xls2xlsx Documentation

Release 0.2.0

Joe Cool

Jan 06, 2023

Contents:

CHAPTER 1

xls2xlsx

Convert xls file to xlsx

- Free software: MIT license
- Documentation: https://xls2xlsx.readthedocs.io.

1.1 Features

- Convert `.xls` files to `.xlsx` using xlrd and openpyxl.
- Convert `.htm` and `.mht` files containing tables or excel contents to `.xlsx` using beautifulsoup4 and openpyxl.

We attempt to support anything that the underlying packages used will support. For example, the following are supported for both input types:

- Multiple worksheets
- Text, Numbers, Dates/Times, Unicode
- Fonts, text color, bold, italic, underline, double underline, strikeout
- Solid and Pattern Fills with color
- Borders: Solid, Hair, Thin, Thick, Double, Dashed, Dotted; with color
- Alignment: Horizontal, Vertical, Rotated, Indent, Shrink To Fit
- Number Formats, including unicode currency symbols
- Hidden Rows and Columns
- Merged Cells

- Hyperlinks (only 1 per cell)
- Comments

These features are additionally supported by the `.xls` input format:

- Freeze panes

These features are additional supported by the `.htm` and `.mht` input formats:

- Images

Not supported by either format:

- Conditional Formatting (the current stylings are preserved)
- Formulas (the calculated values are preserved)
- Charts (the image of the chart is handled by `.htm` and `.mht` input formats)
- Drawings (the image of the drawing is handled by `.htm` and `.mht` input formats)
- Pivot tables (the current data is preserved)
- Text boxes (converted to an image by `.htm` and `.mht` input formats)
- Shapes and Clip Art (converted to an image by `.htm` and `.mht` input formats)
- Autofilter (the current filtered out rows are preserved)
- Rich text in cells (openpyxl doesn't support this: only styles applied to the entire cell are preserved)
- Named Ranges
- Macros (VBA)

1.2 Installation

To install xls2xlsx, run this command in your terminal:

```
$ pip install xls2xlsx
```

This is the preferred method to install xls2xlsx, as it will always install the most recent stable release.

1.3 Usage

To use xls2xlsx from the command line:

```
$ xls2xlsx [-v] file.xls ...
```

This will create `file.xlsx` in the current folder. `file.xls` can be any `.xls`, `.htm`, or `.mht` file and can also be a URL. The `-v` flag will print the input and output filename.

To use xls2xlsx in a project:

```
from xls2xlsx import XLS2XLSX
x2x = XLS2XLSX("spreadsheet.xls")
x2x.to_xlsx("spreadsheet.xlsx")
```

Alternatively:

```
from xls2xlsx import XLS2XLSX
x2x = XLS2XLSX("spreadsheet.xls")
wb = x2x.to_xlsx()
```

The xls2xlsx.to_xlsx method returns the filename given. If no filename is provided, the method returns the openpyxl workbook.

The input file can be in any of the following formats:

- Excel 97-2003 workbook (`.xls`)
- Web page (`.htm`, `.html`), optionally including a _Files folder
- Single file web page (`.mht`, `.mhtml`)

The input specified can also be any of the following:

- A filename / pathname
- A url
- A file-like object (opened in Binary mode for `.xls` and either Binary or Text mode otherwise)
- The contents of a `.xls` file as a `bytes` object
- The contents of a `.htm` or `.mht` file as a `str` object

Note: The file format is determined by examining the file contents, *not* by looking at the file extension.

1.4 Dependencies

Python >= 3.7 is required.

These packages are also required: `xlrd`, `openpyxl`, `requests`, `beautifulsoup4`, `Pillow`, `python-dateutil`, `cssutils`, `webcolors`, `currency-symbols`, `chardet`, `fonttools`, `PyYAML`.

1.5 Implementation Notes

The `.htm` and `.mht` input format conversion uses ImageFont from Pillow to measure the size (width and height) of cell contents. The first time you use it, it will look for font files in standard places on your system and create a Font Name to filename mapping. If the proper font files are not found on your system corresponding to the fonts used in the input file, then as a backup, an estimation algorithm is used.

If passed a `.mht` file (or url), the temporary folder name specified in the file will be used to unpack the contents for processing, then this folder will be removed when done.

1.6 Credits

1.6.1 Development Lead

- Joe Cool <snoopyjc@gmail.com>

1.6.2 Contributors

- Stitch-Zhang: https://github.com/Stitch-Zhang
- tigsinthetrees: https://github.com/tigsinthetrees

1.7 Acknowledgements

A portion of the code is based on the work of John Ricco (johnricco226@gmail.com), Apr 4, 2017: https://johnricco.github.io/2017/04/04/python-html/

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

CHAPTER 2

Installation

2.1 Stable release

To install xls2xlsx, run this command in your terminal:

```
$ pip install xls2xlsx
```

This is the preferred method to install xls2xlsx, 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 xls2xlsx can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/snoopyjc/xls2xlsx
```

Or download the tarball:

```
$ curl -OJL https://github.com/snoopyjc/xls2xlsx/tarball/master
```

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

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

CHAPTER 3

Usage

To use xls2xlsx from the command line:

```
$ xls2xlsx [-v] file.xls ...
```

This will create `file.xlsx` in the current folder. `file.xls` can be any `.xls`, `.htm`, or `.mht` file and can also be a URL. The `-v` flag will print the input and output filename.

To use xls2xlsx in a project:

```
from xls2xlsx import XLS2XLSX
x2x = XLS2XLSX("spreadsheet.xls")
x2x.to_xlsx("spreadsheet.xlsx")
```

Alternatively:

```
from xls2xlsx import XLS2XLSX
x2x = XLS2XLSX("spreadsheet.xls")
wb = x2x.to_xlsx()
```

The xls2xlsx.to_xlsx method returns the filename given. If no filename is provided, the method returns the openpyxl workbook.

The input file can be in any of the following formats:

- Excel 97-2003 workbook (`.xls`)
- Web page (`.htm`, `.html`), optionally including a _Files folder
- Single file web page (`.mht`, `.mhtml`)

The input specified can also be any of the following:

- A filename / pathname
- A url
- A file-like object (opened in Binary mode for `.xls` and either Binary or Text mode otherwise)

- The contents of a `.xls` file as a `bytes` object
- The contents of a `.htm` or `.mht` file as a `str` object

Note: The file format is determined by examining the file contents, *not* by looking at the file extension.

CHAPTER 4

xls2xlsx

4.1 xls2xlsx package

4.1.1 Submodules

4.1.2 xls2xlsx.cli module

Console script for xls2xlsx.

xls2xlsx.cli.**main**()
Console script for xls2xlsx.

4.1.3 xls2xlsx.htmlxls2xlsx module

class xls2xlsx.htmlxls2xlsx.**CSSStyle**
Bases: object

DEFAULT_CELL_WIDTH_PX = 64

DEFAULT_POINT_SIZE = 10

MAX_CELL_HEIGHT_PT = 409

MAX_CELL_WIDTH_UNITS = 255

MIN_CELL_HEIGHT_PT = 11.25

MIN_CELL_WIDTH_PX = 15

RETRIES = 6

SPREADSHEET_HEIGHT_PX = 800

SPREADSHEET_WIDTH_PX = 1900

add_style_sheet(*style*)

apply_style(*tags*)
Apply the appropriate style to the given element(s). The tags is a list of tuples of (tag_name, tag_attrs), given in the proper nested order.

Handled: * .class #id element element.class element,element (handled above) element element (direct descendants only) element>element [attribute] [attribute=value]

Not handled: .class1.class2 .class1 .class2 element+element element~element

static css_escape_unicode(*s*)

static fixup_excel_width(*wid*)

static format_style(*style*)
Convert a parsed style dict back to a style string

static get_pt(*item*, *default=0.0*, *default_units='px'*, *spreadsheet_pt=1425.0*)
For an item like 0.5pt or 2px, get the float value in units of pt

static get_px(*item*, *default=0.0*, *spreadsheet_px=1900*)
For an item like 0.5pt or 2px, get the float value in units of px

static get_units(*item*, *default=None*)
For an item like 0.5pt, return the units = 'pt'

static get_value(*item*, *default=0.0*)
For an item like 0.5pt, get the float value = 0.5

static html_escape_unicode(*s*)

static join(*base*, *fn*)
Join a base URL with a filename or a base local path with a filename

map_font(*font*)
Map the given font to something that Excel probably has

parse_style(*styl*)
Parse a style string (e.g. from style="...") to a dict

static px_to_units(*px*)
Convert pixels to excel column width units (determined empirically)

static read(*f*, *mode=''*, *quiet=False*, *retries=6*)
Read from either a URL or a filename or file-like object. If mode is 'b', then read in binary. If f is a file-like object and you want it read in unicode with the proper encoding, then open it using 'rb' mode and don't pass a mode to this method. If quiet is True, then return None rather than raising an exception on errors

style_to_xlsx(*style*, *font=None*, *fill=None*, *border=None*, *alignment=None*, *number_format=None*)
Convert this style to the appropriate attributes for openpyxl. returns a tuple with (font, fill, border, alignment)

static to_xlsx_color(*color*)

static units_to_px(*units*)
Convert Excel column width units to pixels

update_style(*style*, *new_style*, *new_tag=None*, *parent=False*)
Like normal dict.update() but handle relative size fonts (% and em), inherit, and initial. The new_tag specifies the tag name associated with the new_style and is used to process values of "initial". If parent is True (default), then the "style" element is the parent style of "new_style", which determines if properties are inherited or just merged.

class xls2xlsx.htmlxls2xlsx.**FontUtils**
Bases: object

LINE_HEIGHT_FACTOR = 1.25

get_font

get_font_path(*name*, *bold=False*, *italic=False*)
Given a font name (which may not be in the proper case), return the path to the font file or None if not found

get_font_size(*font*, *s*)
Get the width and height of text 's' in the given font. 's' is a single line of text (without newlines)

get_real_font_name(*name*, *bold=False*, *italic=False*)
Given a font name which may not be in the proper case, return the real font name, or None if not found

lines_needed(*img_width*, *s*, *font*)
How many lines are needed to render this text 's' using img_width pixels?

static pt_to_px(*pt*)

static px_to_pt(*px*)

static str_to_filename(*s*, *ext*)
Convert this string to a valid filename (used for debugging only)

class xls2xlsx.htmlxls2xlsx.**HTMLXLS2XLSX**(*f*, *dirname='.'*)
Bases: object

Convert an xls file with html contents into and xlsx file

to_xlsx(*filename=None*, *workbook=None*, *worksheet=None*, *sheet_name=None*)
Convert to xlsx using openpyxl. If filename is not None, then the result is written to that file, and the filename is returned, else the workbook is returned. If workbook is passed, then the worksheet is written to the given workbook

4.1.4 xls2xlsx.utils module

xls2xlsx.utils.**is_date_format**(*fmt*)

xls2xlsx.utils.**perform_number_format**(*value*, *number_format*)
This is a half-baked attempt at formatting the given value using the given Excel number_format. This is used by the tests to match values. Handled is many of the formats for numbers (int/float), datetime, date, time, and timedelta.

4.1.5 xls2xlsx.xls2xlsx module

class xls2xlsx.xls2xlsx.**XLS2XLSX**(*f*, *dirname='.'*, *ignore_workbook_corruption=False*)
Bases: object

Convert an xls file into an xlsx file. Everything is supported except for the things not supported by xlrd, which include:

1. Conditional Formatting
2. Formulas
3. Charts and Images
4. Pivot Tables

If this xls file is in html format, then we call HTMLXLS2XLSX to convert it.

static read(*f*, *retries=6*)
Read from either a URL or a filename or file-like object.

to_xlsx(*filename=None*)
Convert to xlsx using openpyxl. If filename is not None, then the result is written to that file, and the filename is returned, else the workbook is returned.

xls_color_to_xlsx(*color_ndx*)

xls_date_to_xlsx(*value*)

xls_height_to_xlsx(*height*)

xls_style_to_xlsx(*xf_ndx*)
Convert an xls xf_ndx into a 6-tuple of styles for xlsx

xls_width_to_xlsx(*width*)

4.1.6 Module contents

Top-level package for xls2xlsx.

CHAPTER 5

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/snoopyjc/xls2xlsx/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, including the input spreadsheet.

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

xls2xlsx could always use more documentation, whether as part of the official xls2xlsx 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/snoopyjc/xls2xlsx/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 *xls2xlsx* for local development.

1. Fork the *xls2xlsx* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/xls2xlsx.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 xls2xlsx
$ cd xls2xlsx/
$ 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 the tests:

```
$ pytest
```

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, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/snoopyjc/xls2xlsx/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

6.1 Development Lead

- Joe Cool <snoopyjc@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.2.0 (2023-01-05)

- Modernize for more recent pythons and more recent packages. Drop support for Python 3.6. Fix issues #11, #14, #16. Add feature #12.

7.2 0.1.5 (2020-11-03)

- Fix issues #1, #3, #5

7.3 0.1.4 (2020-11-02)

- Fix issue #4

7.4 0.1.3 (2020-10-15)

- Fix issue #2 - cli not working

7.5 0.1.0 (2020-09-13)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

x

Index