openpyxl Documentation

Release 1.6.2

Eric Gazoni

August 07, 2013

CONTENTS

Author Eric Gazoni

Source code http://bitbucket.org/ericgazoni/openpyxl/src

Issues http://bitbucket.org/ericgazoni/openpyxl/issues

Generated July 20, 2013

Version 1.6.2

CHAPTER

ONE

INTRODUCTION

OpenPyxl is a Python library to read/write Excel 2007 xlsx/xlsm files.

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

All kudos to the PHPExcel team as openpyxl is a Python port of PHPExcel http://www.phpexcel.net/

CHAPTER

TWO

USER LIST

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

CHAPTER

THREE

HOW TO CONTRIBUTE CODE

Any help will be greatly appreciated, just follow those steps:

1. Please start a new fork (https://bitbucket.org/ericgazoni/openpyxl/fork) for each independant feature, don't try to fix all problems at the same time, it's easier for those who will review and merge your changes ;-)
2. Hack hack hack
3. Don't forget to add unit tests for your changes ! (YES, even if it's a one-liner, or there is a high probability your work will not be taken into consideration). There are plenty of examples in the /test directory if you lack know-how or inspiration.
4. If you added a whole new feature, or just improved something, you can be proud of it, so add yourself to the AUTHORS file :-)
5. Let people know about the shiny thing you just implemented, update the docs !
6. When it's done, just issue a pull request (click on the large "pull request" button on *your* repository) and wait for your code to be reviewed, and, if you followed all theses steps, merged into the main repository.

This is an open-source project, maintained by volunteers on their spare time, so while we try to work on this project as often as possible, sometimes life gets in the way. Please be patient.

CHAPTER

FOUR

OTHER WAYS TO HELP

There are several ways to contribute, even if you can't code (or can't code well):

- triaging bugs on the bug tracker: closing bugs that have already been closed, are not relevant, cannot be reproduced, ...
- updating documentation in virtually every area: many large features have been added (mainly about charts and images at the moment) but without any documentation, it's pretty hard to do anything with it
- proposing compatibility fixes for different versions of Python: we try to support 2.5 to 3.3, so if it does not work on your environment, let us know :-)

CHAPTER

FIVE

INSTALLATION

The best method to install openpyxl is using a PyPi client such as easy_install (setuptools) or pip:

```
$ pip install openpyxl
```

or

```
$ easy_install install openpyxl
```

Note: To install from sources (there is nothing to build, openpyxl is 100% pure Python), you can download an archive from bitbucket (look in the "tags" tab).

After extracting the archive, you can do:

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

Warning: To be able to include images (jpeg,png,bmp,...) into an openpyxl file, you will also need the 'PIL' 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.

CHAPTER

SIX

GETTING THE SOURCE

Source code is hosted on bitbucket.org. You can get it using a Mercurial client and the following URLs:

- $ hg clone https://bitbucket.org/ericgazoni/openpyxl -r 1.6.2

or to get the latest development version:

- $ hg clone https://bitbucket.org/ericgazoni/openpyxl

CHAPTER

SEVEN

USAGE EXAMPLES

7.1 Tutorial

7.1.1 Manipulating a workbook in memory

Create a workbook

There is no need to create a file on the filesystem to get started with openpyxl. Just import the Worbook 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.get_active_sheet()` method

```
>>> ws = wb.get_active_sheet()
```

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() # insert at the end (default)
# or
>>> ws2 = wb.create_sheet(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"
```

Once you gave a worksheet a name, you can get it using the `openpyxl.workbook.Workbook.get_sheet_by_name()` method

```
>>> ws3 = wb.get_sheet_by_name("New Title")
>>> ws is ws3
True
```

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

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

Playing with data

Accessing one cell

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

To access a cell, use the `openpyxl.worksheet.Worksheet.cell()` method:

```
>>> c = ws.cell('A4')
```

You can also access a cell using row and column notation:

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

Note: When a worksheet is created in memory, it contains no *cells*. They are created when first accessed. This way we don't create objects that would never be accessed, thus reducing the memory footprint.

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 xrange(0,100):
...         for j in xrange(0,100):
...             ws.cell(row = i, column = j)
```

will create 100x100 cells in memory, for nothing.
However, there is a way to clean all those unwanted cells, we'll see that later.

Accessing many cells

If you want to access a *range*, wich is a two-dimension array of cells, you can use the `openpyxl.worksheet.Worksheet.range()` method:

```
>>> ws.range('A1:C2')
((<Cell Sheet1.A1>, <Cell Sheet1.B1>, <Cell Sheet1.C1>),
 (<Cell Sheet1.A2>, <Cell Sheet1.B2>, <Cell Sheet1.C2>))

>>> for row in ws.range('A1:C2'):
...         for cell in row:
...             print cell
<Cell Sheet1.A1>
<Cell Sheet1.B1>
<Cell Sheet1.C1>
<Cell Sheet1.A2>
<Cell Sheet1.B2>
<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.get_active_sheet()
>>> ws.cell('C9').value = 'hello world'
>>> 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:

```
>>> 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
```

There is also a neat format detection feature that converts data on the fly:

```
>>> 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
```

7.1.2 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.

7.1.3 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

7.2 Cookbook

7.2.1 Simple usage

Write a workbook

```
from openpyxl import Workbook

from openpyxl.cell import get_column_letter

wb = Workbook()

dest_filename = r'empty_book.xlsx'

ws = wb.worksheets[0]

ws.title = "range names"

for col_idx in xrange(1, 40):
```

```
    col = get_column_letter(col_idx)
    for row in xrange(1, 600):
        ws.cell('%s%s'%(col, row)).value = '%s%s' % (col, row)

ws = wb.create_sheet()

ws.title = 'Pi'

ws.cell('F5').value = 3.14

wb.save(filename = dest_filename)
```

Read an existing workbook

```
from openpyxl import load_workbook

wb = load_workbook(filename = r'empty_book.xlsx')

sheet_ranges = wb.get_sheet_by_name(name = 'range names')

print sheet_ranges.cell('D18').value # D18
```

Using number formats

```
import datetime
from openpyxl import Workbook

wb = Workbook()
ws = wb.worksheets[0]

# set date using a Python datetime
ws.cell('A1').value = datetime.datetime(2010, 7, 21)

print ws.cell('A1').style.number_format.format_code # returns 'yyyy-mm-dd'

# set percentage using a string followed by the percent sign
ws.cell('B1').value = '3.14%'

print ws.cell('B1').value # returns 0.031400000000000004

print ws.cell('B1').style.number_format.format_code # returns '0%'
```

Inserting an image

```
from openpyxl import Workbook
from openpyxl.drawing import Image

wb = Workbook()
ws = wb.get_active_sheet()
ws.cell('A1').value = 'You should see a logo below'

# create an image instance
img = Image('logo.png')
```

```
# place it if required
img.drawing.left = 200
img.drawing.top = 100

# you could also 'anchor' the image to a specific cell
# img.anchor(ws.cell('B12'))

# add to worksheet
ws.add_image(img)
wb.save('logo.xlsx')
```

Validating cells

```
from openpyxl import Workbook
from openpyxl.datavalidation import DataValidation, ValidationType

# Create the workbook and worksheet we'll be working with
wb = Workbook()
ws = wb.get_active_sheet()

# Create a data-validation object with list validation
dv = DataValidation(ValidationType.LIST, formula1='"Dog,Cat,Bat"', allow_blank=True)

# Optionally set a custom error message
dv.set_error_message('Your entry is not in the list', 'Invalid Entry')

# Optionally set a custom prompt message
dv.set_prompt_message('Please select from the list', 'List Selection')

# Add the data-validation object to the worksheet
ws.add_data_validation(dv)

# Create some cells, and add them to the data-validation object
c1 = ws.cell("A1")
c1.value = "Dog"
dv.add_cell(c1)
c2 = ws.cell("A2")
c2.value = "An invalid value"
dv.add_cell(c2)

# Or, apply the validation to a range of cells
dv.ranges.append('B1:B1048576')

# Write the sheet out.  If you now open the sheet in Excel, you'll find that
# the cells have data-validation applied.
wb.save("test.xlsx")
```

Other validation examples

Any whole number:

```
dv = DataValidation(ValidationType.WHOLE)
```

Any whole number above 100:

```
dv = DataValidation(ValidationType.WHOLE,
                    ValidationOperator.GREATER_THAN,
                    100)
```

Any decimal number:

```
dv = DataValidation(ValidationType.DECIMAL)
```

Any decimal number between 0 and 1:

```
dv = DataValidation(ValidationType.DECIMAL,
                    ValidationOperator.BETWEEN,
                    0, 1)
```

Any date:

```
dv = DataValidation(ValidationType.DATE)
```

or time:

```
dv = DataValidation(ValidationType.TIME)
```

Any string at most 15 characters:

```
dv = DataValidation(ValidationType.TEXT_LENGTH,
                    ValidationOperator.LESS_THAN_OR_EQUAL,
                    15)
```

Custom rule:

```
dv = DataValidation(ValidationType.CUSTOM,
                    None,
                    "=SOMEFORMULA")
```

Note: See http://www.contextures.com/xlDataVal07.html for custom rules

---

7.3 Read/write large files

7.3.1 Optimized reader

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. Hopefully, there are two modes that enable you to read and write unlimited amounts of data with (near) constant memory consumption.

Introducing `openpyxl.reader.iter_worksheet.IterableWorksheet`:

```
from openpyxl import load_workbook
wb = load_workbook(filename = 'large_file.xlsx', use_iterators = True)
ws = wb.get_sheet_by_name(name = 'big_data') # ws is now an IterableWorksheet

for row in ws.iter_rows(): # it brings a new method: iter_rows()

    for cell in row:

        print cell.internal_value
```

Warning:
- As you can see, we are using cell.internal_value instead of .value.
- `openpyxl.reader.iter_worksheet.IterableWorksheet` are read-only
- cell, range, rows, columns methods and properties are disabled

Cells returned by iter_rows() are not regular `openpyxl.cell.Cell` but `openpyxl.reader.iter_worksheet.RawCell`.

7.3.2 Optimized writer

Here again, the regular `openpyxl.worksheet.Worksheet` has been replaced by a faster alternative, the `openpyxl.writer.dump_worksheet.DumpWorksheet`. When you want to dump large amounts of data, you might find optimized writer helpful:

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

ws = wb.create_sheet()

# now we'll fill it with 10k rows x 200 columns
for irow in xrange(10000):
    ws.append(['%d' % i for i in xrange(200)])

wb.save('new_big_file.xlsx') # don't forget to save !
```

Warning:
- Those worksheet only have an append() method, it's not possible to access independent cells directly (through cell() or range()). They are write-only.
- It is able to export unlimited amount of data (even more than Excel can handle actually), while keeping memory usage under 10Mb.
- A workbook using the optimized writer can only be saved once. After that, every attempt to save the workbook or append() to an existing worksheet will raise an `openpyxl.shared.exc.WorkbookAlreadySaved` exception.

CHAPTER

EIGHT

API DOCUMENTATION

8.1 Module openpyxl.workbook – Workbook

class openpyxl.workbook.**Workbook**(*optimized_write=False*, *encoding='utf-8'*, *worksheet_class=<class 'openpyxl.worksheet.Worksheet'>*, *optimized_worksheet_class=<class 'openpyxl.writer.dump_worksheet.DumpWorksheet'>*, *guess_types=True*)

Workbook is the container for all other parts of the document.

add_named_range(*named_range*)

Add an existing named_range to the list of named_ranges.

add_sheet(*worksheet*, *index=None*)

Add an existing worksheet (at an optional index).

create_named_range(*name*, *worksheet*, *range*, *scope=None*)

Create a new named_range on a worksheet

create_sheet(*index=None*, *title=None*)

Create a worksheet (at an optional index).

Parameters **index** (*int*) – optional position at which the sheet will be inserted

get_active_sheet()

Returns the current active sheet.

get_index(*worksheet*)

Return the index of the worksheet.

get_named_range(*name*)

Return the range specified by name.

get_named_ranges()

Return all named ranges

get_sheet_by_name(*name*)

Returns a worksheet by its name.

Returns None if no worksheet has the name specified.

Parameters **name** (*string*) – the name of the worksheet to look for

get_sheet_names()

Returns the list of the names of worksheets in the workbook.

Names are returned in the worksheets order.

Return type list of strings

remove_named_range(*named_range*)
Remove a named_range from this workbook.

remove_sheet(*worksheet*)
Remove a worksheet from this workbook.

save(*filename*)
Save the current workbook under the given *filename*. Use this function instead of using an *ExcelWriter*.

> **Warning:** When creating your workbook using *optimized_write* set to True, you will only be able to call this function once. Subsequents attempts to modify or save the file will raise an `openpyxl.shared.exc.WorkbookAlreadySaved` exception.

8.2 Module `openpyxl.worksheet` – Worksheet

class `openpyxl.worksheet.`**Worksheet**(*parent_workbook*, *title='Sheet'*)
Represents a worksheet.

Do not create worksheets yourself, use `openpyxl.workbook.Workbook.create_sheet()` instead

add_chart(*chart*)
Add a chart to the sheet

add_data_validation(*data_validation*)
Add a data-validation object to the sheet. The data-validation object defines the type of data-validation to be applied and the cell or range of cells it should apply to.

add_image(*img*)
Add an image to the sheet

add_print_title(*n*, *rows_or_cols='rows'*)
Print Titles are rows or columns that are repeated on each printed sheet. This adds n rows or columns at the top or left of the sheet

append(*list_or_dict*)
Appends a group of values at the bottom of the current sheet.

- If it's a list: all values are added in order, starting from the first column
- If it's a dict: values are assigned to the columns indicated by the keys (numbers or letters)

Parameters **list_or_dict** (*list/tuple or dict*) – list or dict containing values to append

Usage:

- append(['This is A1', 'This is B1', 'This is C1'])
- **or** append({'A' : 'This is A1', 'C' : 'This is C1'})
- **or** append({0 : 'This is A1', 2 : 'This is C1'})

Raise TypeError when list_or_dict is neither a list/tuple nor a dict

auto_filter
get or set auto filtering on columns

calculate_dimension()
Return the minimum bounding range for all cells containing data.

cell(*coordinate=None*, *row=None*, *column=None*)
Returns a cell object based on the given coordinates.

Usage: cell(coodinate='A15') **or** cell(row=15, column=1)

If *coordinates* are not given, then row *and* column must be given.

Cells are kept in a dictionary which is empty at the worksheet creation. Calling *cell* creates the cell in memory when they are first accessed, to reduce memory usage.

Parameters

- **coordinate** (*string*) – coordinates of the cell (e.g. 'B12')
- **row** (*int*) – row index of the cell (e.g. 4)
- **column** (*int*) – column index of the cell (e.g. 3)

Raise InsufficientCoordinatesException when coordinate or (row and column) are not given

Return type `openpyxl.cell.Cell`

create_relationship(*rel_type*)
Add a relationship for this sheet.

freeze_panes
Get or set frozen panes

garbage_collect()
Delete cells that are not storing a value.

get_cell_collection()
Return an unordered list of the cells in this worksheet.

get_highest_column()
Get the largest value for column currently stored.

Return type int

get_highest_row()
Returns the maximum row index containing data

Return type int

get_style(*coordinate*)
Return the style object for the specified cell.

merge_cells(*range_string=None*, *start_row=None*, *start_column=None*, *end_row=None*, *end_column=None*)
Set merge on a cell range. Range is a cell range (e.g. A1:E1)

point_pos(*left=0*, *top=0*)
tells which cell is under the given coordinates (in pixels) counting from the top-left corner of the sheet. Can be used to locate images and charts on the worksheet

range(*range_string*, *row=0*, *column=0*)
Returns a 2D array of cells, with optional row and column offsets.

Parameters

- **range_string** (*string*) – cell range string or *named range* name
- **row** (*int*) – number of rows to offset

- **column** (*int*) – number of columns to offset

Return type tuples of tuples of `openpyxl.cell.Cell`

set_printer_settings (*paper_size*, *orientation*)
Set printer settings

title
Get or set the title of the worksheet. Limited to 31 characters, no special characters.

unmerge_cells (*range_string=None*, *start_row=None*, *start_column=None*, *end_row=None*, *end_column=None*)
Remove merge on a cell range. Range is a cell range (e.g. A1:E1)

8.3 Module openpyxl.reader.iter_worksheet – Optimized reader

class openpyxl.reader.iter_worksheet.**IterableWorksheet** (*parent_workbook*, *title*, *workbook_name*, *sheet_codename*, *xml_source*, *string_table*)

iter_rows (*range_string=''*, *row_offset=0*, *column_offset=0*)
Returns a squared range based on the *range_string* parameter, using generators.

Parameters

- **range_string** (*string*) – range of cells (e.g. 'A1:C4')
- **row** (*int*) – row index of the cell (e.g. 4)
- **column** (*int*) – column index of the cell (e.g. 3)

Return type generator

class openpyxl.reader.iter_worksheet.**RawCell**
Optimized version of the `openpyxl.cell.Cell`, using named tuples.

Useful attributes are:

•row

•column

•coordinate

•internal_value

You can also access if needed:

•data_type

•number_format

8.4 Module openpyxl.cell – Worksheet Cell

class openpyxl.cell.**Cell** (*worksheet*, *column*, *row*, *value=None*)
Describes cell associated properties.

Properties of interest include style, type, value, and address.

address
Return the coordinate string for this cell (e.g. 'B12')

Return type string

anchor
returns the expected position of a cell in pixels from the top-left of the sheet. For example, A1 anchor should be (0,0).

Return type tuple(int, int)

bind_value(*value*)
Given a value, infer type and display options.

check_error(*value*)
Tries to convert Error" else N/A

check_numeric(*value*)
Cast value to int or float if necessary

check_string(*value*)
Check string coding, length, and line break character

data_type
Return the data type represented by this cell

data_type_for_value(*value*)
Given a value, infer the correct data type

get_coordinate()
Return the coordinate string for this cell (e.g. 'B12')

Return type string

has_style
Check if the parent worksheet has a style for this cell

hyperlink
Get or set the hyperlink held in the cell. Automatically sets the *value* of the cell with link text, but you can modify it afterwards by setting the *value* property, and the hyperlink will remain.

Return type string

hyperlink_rel_id
Return the id pointed to by the hyperlink, or None

is_date()
Returns whether the value is *probably* a date or not

Return type bool

offset(*row=0*, *column=0*)
Returns a cell location relative to this cell.

Parameters

- **row** (*int*) – number of rows to offset
- **column** (*int*) – number of columns to offset

Return type `openpyxl.cell.Cell`

set_value_explicit(*value=None*, *data_type='s'*)
Coerce values according to their explicit type

style
Returns the `openpyxl.style.Style` object for this cell

value
Get or set the value held in the cell.

Return type depends on the value (string, float, int or `datetime.datetime`)

8.5 Module `openpyxl.reader.excel` – Filesystem reader

`openpyxl.reader.excel.`**load_workbook**(*filename*, *use_iterators=False*, *keep_vba=False*, *guess_types=True*)
Open the given filename and return the workbook

Parameters

- **filename** (string or a file-like object open in binary mode c.f., `zipfile.ZipFile`) – the path to open or a file-like object
- **use_iterators** (*bool*) – use lazy load for cells
- **keep_vba** (*bool*) – preseve vba content (this does NOT mean you can use it)
- **guess_types** (*bool*) – guess cell content type and do not read it from the file

Return type `openpyxl.workbook.Workbook`

Note: When using lazy load, all worksheets will be `openpyxl.reader.iter_worksheet.IterableWorksheet` and the returned workbook will be read-only.

8.6 Module `openpyxl.writer.dump_worksheet` – Optimized writer

class `openpyxl.writer.dump_worksheet.`**DumpWorksheet**(*parent_workbook*, *title*)

> **Warning:** You shouldn't initialize this yourself, use `openpyxl.workbook.Workbook` constructor instead, with *optimized_write* = *True*.

append(*row*)

Parameters **row** (*iterable*) – iterable containing values to append

8.7 Module `openpyxl.datavalidation`

class `openpyxl.datavalidation.`**DataValidation**(*validation_type*, *operator=None*, *formula1=None*, *formula2=None*, *allow_blank=False*, *attr_map=None*)

add_cell(*cell*)
Adds a openpyxl.cell to this validator

set_error_message(*error*, *error_title='Validation Error'*)
Creates a custom error message, displayed when a user changes a cell to an invalid value

set_prompt_message(*prompt*, *prompt_title='Validation Prompt'*)
Creates a custom prompt message

class openpyxl.datavalidation.**ValidationType**

class openpyxl.datavalidation.**ValidationOperator**

CHAPTER

NINE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

o