating specific converter class, currently there are two such classes:

- *ged2doc.html_writer.HtmlWriter* for conversion into HTML
- *ged2doc.odt_writer.OdtWriter* for conversion into ODT

Constructors of these classes take several parameters:

- file locator instance
- language translator
- output file name
- dictionary with options, includes all formatting options, see class documentation for details

After making converter instance the code should call its *save()* method to produce output file:

```
from .html_writer import HtmlWriter

output = "document.html"
# `flocator` and `tr` are instantiated in above examples, "..." signifies
# multiple optional keyword arguments that control appearance
writer = HtmlWriter(flocator, output, tr, ...)

# save the file
writer.save()
```

For more complete example check ged2doc.cli module.

3.3 Format-specific details

3.3.1 HTML details

`ged2doc` produces single-page HTML document which embeds all graphics (photos and tree graphs which are SVG structures). The size of the resulting document can be quite large. The images are re-sampled to a specified image size before embedding. Images that are smaller than specified image size are rescaled only if `--html-image-upscale` option is given.

3.3.2 ODT details

`ged2doc` does not have logic to correctly paginate output document and assign page numbers to Table of Contents entries. Instead it depends on external tools like LibreOffice to finalize and publish the document. When document is loaded into LibreOffice its Table of Contents needs to be refreshed – go to `Tools` menu, then `Update`, and `Indexes and Tables` which should rebuild all references in ODT file.

ODT files can be opened with MS Office (Word) application, but compatibility of MS Office with ODT format is not great and there are some known issues with MS Word when editing documents produced by `ged2doc`:

- Images in SVG format are not fully supported by MS Office, to visualize ancestor trees in MS Office they need to be produced in EMF format. `ged2doc` supports EMF since version 0.3, where EMF is the default format for ancestor trees (can be changed with `--odt-tree-type=svg` option). `ged2doc` before version 0.3 cannot be used to produce EMF.

- If file with ancestor trees in EMF format was opened an saved by LibreOffice then MS Office cannot render those tree images. There is no reliable interoperability between MS Office and LibreOffice, documents should only be edited by the same application.
- Table of Contents is not shown when ODT file is open by MS Office, it has to be added manually if one needs a table of contents in the document.

CHAPTER

FOUR

GEDDOC API

ged2doc	Top-level package for GEDCOM Utilities.

4.1 ged2doc

Top-level package for GEDCOM Utilities.

Modules

ged2doc.ancestor_tree	Module containing methods/classes for laying out ancestor trees.
ged2doc.ancestor_tree_emf	Module containing methods/classes for laying out ancestor trees.
ged2doc.ancestor_tree_svg	Module containing methods/classes for laying out ancestor trees.
ged2doc.cli	Console script for ged2doc.
ged2doc.dumbemf	Python module for generating EMF.
ged2doc.dumbsvg	Python module for generating SVG.
ged2doc.events	Utilities related to individual or family events.
ged2doc.html_writer	Module which produces HTML output.
ged2doc.i18n	Python module responsible for internationalization of ged2doc.
ged2doc.input	Module which handles input files.
ged2doc.name	Methods for manipulating/formatting names.
ged2doc.odt_writer	Module which produces ODT output.
ged2doc.size	Module which defines class for manipulating size values.
ged2doc.textbox	Module defining `TexBox` class and related methods.
ged2doc.utils	Various utility methods.
ged2doc.writer	Module which defines base class for all writer classes.

4.1.1 ged2doc.ancestor_tree

Module containing methods/classes for laying out ancestor trees.

Classes

AncestorTree(person[, max_gen, width, ...])	Class implementing layout of ancestor trees.
AncestorTreeVisitor()	Interface for tree visitors.
TreeNode(person, gen, motherNode, ...)	Class representing node in a tree, which is a box with a person name.

class ged2doc.ancestor_tree.**AncestorTree**(*person*, *max_gen=4*, *width='5in'*, *gen_dist='12pt'*, *font_size='10pt'*)

Bases: object

Class implementing layout of ancestor trees.

Parameters

person [ged4py.model.Individual] Corresponding individual, may be None.

max_gen [int] Maximum number of generations to plot, default is 4.

width [*ged2doc.size.Size*, optional] Specification for plot width, accepts anything convertible to *ged2doc.size.Size*.

gen_dist [*ged2doc.size.Size*, optional] Distance between generations, accepts anything convertible to *ged2doc.size.Size*.

font_size [*ged2doc.size.Size*, optional] Font size, accepts anything convertible to *ged2doc.size.Size*.

Attributes

height Full height of the tree (*ged2doc.size.Size*)

width Full width of the tree (*ged2doc.size.Size*)

Methods

visit(visitor)	Visit every node and edge in a tree.

property width
Full width of the tree (*ged2doc.size.Size*)

property height
Full height of the tree (*ged2doc.size.Size*)

visit(*visitor*)
Visit every node and edge in a tree.

Parameters

visitor [*AncestorTreeVisitor*] Tree visitor.

_visit(*visitor*, *node*)
Helper method for recursive visiting of the nodes.

_makeTree(*person*, *gen*, *max_gen*, *box_width*, *max_box_width*)
Recursively generate tree of TreeNode instances.

Fro internal use only.

class ged2doc.ancestor_tree.**AncestorTreeVisitor**
Bases: object

Interface for tree visitors.

Instances of this class can be passed to *AncestorTree.visit()* method to iterate over all nodes and edges in an ancestor tree.

Methods

visitFatherEdge(node, parentNode)	Visitor method for an edge leading from node to its mother.
visitMotherEdge(node, parentNode)	Visitor method for an edge leading from node to its mother.
visitNode(node)	Visitor method for a node in tree.

_abc_impl = <_abc_data object>

abstract visitNode(*node*)
Visitor method for a node in tree.

Parameters

node [*TreeNode*] Tree node.

abstract visitMotherEdge(*node*, *parentNode*)
Visitor method for an edge leading from node to its mother.

It is guaranteed that *visitNode* is called for both nodes before this method is called.

Parameters

node [*TreeNode*] Tree node.

parentNode [*TreeNode*] Parent tree node.

abstract visitFatherEdge(*node*, *parentNode*)
Visitor method for an edge leading from node to its mother.

It is guaranteed that *visitNode* is called for both nodes before this method is called.

Parameters

node [*TreeNode*] Tree node.

parentNode [*TreeNode*] Parent tree node.

class ged2doc.ancestor_tree.**TreeNode**(*person*, *gen*, *motherNode*, *fatherNode*, *box_width*, *max_box_width*, *font_size*, *gen_dist*)
Bases: object

Class representing node in a tree, which is a box with a person name.

Parameters

person [ged4py.model.Individual] Corresponding individual, may be None.

gen [int] Generation number, 0 for the tree root.

motherNode [*TreeNode*] Node for mother, can be `None`.

fatherNode [*TreeNode*] Node for father, can be `None`.

box_width [*ged2doc.size.Size*] Desired width of this node, actual width can grow.

max_box_width [*ged2doc.size.Size*] Maximum width this node can grow to.

font_size [*ged2doc.size.Size*] Size of the font for the text.

gen_dist [*ged2doc.size.Size*] Horiz. distance between generations.

Attributes

person Person corresponding to this node, can be None (`ged4py.model.Individual`).

subTreeHeight The height of the whole tree including parent boxes (`Size`).

textbox Textbox for this node (`TextBox`).

Methods

setY0(y0)	Recalculate Y position of box tree so that topmost box is at `y0`.

_vpadding = Size(0.027777777777777776in)

property person
Person corresponding to this node, can be None (`ged4py.model.Individual`).

property textbox
Textbox for this node (`TextBox`).

property subTreeHeight
The height of the whole tree including parent boxes (`Size`).

setY0 (*y0*)
Recalculate Y position of box tree so that topmost box is at `y0`.

Parameters

y0 [*ged2doc.size.Size*] New topmost box position, accepts anything convertible to *ged2doc.size.Size*.

4.1.2 ged2doc.ancestor_tree_emf

Module containing methods/classes for laying out ancestor trees.

Classes

EMFTreeVisitor(width, height[, dpi])	*AncestorTreeVisitor* implementation which makes EMF image.

class ged2doc.ancestor_tree_emf.**EMFTreeVisitor**(*width*, *height*, *dpi=300*)
Bases: *ged2doc.ancestor_tree.AncestorTreeVisitor*

AncestorTreeVisitor implementation which makes EMF image.

Parameters

width, height [*ged2doc.size.Size*] Width and height of the image.

dpi [float] Image resolution.

Methods

makeEMF()	Produce EMF image from a visited tree.
visitFatherEdge(node, parentNode)	Visitor method for an edge leading from node to its mother.
visitMotherEdge(node, parentNode)	Visitor method for an edge leading from node to its mother.
visitNode(node)	Visitor method for a node in tree.

visitNode(*node*)
Visitor method for a node in tree.

Parameters

node [TreeNode] Tree node.

visitMotherEdge(*node*, *parentNode*)
Visitor method for an edge leading from node to its mother.

It is guaranteed that *visitNode* is called for both nodes before this method is called.

Parameters

node [TreeNode] Tree node.

parentNode [TreeNode] Parent tree node.

visitFatherEdge(*node*, *parentNode*)
Visitor method for an edge leading from node to its mother.

It is guaranteed that *visitNode* is called for both nodes before this method is called.

Parameters

node [TreeNode] Tree node.

parentNode [TreeNode] Parent tree node.

makeEMF()
Produce EMF image from a visited tree.

Returns

document [bytes] Concents of generated EMF image.

mime_type [str] MIME type of produced document.

width [*ged2doc.size.Size*] Width of SVG document

height [*ged2doc.size.Size*] Height of SVG document

_abc_impl = <_abc_data object>

4.1.3 ged2doc.ancestor_tree_svg

Module containing methods/classes for laying out ancestor trees.

Classes

SVGTreeVisitor([units, fullxml])	*AncestorTreeVisitor* implementation which makes SVG plots.

class ged2doc.ancestor_tree_svg.**SVGTreeVisitor**(*units='in'*, *fullxml=True*)

Bases: *ged2doc.ancestor_tree.AncestorTreeVisitor*

AncestorTreeVisitor implementation which makes SVG plots.

Parameters

units [str] Type of dimension units for output SVG document.

fullxml [bool, optional] If True then generate full XML header.

Methods

makeSVG(width, height)	Produce SVG document from a visited tree.
visitFatherEdge(node, parentNode)	Visitor method for an edge leading from node to its mother.
visitMotherEdge(node, parentNode)	Visitor method for an edge leading from node to its mother.
visitNode(node)	Visitor method for a node in tree.

visitNode(*node*)

Visitor method for a node in tree.

Parameters

node [TreeNode] Tree node.

visitMotherEdge(*node*, *parentNode*)

Visitor method for an edge leading from node to its mother.

It is guaranteed that *visitNode* is called for both nodes before this method is called.

Parameters

node [TreeNode] Tree node.

parentNode [TreeNode] Parent tree node.

visitFatherEdge(*node*, *parentNode*)

Visitor method for an edge leading from node to its mother.

It is guaranteed that *visitNode* is called for both nodes before this method is called.

Parameters

node [TreeNode] Tree node.

parentNode [TreeNode] Parent tree node.

makeSVG(*width*, *height*)
Produce SVG document from a visited tree.

Parameters

width [*ged2doc.size.Size*] Width of SVG document

height [*ged2doc.size.Size*] Height of SVG document

Returns

document [str] Concents of generated SVG document.

mime_type [str] MIME type of produced document.

width [*ged2doc.size.Size*] Width of SVG document

height [*ged2doc.size.Size*] Height of SVG document

_textbox_svg(*textbox*, *textclass=None*, *units='in'*, *rect_style=None*)
Produces list of SVG elements for a textbox.

_abc_impl = <_abc_data object>

4.1.4 ged2doc.cli

Console script for ged2doc.

Functions

main([args])	Console script for ged2doc.

ged2doc.cli.**_make_writer**(*args=None*)
Make Writer instance based on command line arguments.

Parameters

args [list [str]] List of command line arguments passed to argparse, optional, by default uses sys.argv.

Returns

args [argparse.Namespace] Parsed command line arguments.

writer [*ged2doc.writer.Writer*] Instance of *Writer* to use for writing output file.

ged2doc.cli.**main**(*args=None*)
Console script for ged2doc.

Parameters

args [list [str], optional] Command line arguments, be default sys.argv is used.

4.1.5 ged2doc.dumbemf

Python module for generating EMF.

Only the most trivial features are implemented, stuff that is required by ged2doc package.

Classes

BackgroundMode()	
EMF(width, height)	Class for EMF, top-level structure.
GeneralRecord(type, *pack_args)	Base class for all EMF records.
MapMode()	
PenStyle()	
Record()	Base class for all EMF records.
StockObjects()	

class ged2doc.dumbemf.**EMF**(*width*, *height*)

Bases: object

Class for EMF, top-level structure.

Parameters

width, height [*ged2doc.size.Size*] Document width and height, accepts anything convertible to *ged2doc.size.Size*.

Methods

data()	Produce complete EMF structure.
polyline(points)	Draw polyline.
rectangle(left, top, right, bottom)	Draw rectangle.
set_bkmode(mode)	Set background mode.
text(x, y, text)	Draw text.
text_align([align_mode])	Set text alignment for next text drawing operation
text_color(color)	Set text color for next text drawing operation
use_font(size[, fontname])	Context manager which sets font parameters.
use_pen(style, width, color)	Context manager which sets pen parameters.

data()

Produce complete EMF structure.

Returns

data [bytes] Byte-string with EMF data.

use_pen(*style*, *width*, *color*)

Context manager which sets pen parameters.

Parameters

style [str] Pen style.

width [*ged2doc.size.Size*] Pen width.

color [int] Pen color.

use_font(*size*, *fontname='Times New Roman'*)
Context manager which sets font parameters.

Parameters

size [*ged2doc.size.Size*] Font size.

fontname [str] Font family name.

set_bkmode(*mode*)
Set background mode.

Parameters

mode [int] Mode, one of *BackgroundMode* constants.

polyline(*points*)
Draw polyline.

Parameters

points [list [tuple]] List of 2-tuples with (x, y) coordinates, each coordinate is *ged2doc.size.Size*.

rectangle(*left*, *top*, *right*, *bottom*)
Draw rectangle.

Parameters

left, top, right, bottom [*ged2doc.size.Size*] Rectangle coordinates.

text_align(*align_mode='c'*)
Set text alignment for next text drawing operation

Parameters

align_mode [str, optional] One of "l", "c", "r".

text_color(*color*)
Set text color for next text drawing operation

Parameters

color [int]

text(*x*, *y*, *text*)
Draw text.

Parameters

x, y [*ged2doc.size.Size*] Text coordinates.

text [str] Text to draw.

class ged2doc.dumbemf.**BackgroundMode**
Bases: object

TRANSPARENT = 1

OPAQUE = 0

4.1.6 ged2doc.dumbsvg

Python module for generating SVG.

Only the most trivial features are implemented, stuff that is required by ged2doc package.

Classes

Doc(width, height)	Class for SVG document, top-level structure.
Element(tag[, attributes, value])	Base class for all SVG elements.
Hyperlink(href)	Class for SVG "a" element.
Line(x1, y1, x2, y2[, style])	Class for SVG line element.
Rect(x, y, width, height[, style])	Class for SVG rect element.
Text([value, font_size, text_anchor, style, ...])	Class for SVG text element.
Tspan(x, y[, value])	Class for SVG tspan element.

class ged2doc.dumbsvg.**Doc**(*width*, *height*)

Bases: object

Class for SVG document, top-level structure.

Parameters

width, height [int or str] Document width and height, int for pixels or string.

Methods

add(element)	Add new element to the document.
xml([full_xml])	Produce XML representation of the document.

add(*element*)

Add new element to the document.

Parameters

element [*Element*] Element to add.

xml(*full_xml=True*)

Produce XML representation of the document.

Parameters

full_xml [bool, optional] If True then proper XML header is added.

Returns

doc [str] String containing XML.

class ged2doc.dumbsvg.**Element**(*tag*, *attributes=None*, *value=''*)

Bases: object

Base class for all SVG elements.

Parameters

tag [str] SVG tag name

attributes :`list` [`tuple`], optional List of tuples (attribute, attr_value).

value [str, optional] Element value (text).

Methods

add(element)	Add new sub-element to the element.
xml()	Produce XML fragment for this element.

add(*element*)
Add new sub-element to the element.

Parameters

element [*Element*] Element to add.

xml()
Produce XML fragment for this element.

Returns

xml [str] String containing XML fragment.

class ged2doc.dumbsvg.**Line**(*x1*, *y1*, *x2*, *y2*, *style=None*)
Bases: *ged2doc.dumbsvg.Element*

Class for SVG line element.

Parameters

x1, y1, x2, y2 [str] Coordinates of line ends.

style [str, optional] Line style.

Methods

add(element)	Add new sub-element to the element.
xml()	Produce XML fragment for this element.

class ged2doc.dumbsvg.**Rect**(*x*, *y*, *width*, *height*, *style=None*)
Bases: *ged2doc.dumbsvg.Element*

Class for SVG rect element.

Parameters

x, y [str] Coordinates of top left corner of the box.

width, height [str] Width and height of the box.

style [str, optional] Line style.

Methods

add(element)	Add new sub-element to the element.
xml()	Produce XML fragment for this element.

class ged2doc.dumbsvg.**Text**(*value=''*, *font_size=None*, *text_anchor=None*, *style=None*, *class_=None*)

Bases: *ged2doc.dumbsvg.Element*

Class for SVG text element.

Parameters

value [str, optional] Text to display.

font_size [str, optional] Font size for text.

text_anchor [str] Text anchor property.

style [str, optional] Text style.

class_ [str, optional] Text CSS class.

Methods

add(element)	Add new sub-element to the element.
xml()	Produce XML fragment for this element.

class ged2doc.dumbsvg.**Tspan**(*x*, *y*, *value=''*)

Bases: *ged2doc.dumbsvg.Element*

Class for SVG tspan element.

Parameters

x, y [str] Coordinates of the box.

value [str, optional] Text to display.

Methods

add(element)	Add new sub-element to the element.
xml()	Produce XML fragment for this element.

class ged2doc.dumbsvg.**Hyperlink**(*href*)

Bases: *ged2doc.dumbsvg.Element*

Class for SVG "a" element.

Parameters

href [str] Hyperlink value.

Methods

add(element)	Add new sub-element to the element.
xml()	Produce XML fragment for this element.

4.1.7 ged2doc.events

Utilities related to individual or family events.

Functions

family_events(family[, tags])	Returns all events for a given family.
indi_attributes(person[, tags])	Returns all attributes for a given individual.
indi_events(person[, tags])	Returns all events for a given individual.

Classes

Event(tag, value, type, date, place, note, cause)	Class representing GEDCOM event structure.

class ged2doc.events.**Event**(*tag: str*, *value: str*, *type: str*, *date: ged4py.model.Date*, *place: str*, *note: str*, *cause: str*)

Bases: tuple

Class representing GEDCOM event structure.

This is a reflection of <EVENT_DETAIL>, but only relevant pieces appear in this class.

Attributes

tag [str] Alias for field number 0

value [str, optional] Alias for field number 1

type [str, optional] Alias for field number 2

date [ged4py.model.Date, optional] Alias for field number 3

place [str, optional] Alias for field number 4

note [str, optional] Alias for field number 5

cause [str, optional] Alias for field number 6

Methods

count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	Return first index of value.

property tag

GEDCOM tag name for the event.

property value

GEDCOM record value, optional (`str` or `None`)

property type
GEDCOM event type, optional (`str` or `None`)

property date
Event date, optional (`ged4py.model.Date` or `None`)

property place
Place where event happened, optional (`str` or `None`)

property note
Arbitrary text note, optional (`str` or `None`)

property cause
What caused the event, optional (`str` or `None`)

_asdict()
Return a new OrderedDict which maps field names to their values.

_field_defaults = {}

_field_types = {'cause': <class 'str'>, 'date': <class 'ged4py.model.Date'>, 'note':

_fields = ('tag', 'value', 'type', 'date', 'place', 'note', 'cause')

_fields_defaults = {}

classmethod _make(*iterable*)
Make a new Event object from a sequence or iterable

_replace(***kwds*)
Return a new Event object replacing specified fields with new values

ged2doc.events.**indi_events**(*person*, *tags=None*)
Returns all events for a given individual.

Parameters

person [`ged4py.model.Individual`] GEDCOM INDI record.

tags [`list` [`str`], optional] Set of tags to return, default is all event tags.

Returns

events [`list` [*Event*]] List of events.

ged2doc.events.**indi_attributes**(*person*, *tags=None*)
Returns all attributes for a given individual.

Parameters

person [`ged4py.model.Individual`] GEDCOM INDI record.

tags [`list` [`str`], optional] Set of tags to return, default is all attribute tags.

Returns

events [`list` [*Event*]] List of events.

ged2doc.events.**family_events**(*family*, *tags=None*)
Returns all events for a given family.

Parameters

family [`ged4py.model.Record`] GEDCOM FAM record.

tags [`list` [`str`], optional] Set of tags to return, default is all attribute tags.

Returns

events [list [*Event*]] List of events.

4.1.8 ged2doc.html_writer

Module which produces HTML output.

Functions

TR(x)	This is no-op function, only used to mark translatable strings, to extract all strings run `pygettext -k TR ...`

Classes

HtmlWriter(flocator, output, tr[, encoding, ...])	Transforms GEDCOM file into nicely formatted HTML page.

class ged2doc.html_writer.**HtmlWriter**(*flocator*, *output*, *tr*, *encoding=None*, *encoding_errors='strict'*, *sort_order=<NameOrder.SURNAME_GIVEN: 'last+first'>*, *name_fmt=0*, *make_images=True*, *make_stat=True*, *make_toc=True*, *events_without_dates=True*, *page_width='800px'*, *image_width='300px'*, *image_height='300px'*, *image_upscale=False*, *tree_width=4*)

Bases: *ged2doc.writer.Writer*

Transforms GEDCOM file into nicely formatted HTML page.

This is a sub-class of *Writer* class providing implementation for rendering methods which transform GEDCOM info into HTML constructs. Constructor takes a large number of arguments which configure appearance of the resulting HTML page. After instantiating an object of this type one has to call *save()* method to produce output file.

Parameters

flocator [*ged2doc.input.FileLocator*] File locator instance.

output [str or io.TextIOBase] Name for the output file or file object.

tr [*ged2doc.i18n.I18N*] Object supporting translation.

encoding [str, optional] GEDCOM file encoding, if None then encoding is determined from file itself.

encoding_errors [str, optional] Controls error handling behavior during string decoding, one of "strict" (default), "ignore", or "replace".

sort_order [ged4py.model.NameOrder, optional] Determines ordering of person in output file, one of the constants defined in ged4py.model.NameOrder enum.

name_fmt [int, optional] Bit mask with flags from *ged2doc.name* module.

make_images [bool, optional] If True (default) then generate images for persons.

make_stat [bool, optional] If True (default) then generate statistics section.

make_toc [bool, optional] If True (default) then generate Table of Contents.

events_without_dates [bool, optional] If True (default) then show events that have no associated dates.

page_width [*ged2doc.size.Size*] Width of the produced HTML page.

image_width [*ged2doc.size.Size*] Size of the images.

image_height [*ged2doc.size.Size*] Size of the images.

image_upscale [bool] If True then smaller images will be re-scaled to extend to image size.

tree_width [int] Number of generations in ancestor tree.

Methods

save()	Produce output document.

_render_prolog()
Generate initial document header/title.

_interpolate(*text*)
Takes text with embedded references and returns properly escaped text with HTML links.

Parameters

text [str] Arbitrary text with references.

Returns

html [str] HTML as text.

_render_section(*level*, *ref_id*, *title*, *newpage=False*)
Produces new section in the output document.

This method should also save section reference so that TOC can be later produced when *_render_toc* method is called.

Parameters

level [int] Section level (1, 2, 3, etc.).

ref_id [str] Unique section identifier.

title [str] Printable section name.

newpage [bool, optional] If True then start new page (for documents that support pagination).

_render_person(*person*, *image_data*, *attributes*, *families*, *events*, *notes*)
Output person information.

Parameters

person [ged4py.model.Individual] INDI record representation.

image_data [bytes or None] Either None or binary image data (typically content of JPEG image).

attributes [list [tuple]] List of (attr_name, text) tuples, may be empty.

families [list [str]] List of strings (possibly empty), each string contains description of one family and should be typically rendered as a separate paragraph.

events [list [tuple]] List of (date, text) tuples, may be empty. Date is properly formatted string and does not need any other formatting.

notes [list [str]] List of strings, each string should be rendered as separate paragraph.

Notes

Textual information in parameters to this method can include references to other persons (e.g. mother/father). Such references are embedded into text in encoded format determined by _person_ref method. It is responsibility of the subclasses to extract these references from text and re-encode them using proper backend representation.

_render_name_stat(*n_total*, *n_females*, *n_males*)
Produces summary table.

Sum of male and female counters can be lower than total count due to individuals with unknown/unspecified gender.

Parameters

n_total [int] Total number of individuals.

n_females [int] Number of female individuals.

n_males [int] Number of male individuals.

_render_name_freq(*freq_table*)
Produces name statistics table.

Parameters

freq_table [list [tuple]] List of (name, count) tuples.

_render_toc()
Produce table of contents using info collected in *_render_section()*.

_finalize()
Finalize output.

_abc_impl = <_abc_data object>

_get_image_fragment(*image_data*)
Returns <img> HTML fragment for given image data (byte array).

Parameters

image_data [bytes] Image data.

Returns

html [str] HTML text containing image.

_make_ancestor_tree(*person*)
Make SVG picture for parent tree.

Parameters

person [ged4py.model.Individual] INDI record

Returns

html [list [str]] SVG data (HTML contents), list of strings.

4.1.9 ged2doc.i18n

Python module responsible for internationalization of ged2doc.

It covers all aspects that are language- or locale-dependent. In particular it does these things:

- translates short string messages from application language into output language
- translates dates into printable format according to locale preferences

Note that we do not use system locale, instead we expect client to provide small set of configuration options such as output language and date format.

Functions

TR(x)	This is no-op function, only used to mark translatable strings, to extract all strings run `pygettext -k TR ...`

Classes

I18N(lang[, datefmt, domain])	Class with methods responsible for various aspects of translations.

ged2doc.i18n.**TR**(*x*)
This is no-op function, only used to mark translatable strings, to extract all strings run `pygettext -k TR ...`

class ged2doc.i18n.**_NullFallback**
Bases: `object`

Special fallback class for gettext which returns None for missing translations.

Methods

gettext	
ugettext	

gettext(*message*)

ugettext(*message*)

class ged2doc.i18n.**_TemplateDateVisitor**
Bases: `ged4py.date.DateValueVisitor`

Visitor class that builds template strings and keywords from dates.

Methods

visitAbout(date)	Visit an instance of DateValueAbout type.
visitAfter(date)	Visit an instance of DateValueAfter type.
visitBefore(date)	Visit an instance of DateValueBefore type.
visitCalculated(date)	Visit an instance of DateValueCalculated type.
visitEstimated(date)	Visit an instance of DateValueEstimated type.
visitFrom(date)	Visit an instance of DateValueFrom type.
visitInterpreted(date)	Visit an instance of DateValueInterpreted type.
visitPeriod(date)	Visit an instance of DateValuePeriod type.
visitPhrase(date)	Visit an instance of DateValuePhrase type.
visitRange(date)	Visit an instance of DateValueRange type.
visitSimple(date)	Visit an instance of DateValueSimple type.
visitTo(date)	Visit an instance of DateValueTo type.

visitSimple(*date*)
Visit an instance of DateValueSimple type.

Parameters

date [DateValueSimple] Date value instance.

Returns

value [object] Implementation of this method can return anything, value will be returned from DateValue.accept() method.

visitPeriod(*date*)
Visit an instance of DateValuePeriod type.

Parameters

date [DateValuePeriod] Date value instance.

Returns

value [object] Implementation of this method can return anything, value will be returned from DateValue.accept() method.

visitFrom(*date*)
Visit an instance of DateValueFrom type.

Parameters

date [DateValueFrom] Date value instance.

Returns

value [object] Implementation of this method can return anything, value will be returned from DateValue.accept() method.

visitTo(*date*)
Visit an instance of DateValueTo type.

Parameters

date [DateValueTo] Date value instance.

Returns

value [`object`] Implementation of this method can return anything, value will be returned from `DateValue.accept()` method.

visitRange(*date*)
Visit an instance of `DateValueRange` type.

Parameters

date [`DateValueRange`] Date value instance.

Returns

value [`object`] Implementation of this method can return anything, value will be returned from `DateValue.accept()` method.

visitBefore(*date*)
Visit an instance of `DateValueBefore` type.

Parameters

date [`DateValueBefore`] Date value instance.

Returns

value [`object`] Implementation of this method can return anything, value will be returned from `DateValue.accept()` method.

visitAfter(*date*)
Visit an instance of `DateValueAfter` type.

Parameters

date [`DateValueAfter`] Date value instance.

Returns

value [`object`] Implementation of this method can return anything, value will be returned from `DateValue.accept()` method.

visitAbout(*date*)
Visit an instance of `DateValueAbout` type.

Parameters

date [`DateValueAbout`] Date value instance.

Returns

value [`object`] Implementation of this method can return anything, value will be returned from `DateValue.accept()` method.

visitCalculated(*date*)
Visit an instance of `DateValueCalculated` type.

Parameters

date [`DateValueCalculated`] Date value instance.

Returns

value [`object`] Implementation of this method can return anything, value will be returned from `DateValue.accept()` method.

visitEstimated(*date*)
Visit an instance of `DateValueEstimated` type.

Parameters

date [DateValueEstimated] Date value instance.

Returns

value [object] Implementation of this method can return anything, value will be returned from DateValue.accept() method.

visitInterpreted(*date*)
Visit an instance of DateValueInterpreted type.

Parameters

date [DateValueInterpreted] Date value instance.

Returns

value [object] Implementation of this method can return anything, value will be returned from DateValue.accept() method.

visitPhrase(*date*)
Visit an instance of DateValuePhrase type.

Parameters

date [DateValuePhrase] Date value instance.

Returns

value [object] Implementation of this method can return anything, value will be returned from DateValue.accept() method.

_abc_impl = <_abc_data object>

class ged2doc.i18n.**I18N**(*lang*, *datefmt=None*, *domain='ged2doc'*)
Bases: object

Class with methods responsible for various aspects of translations.

Parameters

lang [str] Output language such as "en", "ru".

datefmt [str, optional] Printable date format.

domain [str, optional] gettext domain (message file name).

Methods

tr(text[, gender])	Translates given text, takes into account gender.
tr_date(date)	Produce language-specific date representation.

tr(*text*, *gender=None*)
Translates given text, takes into account gender.

Parameters

text [str] Text to translate.

gender [str, optional] One of 'F', 'M', 'U', or None.

Returns

text [str] Translated text.

tr_date(*date*)
Produce language-specific date representation.

Parameters

date [ged4py.date.DateValue]

Returns

text_date [str] String representation of a date.

_tr_cal_date(*date*)
Produce language-specific calendar date representation.

Uses date format provided in constructor, month name is translated into a destination language.

Parameters

date [ged4py.date.CalendarDate]

Returns

text_date [str] String representation of a date.

_monthName(*month*)
Returns translation of a month name.

For a given GEDCOM month name return translated month name.

Parameters

month [str] Month name in GEDCOM format.

Returns

month [str] Name of this month in destination language.

4.1.10 ged2doc.input

Module which handles input files.

This module is responsible for locating all files (GEDCOM data and images) given the application inputs. Currently it handles two cases:

- Input is specified as path to GEDCOM file, that file can contain names of image files that are either absolute or relative to directory containing GEDCOM file or some other directory. Program options can specify directory where images are located.
- Input file is a ZIP archive that includes both GEDCOM file and files with images. Depending on how GEDCOM file and archive were prepared names of image files in GEDCOM file can be specified as absolute paths to their original location or relative paths to their common directory.

Additional issue to consider is that files can be prepared on a system which is different from the system where the file is parsed. For example GEDCOM file could be prepared on Windows machine and names of image files could be given using Windows path convention (either absolute as `C:\Users\JosephSmith\Documents\Pictures\Family\Tree\Me.BMP` or relative as `Pictures\Family\Tree\Me.BMP`) and later this GEDCOM file could be copied to Linux host and processed using `ged2doc` package. Files on Linux machine will have different absolute and possibly relative paths (and definitely different path separator character).

In case of ZIP archive the names of images in GEDCOM file could be different from the names in in the archive (e.g. image path in GEDCOM file `C:\Users\JosephSmith\Documents\Pictures\Family\Tree\Me.BMP` could be stored in ZIP archive as `Pictures/Family/Tree/Me.BMP`).

Logic in this module is supposed to handle all those possible cases where names of files in GEDCOM file could be different from their location on a target storage system.

Typical use cases for GEDCOM file returned by this module is to be passed to methods in `ged4py` package and that package expects true filesystem-backed file which supports `seek()` and `tell()` methods. Image files do not typically need support for these methods and are usually read as a byte stream using `read()` method. This module returns seek-able file object open in binary mode for GEDCOM file (meaning that temporary file on disk may need to be created in some cases) and a "simple" binary stream for images.

Functions

make_file_locator(input_file, ...)	Create and return file locator instance

Classes

FileLocator()	Abstract interface for file locator instances.

Exceptions

MultipleMatchesError	Class for exceptions generated when there is more than one file matching specified criteria.

ged2doc.input.**make_file_locator**(*input_file*, *file_name_pattern*, *image_path*)

Create and return file locator instance

For a given input file (which can be GEDCOM file or ZIP archive) return corresponding file locator object (instance of *FileLocator* type).

Parameters

input_file Path of the input file or file object, can be a ZIP archive or a GEDCOM file. If argument is a file object then it must support `seek()` method and be open in a binary mode.

file_name_pattern [`str`] If input file is a ZIP archive then this pattern is used to search for a GEDCOM file in archive. Could be `"*.ged"` for example or can include more specific pattern.

image_path [`str`] Directory on a filesystem where images are found. Images could be located in sub-directories of the given path. If `file_name` is a ZIP archive then images are searched inside ZIP archive and then in `image_path`. If `image_path` is `None` then filesystem is not searched for files. If `image_path` is an empty string then current directory is searched.

Returns

locator [*FileLocator*] File locator instance.

Raises

OSError Raised if file is not found.

AttributeError Raised if file object is given as input file but it does not support `seek()` method.

class ged2doc.input.**FileLocator**
Bases: object

Abstract interface for file locator instances.

Methods

open_gedcom()	Returns file object for the input GEDCOM file.
open_image(name)	Returns open file object for the named image file.

abstract open_gedcom()
Returns file object for the input GEDCOM file.

If no GEDCOM file is found None is returned. If more than one file is found than *MultipleMatchesError* exception is raised. Can throw other exceptions, e.g. if file cannot be open.

Returned file object will be open in binary mode and will support seek() and tell() methods. Note that this may be a temporary file which will be deleted after file is closed.

Returns

file File object open in binary mode supporting seek() and tell() methods.

Raises

MultipleMatchesError Raised if more than one file file is found.

abstract open_image(*name*)
Returns open file object for the named image file.

If image file is not found None is returned. If more than one matching file is found than *MultipleMatchesError* exception is raised. Can throw other exceptions if file cannot be open.

Note that this file object may not support all operations (it may be an object inside zip archive for example) so you may need to copy it if you want full file protocol support.

Parameters

name [str] Name of the image file to open. This can be relative or absolute path name. Usually this is the name that is stored in GEDCOM file and it can use separator character which is different from a system reading this file.

Returns

image File object open in binary mode, only read() method is guaranteed to work.

Raises

MultipleMatchesError Raised if more than one file is found.

_abc_impl = <_abc_data object>

exception ged2doc.input.**MultipleMatchesError**
Bases: RuntimeError

Class for exceptions generated when there is more than one file matching specified criteria.

4.1.11 ged2doc.name

Methods for manipulating/formatting names.

Functions

name_fmt(name[, fmt])	Format name for output.

Classes

NameFormat(value)	Names can be rendered in different formats, this enum defines different types of presentations that can also be combined with logical OR (|).

class ged2doc.name.**NameFormat**(*value*)

Bases: enum.Flag

Names can be rendered in different formats, this enum defines different types of presentations that can also be combined with logical OR (|).

SURNAME_FIRST = 1

Bit flag for surname-first format (e.g. Smith Jane).

COMMA = 2

Bit flag for adding comma in surname-first format (e.g. Smith, Jane).

MAIDEN = 4

Bit flag for adding maiden name (e.g. Jane Smith (Sawyer)).

MAIDEN_ONLY = 8

Bit flag for using maiden name only (e.g. Jane Sawyer).

CAPITAL = 16

Bit flag for rendering surname in capital (e.g. Jane SMITH).

ged2doc.name.**name_fmt**(*name*, *fmt=<NameFormat.0: 0>*)

Format name for output.

Parameters

name [ged4py.model.Name] Person name.

fmt [*NameFormat*, optional] Bitmask combination of *NameFormat* flags.

Returns

name [str] Formatted name representation.

4.1.12 ged2doc.odt_writer

Module which produces ODT output.

Functions

TR(x)	This is no-op function, only used to mark translatable strings, to extract all strings run `pygettext -k TR ...`

Classes

OdtWriter(flocator, output, tr[, encoding, ...])	Transforms GEDCOM file into nicely formatted OpenDocument Text (ODT).
PageLayout(width, height, left, right, top, ...)	Class representing page layout, size and margins

class ged2doc.odt_writer.**OdtWriter**(*flocator*, *output*, *tr*, *encoding=None*, *encoding_errors='strict'*, *sort_order=<NameOrder.SURNAME_GIVEN: 'last+first'>*, *name_fmt=0*, *make_images=True*, *make_stat=True*, *make_toc=True*, *events_without_dates=True*, *page_width='6in'*, *page_height='9in'*, *margin_left='0.5in'*, *margin_right='0.5in'*, *margin_top='0.5in'*, *margin_bottom='0.25in'*, *image_width='2in'*, *image_height='2in'*, *tree_width=4*, *first_page=1*, *tree_format='emf'*)

Bases: *ged2doc.writer.Writer*

Transforms GEDCOM file into nicely formatted OpenDocument Text (ODT).

This is a sub-class of *Writer* class providing implementation for rendering methods which transform GEDCOM info into ODT structures. Constructor takes a large number of arguments which configure appearance of the resulting HTML page. After instantiating an object of this type one has to call *save* method to produce output file.

Parameters

flocator [*ged2doc.input.FileLocator*] File locator instance.

output [str or io.TextIOBase] Name for the output file or file object.

tr [*ged2doc.i18n.I18N*] Object supporting translation.

encoding [str, optional] GEDCOM file encoding, if None then encoding is determined from file itself.

encoding_errors [str, optional] Controls error handling behavior during string decoding, one of "strict" (default), "ignore", or "replace".

sort_order [ged4py.model.NameOrder, optional] Determines ordering of person in output file, one of the constants defined in ged4py.model.NameOrder enum.

name_fmt [int, optional] Bit mask with flags from *ged2doc.name* module.

make_images [bool, optional] If True (default) then generate images for persons.

make_stat [bool, optional] If True (default) then generate statistics section.

make_toc [bool, optional] If True (default) then generate Table of Contents.

events_without_dates [bool, optional] If True (default) then show events that have no associated dates.

page_width, page_height [*ged2doc.size.Size*, optional] Page size of the produced document.

margin_left, margin_right, margin_top, margin_bottom [*ged2doc.size.Size*, optional] Page margins of the produced document.

image_width, image_height [*ged2doc.size.Size*, optional] Size of the images.

tree_width [int] Number of generations in ancestor tree.

first_page [int] Number of the first generated page.

tree_format [str] Image format for ancestor tree, "emf" or "svg".

Methods

save()	Produce output document.

_make_layout(*doc*, *layout*, *firstpage*)
Set paper dimensions.

Parameters

doc [OpenDocumentText] ODT document object.

layout [PageLayout] ODT page layout.

firstpage [int] Number of the first generated page.

_make_styles(*doc*, *layout*)
Generate set of styles for the document.

Parameters

doc [OpenDocumentText] ODT document object.

layout [PageLayout] ODT page layout.

_interpolate(*text*)
Takes text with embedded references and returns text.

Parameters

text [str] Arbitrary text with references.

Returns

text [str] Resulting text.

_render_prolog()
Generate initial document header/title.

_render_section(*level*, *ref_id*, *title*, *newpage=False*)
Produces new section in the output document.

This method should also save section reference so that TOC can be later produced when *_render_toc* method is called.

Parameters

level [int] Section level (1, 2, 3, etc.).

ref_id [str] Unique section identifier.

title [str] Printable section name.

newpage [bool, optional] If True then start new page (for documents that support pagination).

_render_person(*person*, *image_data*, *attributes*, *families*, *events*, *notes*)
Output person information.

Parameters

person [ged4py.model.Individual] INDI record representation.

image_data [bytes or None] Either None or binary image data (typically content of JPEG image).

attributes [list [tuple]] List of (attr_name, text) tuples, may be empty.

families [list [str]] List of strings (possibly empty), each string contains description of one family and should be typically rendered as a separate paragraph.

events [list [tuple]] List of (date, text) tuples, may be empty. Date is properly formatted string and does not need any other formatting.

notes [list [str]] List of strings, each string should be rendered as separate paragraph.

Notes

Textual information in parameters to this method can include references to other persons (e.g. mother/father). Such references are embedded into text in encoded format determined by _person_ref method. It is responsibility of the subclasses to extract these references from text and re-encode them using proper backend representation.

_render_name_stat(*n_total*, *n_females*, *n_males*)
Produces summary table.

Sum of male and female counters can be lower than total count due to individuals with unknown/unspecified gender.

Parameters

n_total [int] Total number of individuals.

n_females [int] Number of female individuals.

n_males [int] Number of male individuals.

_render_name_freq(*freq_table*)
Produces name statistics table.

Parameters

freq_table [list [tuple]] List of (name, count) tuples.

_render_toc()
Produce table of contents using info collected in *_render_section()*.

_finalize()
Finalize output.

_get_image_fragment(*image_data*)
Adds Image to the document as person's picture.

Parameters

image_data [bytes] Image data.

Returns

frame [odf.draw.Frame] Frame containing image.

_make_ancestor_tree(*person*)
"Add a picture for ancestor tree.

Parameters

person [ged4py.model.Individual] INDI record

_abc_impl = <_abc_data object>

4.1.13 ged2doc.size

Module which defines class for manipulating size values.

Classes

Size([value, dpi])	Class for specifying size values.
String2Size([default_unit, accepted_units, ...])	Class implementing callable for conversion of strings to *Size*.

class ged2doc.size.**Size**(*value=0*, *dpi=None*)
Bases: object

Class for specifying size values.

Size can be specified as a number with units, supported units are pt (points), in (inches), cm (centimeters), mm (millimeters), and px (pixels). If units are not specified then inches are assumed.

Constructor converts input value to a size. If input value has numeric type then it is assumed to be size in inches. If input value is a string then it should be a floating number followed by optional suffix (one of pt, in, mm, cm, px). Without suffix the number gives size in inches. Constructor also accepts other Size instances as an argument which copies the size value (but can be use to specify different dpi value).

Class supports most of the regular numeric operators so it can be used as a numeric value (in inches) in expressions. Operator XOR (^) is used for formatting of the size values with the specified unit type, e.g.:

```
size = Size("144pt") / 2
print(size^"mm")            # will produce string "25.4mm"
```

Parameters

value [int, float, str, or *Size*] Input value for size.

dpi [float, optional] Dots-per-inch for converting pixels into other scale. Default is to use class attribute *dpi* value.

Raises

ValueError Raised if string does not have correct format.

TypeError Raised if input value has unsupported type.

Attributes

inches Size in inches, (`float`)

mm Size in millimeters, (`float`)

pt Size in points (`float`)

px Size in pixels, (`int`)

pxf Size in (fractional) pixels, (`float`)

Methods

to_dpi(dpi)	Return copy of itself with updated DPI value.

dpi = 96.0
Class constant used for pixels-to-inches conversion, default value is 96., it is used as default DPI for `Size` instances that do not specify explicit `dpi` argument

property pt
Size in points (`float`)

property px
Size in pixels, (`int`)

property pxf
Size in (fractional) pixels, (`float`)

property inches
Size in inches, (`float`)

property mm
Size in millimeters, (`float`)

to_dpi(*dpi*)
Return copy of itself with updated DPI value.

This is a convenience method which does the same as `Size(self, dpi)`.

_coerce(*other*)
Coerce other object to Size, use this object dpi if needed

class ged2doc.size.**String2Size**(*default_unit='in'*, *accepted_units=None*, *rejected_units=None*)
Bases: `object`

Class implementing callable for conversion of strings to *Size*.

This class defines restrictions on Size units, you can define set of accepted/rejected unit types. This could be useful for command line parser, e.g. as a `type` argument for `argparse` methods.

Parameters

default_unit [`str`] Default unit name to use when unit is not given.

accepted_units [`list` [`str`]] List of acceptable unit names, if string passed to `__call__` has unit not in this list then `ValueError` is raised. If `None` or empty list is passed to this argument then check is not performed.

rejected_units [`list [ str ]`] List of rejected unit names, if string passed to `__call__` has unit on this list then `ValueError` is raised. If `None` or empty list is passed to this argument then check is not performed.

Methods

`__call__`(value)	Implements operator().

all_units = ('pt', 'in', 'cm', 'mm', 'px')
All known unit names.

4.1.14 ged2doc.textbox

Module defining `TexBox` class and related methods.

Classes

TextBox([x0, y0, width, maxwidth, height, ...])	Class representing an SVG box with a text inside.

class ged2doc.textbox.**TextBox**(*x0=0*, *y0=0*, *width=0*, *maxwidth=0*, *height=0*, *text=''*, *font_size='10pt'*, *padding='4pt'*, *line_spacing='1.5pt'*, *href=None*)

Bases: `object`

Class representing an SVG box with a text inside.

This class takes care of the text wrapping and optional resizing of the box in vertical direction to fit all text.

Parameters

x0 [*Size*, optional] Lowest X coordinate of corner (def: 0)

y0 [*Size*, optional] Lowest Y coordinate of corner (def: 0)

width [*Size*, optional] Width of a box (def: 0)

maxwidth [*Size*, optional] Maximum width of a box (def: 0)

height [*Size*, optional] Height of a box (def: 0)

text [`str`, optional] Text contained in a box (def: '')

font_size [*Size*, optional] Font size (def: 10pt)

rect_style [`str`, optional] SVG style for rectangle

text_style [`str`] SVG style for text

line_spacing [*Size*, optional] Space between lines (def: 1.5pt)

padding [*Size*, optional] Box padding space (def: 4pt)

Attributes

font_size

height

href

lines

midx

midy

text

width

x0

x1

y0

y1

Methods

lines_pos()	Iterate over lines and their positions.
move(x0, y0)	Sets new coordinates fo x0 and y0
reflow()	Split the text inside the box so that it fits into box width, then recalculate box height so that all text fits inside the box.

property x0

property x1

property y0

property y1

property midx

property midy

property width

property height

property text

property href

property font_size

property lines

lines_pos()

Iterate over lines and their positions.

For each line of test iterator returns a tuple of two items:

- text for that line
- position as a tuple of two `Size` instances, for horizontal position it returns the center of the box (same as `midx`), and for vertical position it returns the baseline position of that line

Yields

line [`str`] Text for a line.

pos [`tuple` [`Size`]] Text position.

reflow()
Split the text inside the box so that it fits into box width, then recalculate box height so that all text fits inside the box.

move(*x0*, *y0*)
Sets new coordinates fo x0 and y0

Parameters

x0, y0 [int or Size] New box coordinates.

_splitText(*text*)
Tries to split a line of text into a number of lines which fit into box width. It honors embedded newlines, line will always be split at those first.

Parameters

text [str] Text to split into lines.

Returns

lines [list [str]]

_splitText1(*text*, *width*)
Tries to split a line of text into a number of lines which fit into box width.

_textWidth(*text*)
Calculates approximate width of the string of text.

4.1.15 ged2doc.utils

Various utility methods.

Functions

embed_ref(xref_id, name)	Returns encoded person reference.
img_mime_type(img)	Returns image MIME type or None.
img_resize(img, size)	Resize image to fit given size.
img_save(img, file)	Save image into output file.
languages()	Returns list of supported languages.
person_image_file(person)	Finds primary person image file name.
resize(size, max_size[, reduce_only])	Resize a box so that it fits into other box and keeps aspect ratio.
split_refs(text)	Split text with embedded references into a sequence of text and references.
system_lang()	Try to guess system language.

ged2doc.utils.**resize**(*size*, *max_size*, *reduce_only=True*)
Resize a box so that it fits into other box and keeps aspect ratio.

Parameters

size [tuple] Box to resize, (width, height). Elements of tuple are either numbers or *ged2doc.size.Size* instances.

max_size [tuple] Box to fit new resized box into, (width, height).

reduce_only [bool] If True (default) and size is smaller than max_size then return original

box.

Returns

width, height Tuple (width, height) representing resized box. Type of the elements is the same as the type of the `size` elements.

`ged2doc.utils.`**`person_image_file`**(*person*)

Finds primary person image file name.

Scans INDI's OBJE records and finds "best" FILE record from those.

Parameters

person [`ged4py.model.Individual`] INDI record representation.

Returns

file_name [`str` or `None`] String with file name or `None`.

Notes

OBJE record contains one (in 5.5) or few (in 5.5.1) related multimedia files. In 5.5 file contents can be embedded as BLOB record though we do not support this. In 5.5.1 file name is stored in a record.

In 5.5.1 OBJE record is supposed to have structure:

```
OBJE
  +1 FILE <MULTIMEDIA_FILE_REFN>     {1:M}
    +2 FORM <MULTIMEDIA_FORMAT>      {1:1}
       +3 MEDI <SOURCE_MEDIA_TYPE>   {0:1}
  +1 TITL <DESCRIPTIVE_TITLE>        {0:1}
  +1 _PRIM {Y|N}                     {0:1}
```

Some applications which claim to be 5.5.1 version still store OBJE record in 5.5-like format:

```
OBJE
  +1 FILE <MULTIMEDIA_FILE_REFN>     {1:1}
  +1 FORM <MULTIMEDIA_FORMAT>        {1:1}
  +1 TITL <DESCRIPTIVE_TITLE>        {0:1}
  +1 _PRIM {Y|N}                     {0:1}
```

This method returns the name of the FILE corresponding to _PRIM=Y, or if there is no _PRIM record then the first FILE record. Potentially we also need to look at MEDI record to only chose image type, but I have not seen examples of MEDI use yet, so for now I only select FORM which correspond to images.

`ged2doc.utils.`**`languages`**()

Returns list of supported languages.

This should correspond to the existing translations and needs to be updated when new translation is added.

Returns

languages [`list` [`str`]]

`ged2doc.utils.`**`system_lang`**()

Try to guess system language.

Returns

language [`str`] Guessed system language, "en" is returned as a fallback.

ged2doc.utils.**embed_ref**(*xref_id*, *name*)
Returns encoded person reference.

Encoded reference consists of ASCII character SOH (0x01) followed by reference ID, STX (0x02), person name, and ETX (0x03).

Parameters

xref_id [str] Reference ID for a person.

name [str] Person name.

ged2doc.utils.**split_refs**(*text*)
Split text with embedded references into a sequence of text and references.

Reference is returned as tuple (id, name).

Yields

item [str or tuple] Pieces of text and references.

ged2doc.utils.**img_mime_type**(*img*)
Returns image MIME type or None.

Parameters

img: `PIL.Image` PIL Image object.

Returns

mime_type [str] MIME string like "image/jpg" or None.

ged2doc.utils.**img_resize**(*img*, *size*)
Resize image to fit given size.

Image is resized only if it is larger than size, otherwise unmodified image is returned.

Parameters

img [PIL.Image] PIL Image object.

size [tuple] Final image size (width, height)

Returns

image [PIL.Image] Resized image.

ged2doc.utils.**img_save**(*img*, *file*)
Save image into output file.

This method automatically chooses the best file format for output.

Parameters

img [PIL.Image] PIL Image object.

file File object to write output to.

Returns

mime_type [str] MIME type of the output image.

4.1.16 ged2doc.writer

Module which defines base class for all writer classes.

Functions

`TR`(x)	This is no-op function, only used to mark translatable strings, to extract all strings run `pygettext -k TR ...`

Classes

`Writer`(flocator, tr[, encoding, ...])	Base class for document writers.

class ged2doc.writer.**Writer**(*flocator*, *tr*, *encoding=None*, *encoding_errors='strict'*, *sort_order=<NameOrder.SURNAME_GIVEN: 'last+first'>*, *name_fmt=0*, *make_images=True*, *make_stat=True*, *make_toc=True*, *events_without_dates=True*)

Bases: `object`

Base class for document writers.

This class knows how to extract all relevant information from GEDCOM data and convert it into output document. It defines basic structure of the produced document (sequence of section and sub-sections) and it depends on the subclasses to implement specific rendering of output information into document-specific format. Subclasses will need to implement small set of methods (see _render methods below).

Parameters

flocator [*ged2doc.input.FileLocator*] File locator instance.

tr [*ged2doc.i18n.I18N*] Object supporting translation.

encoding [`str`, optional] GEDCOM file encoding, if `None` then encoding is determined from file itself.

encoding_errors [`str`, optional] Controls error handling behavior during string decoding, one of "strict" (default), "ignore", or "replace".

sort_order [`ged4py.model.NameOrder`, optional] Determines ordering of person in output file, one of the constants defined in `ged4py.model.NameOrder` enum.

name_fmt [`int`, optional] Bit mask with flags from *ged2doc.name* module.

make_images [`bool`, optional] If `True` (default) then generate images for persons.

make_stat [`bool`, optional] If `True` (default) then generate statistics section.

make_toc [`bool`, optional] If `True` (default) then generate Table of Contents.

events_without_dates [`bool`, optional] If `True` (default) then show events that have no associated dates.

Methods

save()	Produce output document.

save()
Produce output document.

This is the main (and the only one client-callable) method of the writers, it will parse GEDCOM structure and produce output document from it.

_indi_sort_key(*indi*)
Return name ordering key for individual.

Parameters

indi [ged4py.model.Individual] INDI record representation.

Returns

order [tuple [str]]

_events(*person*)
Returns a list of events for a given person.

Returned list contains tuples (date, info).

Parameters

person [ged4py.model.Individual] INDI record representation.

Returns

events [list [tuple]] List of tuples with two elements: date and event information.

_make_main_image(*person*)
Returns image for a person.

Parameters

person [ged4py.model.Individual] INDI record representation.

Returns

image_data [bytes or None] Bytes of the image data or None.

_name_freq(*people*)
Returns name frequency table.

Parameters

people [iterable of ged4py.model.Individual] Sequence of INDI records.

Returns

table [list [tuple]] List of (name, count) ordered by name.

_format_indi_attr(*person*, *attrib*, *prefix='ATTR.'*)
Formatting of the individual's attributes.

Parameters

person [ged4py.model.Individual] INDI record representation.

attrib [*ged2doc.events.Event*] Attribute structure.

prefix [str, optional] Prefix added to attribute tag before translation.

Returns

attribute [tuple] Tuple (attribute, value).

_person_ref (*person*, *name=None*)
Returns encoded person reference.

If person is None then None is returned. If name is not given then properly formatted person full name is used.

Encoded reference consists of ASCII character SOH () followed by reference ID, STX (), person name, and ETX (). This sequence will be embedded in the text and it should be interpreted later by subclasses to produce properly formatted reference in a backend- specific format.

Parameters

person [ged4py.model.Individual] INDI record representation.

name [str, optional] Name to use instead of person name.

Returns

person_ref [str]

abstract _render_prolog()
Generate initial document header/title.

abstract _render_section (*level*, *ref_id*, *title*, *newpage=False*)
Produces new section in the output document.

This method should also save section reference so that TOC can be later produced when *_render_toc* method is called.

Parameters

level [int] Section level (1, 2, 3, etc.).

ref_id [str] Unique section identifier.

title [str] Printable section name.

newpage [bool, optional] If True then start new page (for documents that support pagination).

abstract _render_person (*person*, *image_data*, *attributes*, *families*, *events*, *notes*)
Output person information.

Parameters

person [ged4py.model.Individual] INDI record representation.

image_data [bytes or None] Either None or binary image data (typically content of JPEG image).

attributes [list [tuple]] List of (attr_name, text) tuples, may be empty.

families [list [str]] List of strings (possibly empty), each string contains description of one family and should be typically rendered as a separate paragraph.

events [list [tuple]] List of (date, text) tuples, may be empty. Date is properly formatted string and does not need any other formatting.

notes [list [str]] List of strings, each string should be rendered as separate paragraph.

Notes

Textual information in parameters to this method can include references to other persons (e.g. mother/father). Such references are embedded into text in encoded format determined by *_person_ref* method. It is responsibility of the subclasses to extract these references from text and re-encode them using proper backend representation.

abstract _render_name_stat(*n_total*, *n_females*, *n_males*)
Produces summary table.

Sum of male and female counters can be lower than total count due to individuals with unknown/unspecified gender.

Parameters

n_total [int] Total number of individuals.

n_females [int] Number of female individuals.

n_males [int] Number of male individuals.

abstract _render_name_freq(*freq_table*)
Produces name statistics table.

Parameters

freq_table [list [tuple]] List of (name, count) tuples.

abstract _render_toc()
Produce table of contents using info collected in *_render_section()*.

abstract _finalize()
Finalize output.

_abc_impl = <_abc_data object>

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/andy-z/ged2doc/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

GEDCOM Utilities could always use more documentation, whether as part of the official GEDCOM Utilities docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/andy-z/ged2doc/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *ged2doc* for local development.

1. Fork the *ged2doc* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/ged2doc.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv ged2doc
$ cd ged2doc/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 ged2doc tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.6+. Check https://travis-ci.org/andy-z/ged2doc/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_ged2doc
```

CHAPTER

SIX

CREDITS

6.1 Development Lead

- Andy Salnikov <ged4py@py-dev.com>

6.2 Contributors

- Dariusz Kania <voyager212@wp.pl> - Polish translation
- Jan Mejstrik <jan.mejstrik@seznam.cz> - Czech translation

CHAPTER

SEVEN

HISTORY

7.1 0.5.1 (2021-05-01)

- Add Python3.9 to supported version list

7.2 0.5.0 (2020-10-09)

- Python3 goodies, use enum classes for enums

7.3 0.4.2 (2020-10-04)

- Use numpydoc style for docstrings, add extension to Sphinx
- Drop Python2 compatibility code

7.4 0.4.1 (2020-09-28)

- Use github actions instead of Travis CI

7.5 0.4.0 (2020-09-28)

- Drop Python2 support
- Python3 supported versions are 3.6 - 3.8

7.6 0.3.1 (2020-08-30)

- No changes in ged2doc code, new version is to build Windows app with latest ged4py version (v0.2.4) which fixes Genery.com dialect detection.

7.7 0.3.0 (2020-08-02)

- New module `dumbemf` implements a subset of EMF drawing primitives to support EMF representation of ancestor trees.
- Ancestor tree in ODT output is now in EMF format by default, there is an option to use SVG format as in previous versions

7.8 0.2.1 (2020-07-18)

- Fix DateValue error and Python2 compatibility issues (#28).

7.9 0.2.0 (2020-07-05)

- Update for improved DATE support in ged4py

7.10 0.1.16 (2018-09-08)

- Add protection from exceptions in Pillow

7.11 0.1.15 (2018-06-02)

- Add Czech translation (kudos to @Mejstro)
- Add –locale option
- Fix for SVG file (ancestor trees) headers

7.12 0.1.14 (2018-05-17)

- Improve logging and error reporting

7.13 0.1.13 (2018-04-23)

- Fix for event date comparison (#15)

7.14 0.1.12 (2018-04-21)

- Fixed (hopefully) crash when dealing with GIF images.
- Added Polish translation (kudos to @voyager212)
- Improved image search algorithm for on-disk and zipped locations

7.15 0.1.11 (2018-04-07)

- Output events without dates, and add option to disable it

7.16 0.1.10 (2018-04-02)

- Changed Russian translation for “Maiden name”

7.17 0.1.9 (2018-02-03)

- Improve format of generic events and attributes, use TYPE as event type
- Add “cause” to formatted event if “CAUS” tag is present

7.18 0.1.8 (2018-01-31)

- Require ged4py 0.1.4 or later
- This improves name parsing for ALTREE dialect

7.19 0.1.7 (2018-01-28)

- Fixed bug causing exception for small images: UnboundLocalError: local variable ‘imgsize’ referenced before assignment

7.20 0.1.6 (2018-01-21)

- Update docs, add russian translation for usage/installation

7.21 0.1.5 (2018-01-16)

- Try to open images using full path

7.22 0.1.4 (2018-01-16)

- Python3 fixes, bytes handling

7.23 0.1.3 (2018-01-14)

- add –version option to print ged2doc/ged4py versions

7.24 0.1.2 (2018-01-13)

- Small fix for packaging

7.25 0.1.1 (2018-01-07)

- Add support for ODT output.
- Add options for names formatting
- Automatic determination of output format from file extension

7.26 0.1.0 (2017-10-20)

- First release on PyPI.
- Only supporting HTML output for now.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X

Y