pylightxl

Release 2019

Jan 12, 2021

Contents

Fig. 1: pylightxl pypi | github

A light weight Microsoft Excel File reader. Although there are several excellent read/write options out there (python-excel.org or excelpython.org) pylightxl focused on the following key features:

- **Zero non-standard library dependencies** No compatibility/version control issues.
- **Light-weight single source code file that supports both Python3 and Python2.7.18.** Single source files that can easily be copied directly into a project for true zero-dependency. Great for those that have installation/download restrictions. In addition the library's size and zero dependency makes this library pyinstaller compilation small and easy!
- **100% test-driven development for highest reliability/maintainability with 100% coverage on all supported versions**
- **API aimed to be user friendly and intuitive. Structure: database > worksheet > indexing** example: `db.ws('Sheet1').index(row=1,col=2)` or `db.ws('Sheet1').address(address='B1')`

CHAPTER 1

High-Level Feature Summary

- Read excel files (`.xlsx`, `.xlsm`), all sheets or selective few for speed/memory management
- Index cell data by row/col number or address
- Calling an entire row/col of data returns an easy to use list output: `db.ws('Sheet1').row(1)` or `db.ws('Sheet1').rows`
- Worksheet data size is consistent for each row/col. Any data that is empty will return a '' (default empty cell can be updated)
- Write to existing or now spreadsheets

CHAPTER 2

Limitations

Although every effort was made to support a variety of users, the following limitations should be read carefully:

- Does not support `.xls` files (Microsoft Excel 2003 and older files)
- Writer does not support anything other than cell data (no graphs, images, macros, formatting)
- Does not support worksheet cell data more than 536,870,912 cells (32-bit list limitation)

2.1 Installation

There are several ways to use `pylightxl`. The easiest download is through `pip` however we understand that certain workplaces may not allow externally downloaded content, therefore we made it easy to copy/paste the source code needed to get going as well. Please see License terms of agreement: *License*

2.1.1 pip install

Download via Python Package Installer:

```
pip install pylightxl
```

2.1.2 Git Clone

Download via github clone:

```
git clone https://github.com/PydPiper/pylightxl.git
```

2.1.3 Download Source Files

Download via github: https://github.com/PydPiper/pylightxl/archive/master.zip

2.1.4 Copy Source Files

Create a copy of the entire library that the user can copy directly into a project, a virtual environment, or into the python/lib/site-packages folder for general use.

1.) Create a folder `pylightxl`

2.) Create the following files within the `pylightxl` folder:

```
pylightxl
    1- __init__.py
    2- pylightxl.py
```

3.) Populate the files with their respective source code contents:

3.1) File1: __init__.py

3.2) File2: pylightxl.py

2.2 Quick Start Guide

Get up and running in less than 5 minutes with pylightxl!

2.2.1 Read/Write CSV File

Read a csv file with contents:

```
import pylightxl as xl

# set the delimiter of the CSV to be the value of your choosing
# set the default worksheet to write the read in CSV data to
db = xl.readcsv(fn='input.csv', delimiter='/', ws='sh2')
# make modifications to it then,
# now write it back out as a csv; or as an excel file, see xl.writexl()
xl.writecsv(db=db, fn='new.csv', ws=('sh2'), delimiter=',')
```

2.2.2 Read Excel File

```
import pylightxl as xl

# readxl returns a pylightxl database that holds all worksheets and its data
db = xl.readxl(fn='folder1/folder2/excelfile.xlsx')

# read only selective sheetnames
db = xl.readxl(fn='folder1/folder2/excelfile.xlsx', ws=('Sheet1','Sheet3'))

# return all sheetnames
db.ws_names
>>> ['Sheet1', 'Sheet3']
```

2.2.3 Access Worksheet and Cell Data

The following example assumes `excelfile.xlsx` contains a worksheet named `Sheet1` and it has the following cell content:

	A	B	C
1	10	20	
2		30	40

Via Cell Address

```
db.ws(ws='Sheet1').address(address='A1')
>>> 10
db.ws(ws='Sheet1').address(address='A1', formula=True)
>>> ''
# note index a empty cell will return an empty string
db.ws(ws='Sheet1').address(address='A100')
>>> ''
# however default empty value can be overwritten for each worksheet
db.ws(ws='Sheet1').set_emptycell(val=0)
db.ws(ws='Sheet1').address(address='A100')
>>> 0
```

Via Cell Index

```
db.ws(ws='Sheet1').index(row=1, col=2)
>>> 20
db.ws(ws='Sheet1').index(row=1, col=2, formula=True)
>>> '=A1+10'
# note index a empty cell will return an empty string
db.ws(ws='Sheet1').index(row=100, col=1)
>>> ''
# however default empty value can be overwritten for each worksheet
db.ws(ws='Sheet1').set_emptycell(val=0)
db.ws(ws='Sheet1').index(row=100, col=1)
>>> 0
```

Via Cell Range

```
db.ws(ws='Sheet1').range(address='A1', formula=False)
>>> 10
db.ws(ws='Sheet1').range(address='A1:C2', formula=False)
>>> [[10, 20, ''], ['', 30, 40]]
db.ws(ws='Sheet1').range(address='A1:B1', formula=True)
>>> [['=10', '=A1+10']]
```

Get entire row or column

```
db.ws(ws='Sheet1').row(row=1)
>>> [10,20,'']

db.ws(ws='Sheet1').col(col=1)
>>> [10,'']
```

Iterate through rows/cols

```
for row in db.ws(ws='Sheet1').rows:
    print(row)

>>> [10,20,'']
>>> ['',30,40]

for col in db.ws(ws='Sheet1').cols:
    print(col)

>>> [10,'']
>>> [20,30]
>>> ['',40]
```

Update Cell Value

```
db.ws(ws='Sheet1').address(address='A1')
>>> 10
db.ws(ws='Sheet1').update_address(address='A1', val=100)
db.ws(ws='Sheet1').address(address='A1')
>>> 100

db.ws(ws='Sheet1').update_index(row=1, col=1, val=10)
db.ws(ws='Sheet1').index(row=1, col=1)
>>> 10
```

Update Cell Formula

Same as update cell value except the entry must begin with a equal sign "=”

Note: updating a cell formula will clear the previously read in cell value. Formulas will not calculate their cell value until the excel file is opened.

```
db.ws(ws='Sheet1').update_address(address='A1', val='=B1+100')
db.ws(ws='Sheet1').update_index(row=1, col=1, val='=B1+100')
```

Get Named Ranges

```
# define a named range
db.add_nr(name='Table1', ws='Sheet1', address='A1:B2')
# see all existing named ranges
```

(continues on next page)

(continued from previous page)

```
db.nr_names
>>> {'Table1': 'Sheet1!A1:B2'}
# get the contents of a named ranges
db.nr(name='Table1', formula=False)
>>> [[10, 20], ['', 30]]
# remove a named range
db.remove_nr(name='Table1')
```

Get row/col based on key-value

Note: key is type sensitive

```
# lets say we would like to return the column that has a cell value = 20 in row=1
db.ws(ws='Sheet1').keycol(key=20, keyindex=1)
>>> [20,30]

# we can also specify a custom keyindex (not just row=1), note that we now are␣
↪matched based on row=2
db.ws(ws='Sheet1').keycol(key=30, keyindex=2)
>>> [20,30]

# similarly done for keyrow with keyindex=1 (look fora match in col=1)
db.ws(ws='Sheet1').keyrow(key='', keyindex=1)
>>> ['',30,40]
```

2.2.4 Read Semi-Structured Data

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2												
3												
4	Data Table 2											
5	KEYCOLSKEYROWS	c1	c2	c3								
6	r1	1	2	3								
7	r2	4		6								
8	r3	7	8	9								
9												
10												
11												
12			Data Table 2									
13						KEYCOLS	cc1	cc2	cc3		cc4	
14												
15												
16			KEYROWS									
17			rr1				10	20	30		x	
18			rr2				40	50	60		x	
19			rr3				70	80	90		x	
20			rr4				100	110	120		x	
21												
22			rr5				x	x	x		x	
23												

- note that `ssd` function takes any key-word argument as your KEYROWS/KEYCOLS flag

- multiple tables are read the same way as you would read a book. Top left-to-right, then down

```
import pylightxl
db = pylightxl.readxl(fn='Book1.xlsx')

# request a semi-structured data (ssd) output
ssd = db.ws(ws='Sheet1').ssd(keycols="KEYCOLS", keyrows="KEYROWS")

ssd[0]
>>> {'keyrows': ['r1', 'r2', 'r3'], 'keycols': ['c1', 'c2', 'c3'], 'data': [[1, 2, 3],
↪ [4, '', 6], [7, 8, 9]]}
ssd[1]
>>> {'keyrows': ['rr1', 'rr2', 'rr3', 'rr4'], 'keycols': ['cc1', 'cc2', 'cc3'], 'data
↪': [[10, 20, 30], [40, 50, 60], [70, 80, 90], [100, 110, 120]]}
```

2.2.5 Write out a pylightxl.Database as an excel file

Pylightxl support excel writing without having excel installed on the machine. However it is not without its limitations. The writer only supports cell data writing (ie.: does not support graphs, formatting, images, macros, etc) simply just strings/numbers/equations in cells.

Note that equations typed by the user will not calculate for its value until the excel sheet is opened in excel.

```
import pylightxl as xl

# read in an existing worksheet and change values of its cells (same worksheet as␣
↪above)
db = xl.readxl(fn='excelfile.xlsx')
# overwrite existing number value
db.ws(ws='Sheet1').index(row=1, col=1)
>>> 10
db.ws(ws='Sheet1').update_index(row=1, col=1, val=100)
db.ws(ws='Sheet1').index(row=1, col=1)
>>> 100
# write text
db.ws(ws='Sheet1').update_index(row=1, col=2, val='twenty')
# write equations
db.ws(ws='Sheet1').update_address(address='A3', val='=A1')

xl.writexl(db=db, fn='updated.xlsx')
```

2.2.6 Write a new excel file from python data

For new python data that did not come from an existing excel speadsheet.

```
import pylightxl as xl

# take this list for example as our input data that we want to put in column A
mydata = [10,20,30,40]

# create a black db
db = xl.Database()

# add a blank worksheet to the db
db.add_ws(ws="Sheet1")
```

(continues on next page)

(continued from previous page)

```
# loop to add our data to the worksheet
for row_id, data in enumerate(mydata, start=1)
    db.ws(ws="Sheet1").update_index(row=row_id, col=1, val=data)

# write out the db
xl.writexl(db=db, fn="output.xlsx")
```

2.3 Source Code Documentation

2.3.1 readxl

pylightxl.pylightxl.**readxl**(*fn*, *ws=None*)

Reads an xlsx or xlsm file and returns a pylightxl database

Parameters

- **fn** (*str*) – Excel file name
- **or list ws** (*str*) – sheetnames to read into the database, if not specified - all sheets are read entry support single ws name (ex: ws='sh1') or multi (ex: ws=['sh1', 'sh2'])

Returns pylightxl.Database class

2.3.2 writexl

pylightxl.pylightxl.**writexl**(*db*, *fn*)

Writes an excel file from pylightxl.Database

Parameters

- **db** (pylightxl.Database) – database contains sheetnames, and their data
- **fn** (*str/pathlib*) – file output path

Returns None

2.3.3 database

Database Class

class pylightxl.pylightxl.**Database**

add_nr(*name*, *ws*, *address*)

Add a NamedRange to the database. There can not be duplicate name or addresses. A named range that overlaps either the name or address will overwrite the database's existing NamedRange

Parameters

- **name** (*str*) – NamedRange name
- **ws** (*str*) – worksheet name
- **address** (*str*) – range of address (single cell ex: "A1", range ex: "A1:B4")

Returns None

add_ws(*ws*, *data=None*)
Logs worksheet name and its data in the database

Parameters

- **ws** (*str*) – worksheet name
- **data** – dictionary of worksheet cell values (ex: {'A1': {'v':10,'f':'','s':''}, 'A2': {'v':20,'f':'','s':''}})

Returns None

nr(*name*, *formula=False*)
Returns the contents of a name range in a nest list form [row][col]

Parameters

- **name** (*str*) – NamedRange name
- **formula** (*bool*) – flag to return the formula of this cell

Return list nest list form [row][col]

nr_names
Returns the dictionary of named ranges ex: {unique_name: unique_address, ... }

Return dict {unique_name: unique_address, ... }

remove_nr(*name*)
Removes a Named Range from the database

Parameters **name** (*str*) – NamedRange name

Returns None

remove_ws(*ws*)
Removes a worksheet and its data from the database

Parameters **ws** (*str*) – worksheet name

Returns None

rename_ws(*old*, *new*)
Renames an existing worksheet. Caution, renaming to an existing new worksheet name will overwrite

Parameters

- **old** (*str*) – old name
- **new** (*str*) – new name

Returns None

set_emptycell(*val*)
Custom definition for how pylightxl returns an empty cell

Parameters **val** – (default='') empty cell value

Returns None

ws(*ws*)
Indexes worksheets within the database

Parameters **ws** (*str*) – worksheet name

Returns pylightxl.Database.Worksheet class object

ws_names
Returns a list of database stored worksheet names

Returns list of worksheet names

Worksheet Class

class pylightxl.pylightxl.**Worksheet**(*data=None*)

address(*address*, *formula=False*)
Takes an excel address and returns the worksheet stored value

Parameters

- **address** (*str*) – Excel address (ex: "A1")
- **formula** (*bool*) – flag to return the formula of this cell

Returns cell value

col(*col*, *formula=False*)
Takes a col index input and returns a list of cell data

Parameters

- **col** (*int*) – col index (start at 1 that corresponds to column "A")
- **formula** (*bool*) – flag to return the formula of this cell

Returns list of cell data

cols
Returns a list of cols that can be iterated through

Returns list of cols-lists (ex: [[11,21],[12,22],[13,23]] for 2 rows with 3 columns of data

index(*row*, *col*, *formula=False*)
Takes an excel row and col starting at index 1 and returns the worksheet stored value

Parameters

- **row** (*int*) – row index (starting at 1)
- **col** (*int*) – col index (start at 1 that corresponds to column "A")
- **formula** (*bool*) – flag to return the formula of this cell

Returns cell value

keycol(*key*, *keyindex=1*)
Takes a column key value (value of any cell within keyindex row) and returns the entire column, no match returns an empty list

Parameters

- **key** (*str/int/float*) – any cell value within keyindex row (type sensitive)
- **keyindex** (*int*) – option keyrow override. Must be >0 and smaller than worksheet size

Return list list of the entire matched key column data (only first match is returned)

keyrow(*key*, *keyindex=1*)
Takes a row key value (value of any cell within keyindex col) and returns the entire row, no match returns an empty list

Parameters

- **key** (*str/int/float*) – any cell value within keyindex col (type sensitive)
- **keyrow** (*int*) – option keyrow override. Must be >0 and smaller than worksheet size

Return list list of the entire matched key row data (only first match is returned)

range(*address*, *formula=False*)
Takes an range (ex: "A1:A2") and returns a nested list [row][col]

Parameters

- **address** (*str*) – cell range (ex: "A1:A2", or "A1")
- **formula** (*bool*) – returns the values if false, or formulas if true of cells

Return list nested list [row][col] regardless if range is a single cell or a range

row(*row*, *formula=False*)
Takes a row index input and returns a list of cell data

Parameters

- **row** (*int*) – row index (starting at 1)
- **formula** (*bool*) – flag to return the formula of this cell

Returns list of cell data

rows
Returns a list of rows that can be iterated through

Returns list of rows-lists (ex: [[11,12,13],[21,22,23]] for 2 rows with 3 columns of data

set_emptycell(*val*)
Custom definition for how pylightxl returns an empty cell

Parameters **val** – (default='') empty cell value

Returns None

size
Returns the size of the worksheet (row/col)

Returns list of [maxrow, maxcol]

ssd(*keyrows='KEYROWS'*, *keycols='KEYCOLS'*)
Runs through the worksheet and looks for "KEYROWS" and "KEYCOLS" flags in each cell to identify the start of a semi-structured data. A data table is read until an empty header is found by row or column. The search supports multiple tables.

Parameters

- **keyrows** (*str*) – (default='KEYROWS') a flag to indicate the start of keyrow's cells below are read until an empty cell is reached
- **keycols** (*str*) – (default='KEYCOLS') a flag to indicate the start of keycol's cells to the right are read until an empty cell is reached

Return list list of data dict in the form of [{'keyrows': [], 'keycols': [], 'data': [[], ...]}, {...},]

update_address(*address*, *val*)
Update worksheet data via address

Parameters

- **address** (*str*) – excel address (ex: "A1")

- **val** (*int/float/str*) – cell value; equations are strings and must begin with "="

Returns None

update_index(*row*, *col*, *val*)
Update worksheet data via index

Parameters

- **row** (*int*) – row index
- **col** (*int*) – column index
- **val** (*int/float/str*) – cell value; equations are strings and must begin with "="

Returns None

Support Functions

2.4 Example Solutions

2.4.1 Reading Semi Structured data

- Question posted on stackoverflow
- Problem: read groups of 2D data from a single sheet that can begin at any row/col and has any number of rows/columns per data group, see figure below.

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2												
3												
4	Data Table 2											
5	KEYCOLSKEYROWS	c1	c2	c3								
6	r1	1	2	3								
7	r2	4		6								
8	r3	7	8	9								
9												
10												
11												
12			Data Table 2									
13						KEYCOLS	cc1	cc2	cc3		cc4	
14												
15												
16			KEYROWS									
17			rr1				10	20	30		x	
18			rr2				40	50	60		x	
19			rr3				70	80	90		x	
20			rr4				100	110	120		x	
21												
22			rr5				x	x	x		x	
23												

- Solution: note that `ssd` function takes any key-word argument as your KEYROWS/KEYCOLS flag and multiple tables are read the same way as you would read a book. Top left-to-right, then down.

```
import pylightxl
db = pylightxl.readxl('Book1.xlsx')

# request a semi-structured data (ssd) output
ssd = db.ws('Sheet1').ssd(keycols="KEYCOLS", keyrows="KEYROWS")

ssd[0]
>>> {'keyrows': ['r1', 'r2', 'r3'], 'keycols': ['c1', 'c2', 'c3'], 'data': [[1, 2, 3],
↪ [4, '', 6], [7, 8, 9]]}
ssd[1]
>>> {'keyrows': ['rr1', 'rr2', 'rr3', 'rr4'], 'keycols': ['cc1', 'cc2', 'cc3'], 'data
↪': [[10, 20, 30], [40, 50, 60], [70, 80, 90], [100, 110, 120]]}
```

2.5 Revision Log

2.5.1 pypi version 1.52 (in-work)

- speed improvements
- context manager for database (tear down for memory management)

2.5.2 pypi version 1.51

- license update within setup.py

2.5.3 pypi version 1.50

- hot-fix: added python2 support for encoding with cgi instead of html

2.5.4 pypi version 1.49

- bug-fix: updated encoding for string cells that contained xml-like data (ex: cell A1 "<cell content>")

2.5.5 pypi version 1.48

- add feature to `writecsv` to be able to handle `pathlib` object and `io.StreamIO` object
- refactored readxl to remove regex, now readxl is all cElementTree
- refactored readxl/writexl to able to handle excel files written by openpyxl that is generated differently than how excel write files.

2.5.6 pypi version 1.47

- added new function: `db.nr('table1')` returns the contents of named range "table1"
- added new function: `db.ws('Sheet1').range('A1:C3')` that returns the contents of a range it also has the ability to return the formulas of the range

- updated `db.ws('Sheet1').row()` and `db.ws('Sheet1').col()` to take in a new argument `formual` that returns the formulas of a row or col
- bugfix: write to existing without named ranges was throwing a "repair" error. Fixed typo on xml for it and added unit tests to capture it
- added new function: `xl.readcsv(fn, delimiter, ws)` to read csv files and create a pylightxl db out of it (type converted)
- added new function: `xl.writecsv(db, fn, ws, delimiter)` to write out a pylightxl worksheet as a csv

2.5.7 pypi version 1.46

- bug fix: added ability to input an empty string into the cell update functions (previously entering val='') threw and error

2.5.8 pypi version 1.45

- added support for cell values that have multiple formats within a single cell. previous versions did not support this functionality since it is logged differently in sharedString.xml
- added support for updating formulas and viewing them:
  - view formula: `db.ws('Sheet1').address('A1', formula=True)`
  - edit formula: `db.ws('Sheet1').update_address('A1', val='=A1+10')`
- updated the following function arguments to drive commonality:
  - was: `readxl(fn, sheetnames)` new: `readxl(fn, ws)`
  - was: `writexl(db, path)` new: `writexl(db, fn)`
  - was: `db.ws(sheetname)` new: `db.ws(ws)`
  - was: `db.add_ws(sheetname, data)` new: `db.add_ws(ws, data)`
- added new feature to be able to read-in NamedRanges, store it in the Database, update it, remove it, and write it. NamedRanges were integrated with existing function to handle semi-structured-data
  - `db.add_nr(name'range1', ws='sheet1', address='A1:C2')`
  - `db.remove_nr(name='range1')`
  - `db.nr_names`
- add feature to remove worksheet: `db.remove_ws(ws='Sheet1')`
- add feature to rename worksheet: `db.rename_ws(old='sh1', new='sh2')`
- added a cleanup function upon writing to delete _pylightxl_ temp folder in case an error left them
- added feature to write to file that is open by excel by appending a "**new_**" tag to the file name and a warning message that file is opened by excel so a file was saved as "**new_**" + filename

2.5.9 pypi version 1.44

- bug fix: accounted for num2letter roll-over issue
- new feature: added a pylightxl native function for handling semi-structured data

2.5.10 pypi version 1.43

- bug fix: accounted for reading error'ed out cell "#N/A"
- bug fix: accounted for bool TRUE/FALSE cell values not registering on readxl
- bug fix: accounted for edge case that was prematurely splitting cell tags <c r /> by formula closing bracket <f />
- bug fix: accounted for cell address roll-over

2.5.11 pypi version 1.42

- added support for pathlib file reading
- bug fix: previous version did not handle merged cells properly
- bug fix: database updates did not update maxcol maxrow if new data addition was larger than the initial dataset
- bug fix: writexl that use linefeeds did not read in properly into readxl (fixed regex)
- bug fix: writexl filepath issues

2.5.12 pypi version 1.41

- new-feature: write new excel file from pylightxl.Database
- new-feature: write to existing excel file from pylightxl.Database
- new-feature: db.update_index(row, col, val) for user defined cell values
- new-feature: db.update_address(address, val) for user defined cell values
- bug fix for reading user defined sheets
- bug fix for mis-alignment of reading user defined sheets and xml files

2.5.13 pypi version 1.3

- new-feature: add the ability to call rows/cols via key-value ex: `db.ws('Sheet1').keycol('my column header')` will return the entire column that has 'my column header' in row 1
- fixed-bug: fixed leading/trailing spaced cell text values that are marked `<t xml:space="preserve">` in the sharedString.xml

2.5.14 pypi version 1.2

- fixed-bug: fixed Sheet number to custom Sheet name matching for 10+ sheets that were previously only sorting alphabetical which resulted with sorting: Sheet1, Sheet10, Sheet11, Sheet2... and so on.

2.5.15 pypi version 1.1

- initial release

2.6 License

CHAPTER 3

Support Content Creator

If you enjoyed this library, please consider supporting its creators! Help Today

Index