Openpyxl 2.4.2

A Python library to read/write Excel 2010 xlsx/xlsm files

Contenu

Pypi.python.org

https://pypi.python.org/pypi/openpyxl

Package Documentation

openpyxl

openpyxl is a Python library to read/write Excel 2010 xlsx/xlsm/xltx/xltm files.

It was born from lack of existing library to read/write natively from Python the Office Open XML format.

All kudos to the PHPExcel team as openpyxl was initially based on PHPExcel

Mailing List

Official user list can be found on http://groups.google.com/group/openpyxl-users

Sample code:

```
from openpyxl import Workbook
wb = Workbook()

# grab the active worksheet
ws = wb.active

# Data can be assigned directly to cells
ws['A1'] = 42

# Rows can also be appended
ws.append([1, 2, 3])

# Python types will automatically be converted
import datetime
ws['A2'] = datetime.datetime.now()

# Save the file
wb.save("sample.xlsx")
```

Official documentation

The documentation is at: https://openpyxl.readthedocs.io

- installation methods
- code examples

readthedocs.io

https://openpyxl.readthedocs.io/en/default/

openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files

Author:	Eric Gazoni, Charlie Clark
Source code:	http://bitbucket.org/openpyxl/openpyxl/src
Issues:	http://bitbucket.org/openpyxl/openpyxl/issues
Generated:	February 18, 2017
License:	MIT/Expat
Version:	2.4.2

Introduction

Openpyxl is a Python library for reading and writing Excel 2010 xlsx/xlsm/xltx/xltm files.

It was born from lack of existing library to read/write natively from Python the Office Open XML format.

All kudos to the PHPExcel team as openpyxl was initially based on PHPExcel.

Support

This is an open source project, maintained by volunteers in their spare time. This may well mean that particular features or functions that you would like are missing. But things don't have to stay that way. You can contribute the project *Development* yourself or contract a developer for particular features.

Professional support for openpyxl is available from Clark Consulting & Research and Adimian. Donations to the project to support further development and maintenance are welcome.

Bug reports and feature requests should be submitted using the issue tracker. Please provide a full traceback of any error you see and if possible a sample file. If for reasons of confidentiality you are unable to make a file publicly available then contact of one the developers.

Sample code:

```
from openpyxl import Workbook
wb = Workbook()

# grab the active worksheet
ws = wb.active
```

```
# Data can be assigned directly to cells
ws['A1'] = 42

# Rows can also be appended
ws.append([1, 2, 3])

# Python types will automatically be converted
import datetime
ws['A2'] = datetime.datetime.now()

# Save the file
wb.save("sample.xlsx")
```

Installation

Install openpyxl using pip. It is advisable to do this in a Python virtualenv without system packages:

```
$ pip install openpyxl
```

Note

There is support for the popular lxml library which will be used if it is installed. This is particular useful when creating large files.

Warning

To be able to include images (jpeg, png, bmp,...) into an openpyxl file, you will also need the “pillow” library that can be installed with:

```
$ pip install pillow
```

or browse https://pypi.python.org/pypi/Pillow/, pick the latest version and head to the bottom of the page for Windows binaries.

Usage examples

Tutorial

Cookbook

Pandas and NumPy

Charts

Tutorial : Manipulating a workbook in memory

https://openpyxl.readthedocs.io/en/default/tutorial.html

Create a workbook

There is no need to create a file on the filesystem to get started with openpyxl. Just import the Workbook class and start using it

```
>>> from openpyxl import Workbook
>>> wb = Workbook()
```

A workbook is always created with at least one worksheet. You can get it by using the `openpyxl.workbook.Workbook.active()` property

```
>>> ws = wb.active
```

Note

This function uses the *_active_sheet_index* property, set to 0 by default. Unless you modify its value, you will always get the first worksheet by using this method.

You can also create new worksheets by using the `openpyxl.workbook.Workbook.create_sheet()` method

```
>>> ws1 = wb.create_sheet("Mysheet") # insert at the end (default)
# or
>>> ws2 = wb.create_sheet("Mysheet", 0) # insert at first position
```

Sheets are given a name automatically when they are created. They are numbered in sequence (Sheet, Sheet1, Sheet2, ...). You can change this name at any time with the *title* property:

```
ws.title = "New Title"
```

The background color of the tab holding this title is white by default. You can change this providing an RRGGBB color code to the sheet_properties.tabColor property:

```
ws.sheet_properties.tabColor = "1072BA"
```

Once you gave a worksheet a name, you can get it as a key of the workbook:

```
>>> ws3 = wb["New Title"]
```

You can review the names of all worksheets of the workbook with the `openpyxl.workbook.Workbook.sheetnames()` property

```
>>> print(wb.sheetnames)
['Sheet2', 'New Title', 'Sheet1']
```

You can loop through worksheets

```
>>> for sheet in wb:
...     print(sheet.title)
```

You can create copies of worksheets within a single workbook:

`openpyxl.workbook.Workbook.copy_worksheet()` method:

```
>>> source = wb.active
>>> target = wb.copy_worksheet(source)
```

Note

Only cells and styles can be copied. You cannot copy worksheets between workbooks.

You can copy worksheets in a workbook with the

Playing with data

Accessing one cell

Now we know how to access a worksheet, we can start modifying cells content.

Cells can be accessed directly as keys of the worksheet

```
>>> c = ws['A4']
```

This will return the cell at A4 or create one if it does not exist yet. Values can be directly assigned

```
>>> ws['A4'] = 4
```

There is also the `openpyxl.worksheet.Worksheet.cell()` method.

This provides access to cells using row and column notation:

```
>>> d = ws.cell(row=4, column=2, value=10)
```

Note

When a worksheet is created in memory, it contains no *cells*. They are created when first accessed.

Warning

Because of this feature, scrolling through cells instead of accessing them directly will create them all in memory, even if you don’t assign them a value.

Something like

```
>>> for i in range(1,101):
...         for j in range(1,101):
...             ws.cell(row=i, column=j)
```

will create 100x100 cells in memory, for nothing.

Accessing many cells

Ranges of cells can be accessed using slicing

```
>>> cell_range = ws['A1':'C2']
```

Ranges of rows or columns can be obtained similarly:

```
>>> colC = ws['C']
>>> col_range = ws['C:D']
>>> row10 = ws[10]
>>> row_range = ws[5:10]
```

You can also use the `openpyxl.worksheet.Worksheet.iter_rows()` method:

```
>>> for row in ws.iter_rows(min_row=1, max_col=3, max_row=2):
...     for cell in row:
...         print(cell)
<Cell Sheet1.A1>
<Cell Sheet1.B1>
<Cell Sheet1.C1>
<Cell Sheet1.A2>
<Cell Sheet1.B2>
<Cell Sheet1.C2>
```

Likewise the `openpyxl.worksheet.Worksheet.iter_cols()` method will return columns:

```
>>> for col in ws.iter_cols(min_row=1, max_col=3, max_row=2):
...     for cell in col:
...         print(cell)
<Cell Sheet1.A1>
<Cell Sheet1.A2>
<Cell Sheet1.B1>
<Cell Sheet1.B2>
<Cell Sheet1.C1>
<Cell Sheet1.C2>
```

If you need to iterate through all the rows or columns of a file, you can instead use the `openpyxl.worksheet.Worksheet.rows()` property:

```
>>> ws = wb.active
>>> ws['C9'] = 'hello world'
>>> tuple(ws.rows)
((<Cell Sheet.A1>, <Cell Sheet.B1>, <Cell Sheet.C1>),
(<Cell Sheet.A2>, <Cell Sheet.B2>, <Cell Sheet.C2>),
(<Cell Sheet.A3>, <Cell Sheet.B3>, <Cell Sheet.C3>),
(<Cell Sheet.A4>, <Cell Sheet.B4>, <Cell Sheet.C4>),
(<Cell Sheet.A5>, <Cell Sheet.B5>, <Cell Sheet.C5>),
(<Cell Sheet.A6>, <Cell Sheet.B6>, <Cell Sheet.C6>),
(<Cell Sheet.A7>, <Cell Sheet.B7>, <Cell Sheet.C7>),
(<Cell Sheet.A8>, <Cell Sheet.B8>, <Cell Sheet.C8>),
(<Cell Sheet.A9>, <Cell Sheet.B9>, <Cell Sheet.C9>))
```

or the `openpyxl.worksheet.Worksheet.columns()` property:

```
>>> tuple(ws.columns)
((<Cell Sheet.A1>,
<Cell Sheet.A2>,
<Cell Sheet.A3>,
<Cell Sheet.A4>,
<Cell Sheet.A5>,
<Cell Sheet.A6>,
...
<Cell Sheet.B7>,
<Cell Sheet.B8>,
<Cell Sheet.B9>),
(<Cell Sheet.C1>,
<Cell Sheet.C2>,
<Cell Sheet.C3>,
<Cell Sheet.C4>,
<Cell Sheet.C5>,
<Cell Sheet.C6>,
<Cell Sheet.C7>,
<Cell Sheet.C8>,
<Cell Sheet.C9>))
```

Data storage

Once we have a `openpyxl.cell.Cell`, we can assign it a value:

```
>>> c.value = 'hello, world'
>>> print(c.value)
'hello, world'

>>> d.value = 3.14
>>> print(d.value)
3.14
```

You can also enable type and format inference:

```
>>> wb = Workbook(guess_types=True)
>>> c.value = '12%'
>>> print(c.value)
0.12

>>> import datetime
>>> d.value = datetime.datetime.now()
>>> print d.value
datetime.datetime(2010, 9, 10, 22, 25, 18)

>>> c.value = '31.50'
>>> print(c.value)
31.5
```

Saving to a file

The simplest and safest way to save a workbook is by using the `openpyxl.workbook.Workbook.save()` method of the `openpyxl.workbook.Workbook` object:

```
>>> wb = Workbook()
>>> wb.save('balances.xlsx')
```

Warning

This operation will overwrite existing files without warning.

Note

Extension is not forced to be xlsx or xlsm, although you might have some trouble opening it directly with another application if you don’t use an official extension.

As OOXML files are basically ZIP files, you can also end the filename with .zip and open it with your favourite ZIP archive manager.

You can specify the attribute *template=True*, to save a workbook as a template:

```
>>> wb = load_workbook('document.xlsx')
>>> wb.template = True
>>> wb.save('document_template.xltx')
```

or set this attribute to *False* (default), to save as a document:

```
>>> wb = load_workbook('document_template.xltx')
>>> wb.template = False
>>> wb.save('document.xlsx', as_template=False)
```

Warning

You should monitor the data attributes and document extensions for saving documents in the document templates and vice versa, otherwise the result table engine can not open the document.

Note

The following will fail:

```
>>> wb = load_workbook('document.xlsx')
>>> # Need to save with the extension *.xlsx
>>> wb.save('new_document.xlsm')
>>> # MS Excel can't open the document
>>>
>>> # or
>>>
>>> # Need specify attribute keep_vba=True
>>> wb = load_workbook('document.xlsm')
>>> wb.save('new_document.xlsm')
>>> # MS Excel will not open the document
>>>
>>> # or
>>>
>>> wb = load_workbook('document.xltm', keep_vba=True)
>>> # If we need a template document, then we must specify extension as
*.xltm.
>>> wb.save('new_document.xlsm')
>>> # MS Excel will not open the document
```

Loading from a file

The same way as writing, you can import `openpyxl.load_workbook()` to open an existing workbook:

```
>>> from openpyxl import load_workbook
>>> wb2 = load_workbook('test.xlsx')
>>> print wb2.get_sheet_names()
['Sheet2', 'New Title', 'Sheet1']
```

This ends the tutorial for now, you can proceed to the *Simple usage* section

Tutorial : Simple usage

Write a workbook

```
>>> from openpyxl import Workbook
>>> from openpyxl.compat import range
>>> from openpyxl.utils import get_column_letter
>>>
>>> wb = Workbook()
>>>
>>> dest_filename = 'empty_book.xlsx'
>>>
>>> ws1 = wb.active
>>> ws1.title = "range names"
>>>
>>> for row in range(1, 40):
...     ws1.append(range(600))
>>>
>>> ws2 = wb.create_sheet(title="Pi")
>>>
>>> ws2['F5'] = 3.14
>>>
>>> ws3 = wb.create_sheet(title="Data")
>>> for row in range(10, 20):
...     for col in range(27, 54):
...         _ = ws3.cell(column=col, row=row,
value="{0}".format(get_column_letter(col)))
>>> print(ws3['AA10'].value)
AA
>>> wb.save(filename = dest_filename)
```

Read an existing workbook

```
>>> from openpyxl import load_workbook
>>> wb = load_workbook(filename = 'empty_book.xlsx')
>>> sheet_ranges = wb['range names']
>>> print(sheet_ranges['D18'].value)
3
```

Note

There are several flags that can be used in load_workbook.

- *guess_types* will enable or disable (default) type inference when reading cells.
- *data_only* controls whether cells with formulae have either the formula (default) or the value stored the last time Excel read the sheet.
- *keep_vba* controls whether any Visual Basic elements are preserved or not (default). If they are preserved they are still not editable.

Warning

openpyxl does currently not read all possible items in an Excel file so images and charts will be lost from existing files if they are opened and saved with the same name.

Using number formats

```
>>> import datetime
>>> from openpyxl import Workbook
>>> wb = Workbook()
>>> ws = wb.active
>>> # set date using a Python datetime
>>> ws['A1'] = datetime.datetime(2010, 7, 21)
>>>
>>> ws['A1'].number_format
'yyyy-mm-dd h:mm:ss'
>>> # You can enable type inference on a case-by-case basis
>>> wb.guess_types = True
>>> # set percentage using a string followed by the percent sign
>>> ws['B1'] = '3.14%'
>>> wb.guess_types = False
>>> ws['B1'].value
0.031400000000000004
>>>
>>> ws['B1'].number_format
'0%'
```

Using formulae

```
>>> from openpyxl import Workbook
>>> wb = Workbook()
>>> ws = wb.active
>>> # add a simple formula
>>> ws["A1"] = "=SUM(1, 1)"
>>> wb.save("formula.xlsx")
```

Warning

NB you must use the English name for a function and function arguments *must* be separated by commas and not other punctuation such as semi-colons.

openpyxl never evaluates formula but it is possible to check the name of a formula:

```
>>> from openpyxl.utils import FORMULAE
>>> "HEX2DEC" in FORMULAE
True
```

If you’re trying to use a formula that isn’t known this could be because you’re using a formula that was not included in the initial specification. Such formulae must be prefixed with *xlfn.* to work.

Merge / Unmerge cells

When you merge cells all cells but the top-left one are **removed** from the worksheet. See Styling Merged Cells for information on formatting merged cells.

```
>>> from openpyxl.workbook import Workbook
>>>
>>> wb = Workbook()
>>> ws = wb.active
>>>
>>> ws.merge_cells('A1:B1')
>>> ws.unmerge_cells('A1:B1')
>>>
>>> # or
>>> ws.merge_cells(start_row=2,start_column=1,end_row=2,end_column=4)
>>> ws.unmerge_cells(start_row=2,start_column=1,end_row=2,end_column=4)
```

Inserting an image

```
>>> from openpyxl import Workbook
>>> from openpyxl.drawing.image import Image
>>>
>>> wb = Workbook()
>>> ws = wb.active
>>> ws['A1'] = 'You should see three logos below'
>>> # create an image
>>> img = Image('logo.png')
>>> # add to worksheet and anchor next to cells
>>> ws.add_image(img, 'A1')
>>> wb.save('logo.xlsx')
```

Fold columns (outline)

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> ws = wb.create_sheet()
>>> ws.column_dimensions.group('A','D', hidden=True)
>>> wb.save('group.xlsx')
```

Tutorial : Read/write large files

Read-only mode

Sometimes, you will need to open or write extremely large XLSX files, and the common routines in openpyxl won’t be able to handle that load. Fortunately, there are two modes that enable you to read and write unlimited amounts of data with (near) constant memory consumption.

Introducing `openpyxl.worksheet.read_only.ReadOnlyWorksheet`:

```
from openpyxl import load_workbook
wb = load_workbook(filename='large_file.xlsx', read_only=True)
ws = wb['big_data']

for row in ws.rows:
    for cell in row:
        print(cell.value)
```

Warning

- `openpyxl.worksheet.read_only.ReadOnlyWorksheet` is read-only

Cells returned are not regular `openpyxl.cell.cell.Cell` but `openpyxl.cell.read_only.ReadOnlyCell`.

Worksheet dimensions

Read-only mode relies applications and libraries that created the file providing correct information about the worksheets, specifically the used part of it, known as the dimensions. Some applications set this incorrectly. You can check the apparent dimensions of a worksheet using *ws.calculate_dimensions()*. If this returns a range that you know is incorrect, say *A1:A1* then simply resetting the max_row and max_column attributes should allow you to work with the file:

```
ws.max_row = ws.max_column = None
```

Write-only mode

Here again, the regular `openpyxl.worksheet.worksheet.Worksheet` has been replaced by a faster alternative, the `openpyxl.writer.write_only.WriteOnlyWorksheet`. When you want to dump large amounts of data, you might find write-only helpful.

```
>>> from openpyxl import Workbook
>>> wb = Workbook(write_only=True)
>>> ws = wb.create_sheet()
>>>
>>> # now we'll fill it with 100 rows x 200 columns
>>>
>>> for irow in range(100):
...     ws.append(['%d' % i for i in range(200)])
>>> # save the file
>>> wb.save('new_big_file.xlsx')
```

If you want to have cells with styles or comments then use a `openpyxl.writer.write_only.WriteOnlyCell()`

```
>>> from openpyxl import Workbook
>>> wb = Workbook(write_only = True)
>>> ws = wb.create_sheet()
>>> from openpyxl.writer.write_only import WriteOnlyCell
>>> from openpyxl.comments import Comment
>>> from openpyxl.styles import Font
>>> cell = WriteOnlyCell(ws, value="hello world")
>>> cell.font = Font(name='Courier', size=36)
>>> cell.comment = Comment(text="A comment", author="Author's Name")
>>> ws.append([cell, 3.14, None])
>>> wb.save('write_only_file.xlsx')
```

This will create a write-only workbook with a single sheet, and append a row of 3 cells: one text cell with a custom font and a comment, a floating-point number, and an empty cell (which will be discarded anyway).

Warning

- Unlike a normal workbook, a newly-created write-only workbook does not contain any worksheets; a worksheet must be specifically created with the `create_sheet()` method.
- In a write-only workbook, rows can only be added with `append()`. It is not possible to write (or read) cells at arbitrary locations with `cell()` or `iter_rows()`.
- It is able to export unlimited amount of data (even more than Excel can handle actually), while keeping memory usage under 10Mb.
- A write-only workbook can only be saved once. After that, every attempt to save the workbook or append() to an existing worksheet will raise an `openpyxl.utils.exceptions.WorkbookAlreadySaved` exception.