xlref Documentation

Release 1.1.2

Vincenzo Arcidiacono

May 19, 2021

TABLE OF CONTENTS

2021-05-19 10:35:00

https://github.com/vinci1it2000/xlref

https://pypi.org/project/xlref/

http://xlref.readthedocs.io/

https://github.com/vinci1it2000/xlref/wiki/

http://github.com/vinci1it2000/xlref/releases/

https://donorbox.org/xlref

data, excel, tables, parser, reference, ranges

- Vincenzo Arcidiacono <vinci1it2000@gmail.com>

CHAPTER

ONE

ABOUT XLREF

xlref is an useful library to capture by a simple reference (e.g., *A1(RD):...:RD*) a table with non-empty cells from Excel-sheets when its exact position is not known beforehand.

This code was inspired by the *xleash* module of the pandalone library. The reason of developing a similar tool was to have a smaller library to install and improve the performances of reading *.xlsx* files.

CHAPTER

TWO

INSTALLATION

To install it use (with root privileges):

```
$ pip install xlref
```

Or download the last git version and use (with root privileges):

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

CHAPTER

THREE

REFERENCE SYNTAX

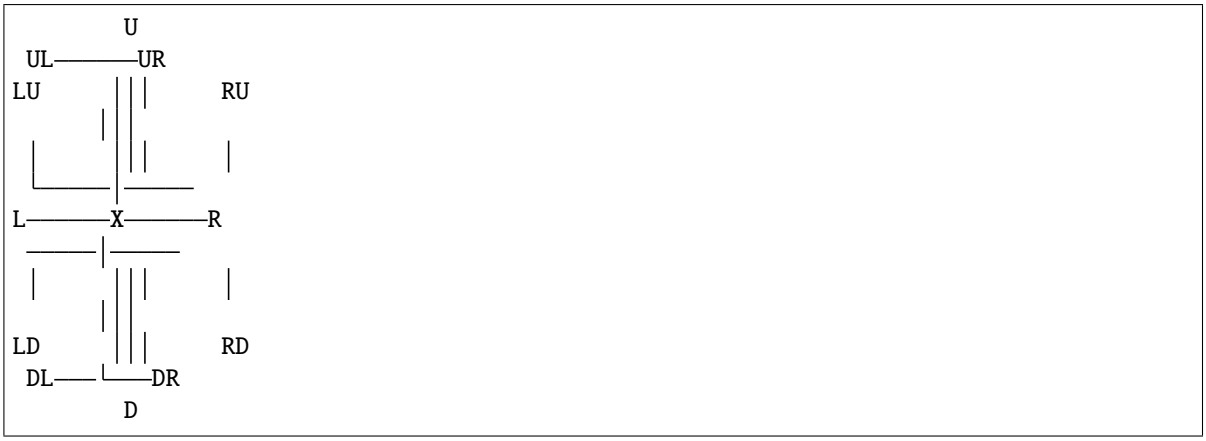
The *capturing* is preformed according to an excel like reference syntax and the non-empty cells of the targeted excel-sheet. The syntax is defined as follows:

[<excel>]#[<sheet>!]<st-cel>[(<moves>)][:<nd-cel>[(<moves>)]][:<expansion>][<filters>]

Note: The fields between square parenthesis are optionals.

Follows the description of the parameters:

- **excel**: excel file path relative to the parent reference file directory. If not defined, the parent reference excel is inherited.
- **sheet**: excel sheet name if not defined, the parent reference excel sheet name is inherited.
- **st-cel**: first cell coordinate of excel range. The cell coordinate (i.e., <*column*><*row*>) is defined by a column (letter) and row (number), like in excel. *xlref* allows two special characters ^ and _, that represents the left-most/topmost and rightmost/bottommost non-empty cell column/row.
- **moves**: the sequence of primitive directions (i.e., *L*:left, *U*: up, *R*: right, *D*: down) that *xlref* uses iteratively for finding the first non-empty cell. The allowed primitive direction combinations are *L*, *U*, *R*, *D*, *LD*, *LU*, *UL*, *UR*, *RU*, *RD*, *DL*, and *DR*. The following diagram shows the graphically the *moves* from the starting cell *X*:



- **nd-cel**: second cell coordinate of excel range. It has the same syntax of the *st-cel*, but it has and extra special character .. This represents the column or row of the *st-cel* after the application of the *moves*.
- **expansion**: the sequence of primitive directions to expand the captured range.
- **filters**: list of string and or dictionaries that defines the filters to apply iteratively on the captured range.

3.1 Reference Reading Steps

The library performs the following steps to read a reference:

1. Open the excel file or inherits the parent's one,
2. Open the sheet by its name or inherits the parent's one,
3. Set the first range cell,
4. Move the first cell according to the specified *moves* until it finds the first non-empty cell,
5. Set the second range cell or inherits the moved first range cell,
6. Move the second cell like in point *4*,
7. Expand the range according to the defined *expansions*,
8. Apply the iteratively the filters on the captured range.

CHAPTER

FOUR

TUTORIAL

A typical example is *capturing* a table with a "header" row and convert into a dictionary. The code below shows how to do it:

```
>>> import xlref as xl
>>> _ref = 'excel.xlsx#ref!A1(RD):RD[%s]'
>>> ref = xl.Ref(_ref % '"dict"')
>>> ref.range  # Captured range.
B2:C25
>>> values = ref.values; values  # Captured values.
{...}
>>> values['st-cell-move']
'#D5(RU):H1(DL)'
```

You can notice from the code above that all the values of the dictionary are references. To parse it recursively, there are two options:

1. add the "recursive" filter before the "dict":

   ```
   >>> values = xl.Ref(_ref % '"recursive", "dict"').values
   >>> values['st-cell-move'].tolist()
   [[1.0, 2.0, 3.0],
    [4.0, 5.0, 6.0],
    [7.0, 8.0, 9.0]]
   ```

2. apply a filter onto dictionary' values using the extra functionality of the "dict" filter:

   ```
   >>> values = xl.Ref(_ref % '{"fun": "dict", "value":"ref"}').values
   >>> values['st-cell-move'].tolist()
   [[1.0, 2.0, 3.0],
    [4.0, 5.0, 6.0],
    [7.0, 8.0, 9.0]]
   ```

You have also the possibility to define and use your custom filters as follows:

```
>>> import numpy as np
>>> xl.FILTERS['my-filter'] = lambda parent, x: np.sum(x)
>>> xl.Ref('#D5(RU):H1(DL)["my-filter"]', ref).values
45.0
```

An alternative way is to use directly the methods of the filtered results as follows:

```
>>> xl.Ref('#D5(RU):H1(DL)["sum"]', ref).values
45.0
```

CHAPTER

FIVE

CONTRIBUTING TO XLREF

If you want to contribute to **xlref** and make it better, your help is very welcome. The contribution should be sent by a *pull request*. Next sections will explain how to implement and submit a new functionality:

- clone the repository
- implement a new functionality
- open a pull request

5.1 Clone the repository

The first step to contribute to **xlref** is to clone the repository:

- Create a personal fork of the xlref repository on Github.
- Clone the fork on your local machine. Your remote repo on Github is called `origin`.
- Add the original repository as a remote called `upstream`, to maintain updated your fork.
- If you created your fork a while ago be sure to pull `upstream` changes into your local repository.
- Create a new branch to work on! Branch from `dev`.

5.2 How to implement a new functionality

Test cases are very important. This library uses a data-driven testing approach. To implement a new function I recommend the test-driven development cycle. Hence, when you think that the code is ready, add new test in `test` folder.

When all test cases are ok (`python setup.py test`), open a pull request.

Note: A pull request without new test case will not be taken into consideration.

5.3 How to open a pull request

Well done! Your contribution is ready to be submitted:

- Squash your commits into a single commit with git's interactive rebase. Create a new branch if necessary. Always write your commit messages in the present tense. Your commit message should describe what the commit, when applied, does to the code – not what you did to the code.
- Push your branch to your fork on Github (i.e., `git push origin dev`).
- From your fork open a *pull request* in the correct branch. Target the project's `dev` branch!
- Once the *pull request* is approved and merged you can pull the changes from `upstream` to your local repo and delete your extra branch(es).

CHAPTER

SIX

DONATE

If you want to support the **xlref** development please donate.

CHAPTER

SEVEN

API REFERENCE

The core of the library is composed from the following modules:

It contains a comprehensive list of all modules and classes within xlref.

Docstrings should provide sufficient understanding for any individual function.

Modules:

cli	Define the command line interface.
errors	Defines the xlref exceptions.
filters	It provides functions implementations to filter the parsed data.
parser	It provides xlparser reference parser class.
process	Defines the file processing chain model *dsp*.

7.1 cli

Define the command line interface.

7.1.1 xlref

xlref command line tool.

```
xlref [OPTIONS] COMMAND [ARGS]...
```

Options

--version
: Show the version and exit.

read

Read recursively the list of xlref data excel references.

OUTPUT_FILE: output file (format: .json).

INPUT_REFERENCE: xlref data excel reference.

```
xlref read [OPTIONS] OUTPUT_FILE [INPUT_REFERENCE]...
```

Options

-F, --input-file <input_file>
JSON xlref data excel references.

-v, --verbosity <LVL>
Either CRITICAL, ERROR, WARNING, INFO or DEBUG

Arguments

OUTPUT_FILE
Required argument

INPUT_REFERENCE
Optional argument(s)

7.2 errors

Defines the xlref exceptions.

Exceptions

InvalidReference
InvalidSyntax
NoFullCell
XlParserError

7.2.1 InvalidReference

exception InvalidReference(**args*)

7.2.2 InvalidSyntax

exception InvalidSyntax(**args*)

7.2.3 NoFullCell

exception NoFullCell(**args*)

7.2.4 XlParserError

exception XlParserError(**args*)

7.3 filters

It provides functions implementations to filter the parsed data.

Functions

fdict	Convert the input array into a dictionary.
full	Remove the empty value from each row of the input array.
recursive	Parse recursively all values in the array.
ref	If the input is a valid reference, returns the captured values otherwise the input.

7.3.1 fdict

fdict(*parent*, *x*, *key=None*, *value=None*)

Convert the input array into a dictionary.

Parameters

- **parent** (xlref.parser.Ref) – Parent parser.
- **x** (*list|numpy.array*) – Array.
- **key** (*str|list*) – Filter applied to keys.
- **value** (*str|list*) – Filter applied to values.

Returns Parsed dictionary.

Return type dict

7.3.2 full

full(*parent*, *x*)

Remove the empty value from each row of the input array.

Parameters

- **parent** (xlref.parser.Ref) – Parent parser.
- **x** (*list|numpy.array*) – Array.

Returns Filtered array.

Return type list

7.3.3 recursive

recursive(*parent*, *x*, *dtype=None*)

Parse recursively all values in the array.

Parameters

- **parent** (xlref.parser.Ref) – Parent parser.
- **x** (*list|numpy.array*) – Array.

Returns Parsed array.

Return type numpy.array

7.3.4 ref

ref(*parent*, *x*)

If the input is a valid reference, returns the captured values otherwise the input.

Parameters

- **parent** (xlref.parser.Ref) – Parent parser.
- **x** (*object*) – Input value.

Returns If the input is a valid reference, returns the captured values otherwise the input.

Return type object

Classes

FiltersFactory

7.3.5 FiltersFactory

class FiltersFactory

Methods

__init__	Initialize self.
clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	2-tuple; but raise KeyError if D is empty.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

__init__

FiltersFactory.**__init__**(**args*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

clear

FiltersFactory.**clear**() → None. Remove all items from D.

copy

FiltersFactory.**copy**() → a shallow copy of D

fromkeys

FiltersFactory.**fromkeys**(*value=None*, /)
Create a new dictionary with keys from iterable and values set to value.

get

FiltersFactory.**get**(*key*, *default=None*, /)
Return the value for key if key is in the dictionary, else default.

items

FiltersFactory.**items**() → a set-like object providing a view on D's items

keys

FiltersFactory.**keys**() → a set-like object providing a view on D's keys

pop

FiltersFactory.**pop**(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem

FiltersFactory.**popitem**() → (k, v), remove and return some (key, value) pair as a
2-tuple; but raise KeyError if D is empty.

setdefault

FiltersFactory.**setdefault**(*key*, *default=None*, /)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update

FiltersFactory.**update**([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values

FiltersFactory.**values**() → an object providing a view on D's values

__init__(**args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

7.4 parser

It provides xlparser reference parser class.

Functions

compile_filters	

7.4.1 compile_filters

compile_filters(*filters*, *parent*)

Classes

Range	
Ref	Reference parser

7.4.2 Range

class Range(*st_cell*, *nd_cell*)

Methods

__init__	Initialize self.
get	

__init__

Range.**__init__**(*st_cell*, *nd_cell*)
Initialize self. See help(type(self)) for accurate signature.

get

Range.**get**()

__init__(*st_cell*, *nd_cell*)
Initialize self. See help(type(self)) for accurate signature.

7.4.3 Ref

class Ref(*ref*, *parent=None*, *cache=None*)
Reference parser

Methods

__init__	Initialize self.

__init__

Ref.**__init__**(*ref*, *parent=None*, *cache=None*)
Initialize self. See help(type(self)) for accurate signature.

__init__(*ref*, *parent=None*, *cache=None*)
Initialize self. See help(type(self)) for accurate signature.

7.5 process

Defines the file processing chain model *dsp*.

Functions

load_json	Load the data excel references from files.
merge_references	Merge data excel references.
read_references	Read recursively the list of data excel references.
save_json	Save data output in an JSON file.

7.5.1 load_json

load_json(*input_fpaths*)

Load the data excel references from files.

Parameters **input_fpaths** (*list[str]*) – File paths of the json data excel references.

Returns Data excel references from files.

Return type tuple

7.5.2 merge_references

merge_references(*input_references=()*, *file_references=()*)

Merge data excel references.

Parameters

- **input_references** (*tuple*) – Data excel references from user.
- **file_references** (*tuple*) – Data excel references from files.

Returns Full list of data excel references.

Return type list

7.5.3 read_references

read_references(*references*)

Read recursively the list of data excel references.

Parameters **references** (*list*) – Full list of data excel references.

Returns Data output.

Return type list

7.5.4 save_json

save_json(*output_fpath*, *data*)

Save data output in an JSON file.

Parameters

- **output_fpath** (*str*) – Output file path.
- **data** (*list*) – Data output.

Returns File path where output are written.

Return type str

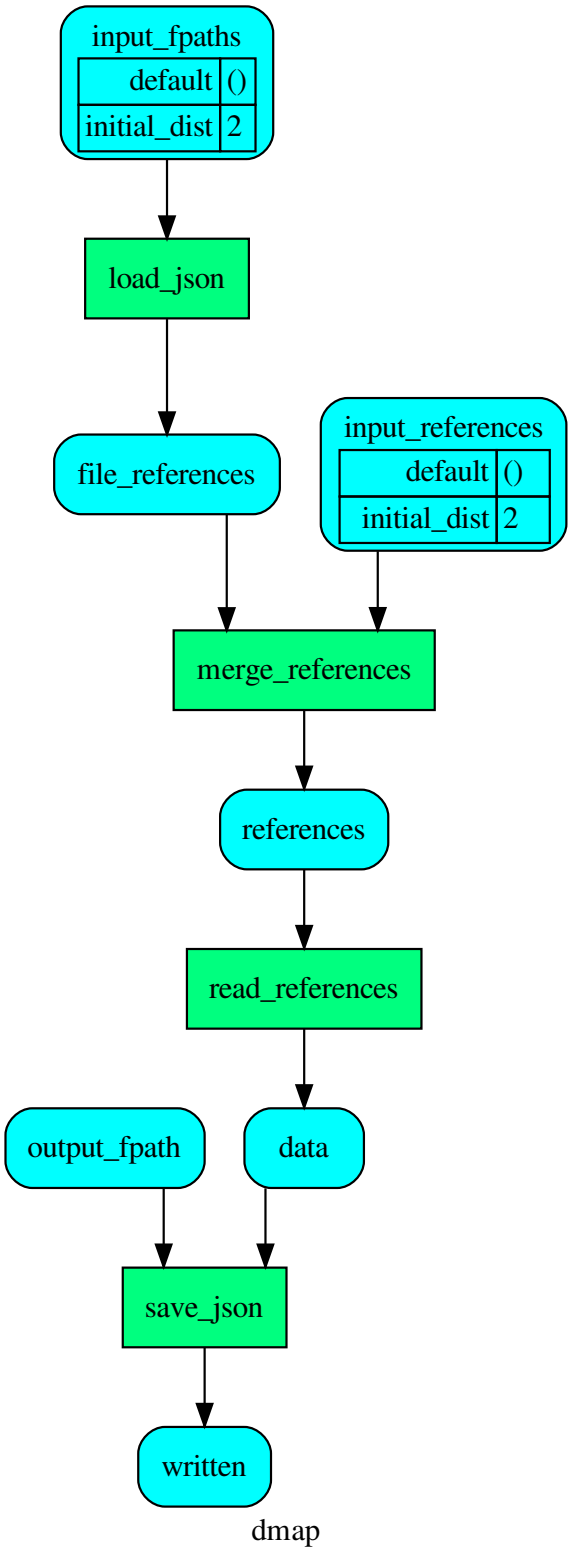
Dispatchers

dsp	Process Model.

7.5.5 dsp

dsp = Processing Model

Process Model.


input_fpaths
default ()
initial_dist 2
load_json
file_references
input_references
default ()
initial_dist 2
merge_references
references
read_references
output_fpath
data
save_json
written

dmap

Table 12: **Processing Model's data**

`data`	Data output.
`file_references`	Data excel references from files.
`input_fpaths`	File paths of the json data excel references.
`input_references`	Data excel references from user.
`output_fpath`	Output file path.
`references`	Full list of data excel references.
`written`	File path where output are written.

Table 13: **Processing Model's functions**

load_json	Load the data excel references from files.
merge_references	Merge data excel references.
read_references	Read recursively the list of data excel references.
save_json	Save data output in an JSON file.

CHAPTER

EIGHT

CHANGELOG

8.1 v1.1.2 (2021-05-19)

8.1.1 Feat

- (core): Update build scripts.

8.1.2 Fix

- (parser): Correct file regex.
- (filter): Correct VisibleDeprecationWarning.
- (doc): Correct readme version.

8.2 v1.1.1 (2021-01-04)

8.2.1 Fix

- (doc): Update copyrights.
- (parser): Use *openpyxl* to read excel files instead *xlrd*.

8.3 v1.1.0 (2020-11-05)

8.3.1 Feat

- (filters): Remove *merge* option in *dict* filter and add list filters for *key* and *value* options.

8.3.2 Fix

- (travis): Correct coverage setting.

8.4 v1.0.1 (2020-11-04)

8.4.1 Feat

- (filters): Add *merge* option in *dict* filter.

8.4.2 Fix

- (parser): Implement case insensitive parser for sheet names.

8.5 v1.0.0 (2020-04-07)

First release.

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

X

INDEX

Symbols

C

D

F

I

L

M

O

R

S

X