# ExcellentLibrary

| | |
|---|---|
| **Library version:** | 0.9.2 |
| **Library scope:** | global |
| **Named arguments:** | supported |

## Introduction

This library is built on top of *OpenPyXL* in order to bring its functionality to *Robot Framework*. The major motivation for this was to add support for *Excel 2010* (XSLX) files, which *ExcelLibrary* does not support.

## Usage

**TODO**: Showcase one full-feature, gigantic test suite covering pretty much all functionality and variety one could possibly encounter.

The OpenPyXL documentation is quite immature, so if you really need to understand the implementation better you are forced to experiment or read the source code.

## Shortcuts

**C**lose All Workbooks · **C**lose Workbook · **C**reate Sheet · **C**reate Workbook · **G**et Column Count · **G**et Row Count · **G**et Row Iterator · **L**og Opened Workbooks · **O**pen Workbook · **R**ead From Cell · **R**ead Sheet Data · **R**emove Sheet · **S**ave · **S**witch Sheet · **S**witch Workbook · **W**rite To Cell

## Keywords

| Keyword | Arguments | Documentation |
|---|---|---|
| **Close All Workbooks** | | Closes all opened workbooks.<br>Changes made to the file won't be saved automatically. Use the *Save* keyword to save the changes to the file. |
| **Close Workbook** | *alias=None* | Closes the workbook identified by the supplied alias.<br>If no alias is provided, the alias of the active workbook is used. In this case a new workbook becomes active.<br>Changes made to the file won't be saved automatically. Use the *Save* keyword to save the changes to the file. |
| **Create Sheet** | *sheet_name* | Creates a sheet in the active workbook.<br>The `name` parameter must be used to supply the name of the sheet. If the sheet already exists, a `SheetAlreadyExistsException` is raised. |
| **Create Workbook** | *file_path*, *overwrite_file_if_exists=False*, *alias=None* | Creates a new workbook and saves it to the given file path.<br>The file will also be considered opened, i.e. it will be added to the internal dictionary of opened workbooks using the supplied alias. If no alias is supplied, it will default to the file path.<br>In case the given file already exists, an `FileAlreadyExistsException` is raised. If you wish to overwrite the existing file, pass the argument `overwrite_file_if_exists=${TRUE}`.<br>*NOTE*: It is advised to supply an absolute path to avoid confusion.<br>Examples:<br>Create workbook H:\Workbook 1.xlsx # *alias* defaults to absolute file path<br>Create workbook H:\Workbook 2.xlsx alias=second workbook |
| **Get Column Count** | | Returns the number of non-empty columns in the active sheet. |
| **Get Row Count** | | Returns the number of non-empty rows in the active sheet. |
| **Get Row Iterator** | | Returns an iterator for looping over the rows in the active sheet.<br>*NOTE*: This won't be needed often and it is advised to avoid this as much as possible, since it is unfriendly to read and hacky in its use with respect to Robot Framework. |
| **Log Opened Workbooks** | *to_log=True*, *to_console=False* | Logs the dictionary in which the opened workbooks are kept.<br>If `to_log` is `True`, this keyword outputs in the log file.<br>If `to_console` is `True`, this keyword outputs on the console.<br>Note that it is perfectly fine to log to both the log file and console simultaneously. |
| **Open Workbook** | *file_path*, *alias=None*, *keep_vba=False* | Opens the workbook found at the given file path. *NOTE*: Please note that at present *XLS* files are not supported. |

The file will be added to the internal dictionary of opened workbooks using the supplied alias. If no alias is supplied, it will default to the file path. The opened workbook will also be made the active workbook.

The `file_path` parameter should point to the location of the file on the filesystem. It is advisable to make this an absolute path to avoid confusion.

The `alias` can be used to give a more practical name to your workbook, which comes in handy when working with several opened workbooks simultaneously.

If the file you want to open contains VBA (macros), please pass `keep_vba=${TRUE}` in order to preserve the VBA code.

**Warning**: make sure to explicitly switch to the sheet you want to work with by using the *Switch sheet* keyword. Contrary to expectations, the active sheet by default is not necessarily the first one in tab-order.

Examples:

| | | | | |
|---|---|---|---|---|
| Open workbook | H:\Data\Wb1.xlsx | | | # now *Wb1.xlsx* is the active workbook |
| Open workbook | H:\Data\Wb2.xlsx | alias=wb2 | | # Now *Wb2.xlsx* is the active workbook. |
| Switch workbook | H:\Data\Wb1.xlsx | | | # now *Wb1.xlsx* is the active workbook |
| Close workbook | wb2 | | | # now *wb2* is closed and *Wb1.xlsx* is set to be the active workbook |
| Close workbook | | | | # now *Wb1.xlsx* is closed because it was the active workbook |
| Open workbook | H:\Data\WbWithMacro.xlsx | alias=Macro Workbook | keep_vba=${TRUE} | # Macro's are preserved and properly saved on *Save* |

**Read From Cell** | *cell*, *cell_obj=None*, *trim=False*

Reads the data from the cell identified by the given locator.

A cell can be identified in two ways:

- *Coordinates*: provide both the row and column numbers of the cell, starting with 1.
- *A1 Notation*: provide the commonly used A1 Notation from Excel.

See the examples below for more detailed use:

| | | | |
|---|---|---|---|
| # Coordinates. | | | # no parentheses and space after comma is ok |
| ${value}= | Read from cell | 1, 2 | # coords prefix is ok |
| ${value}= | Read from cell | coords:1,3 | # parentheses are fine |
| ${value}= | Read from cell | (1,4) | |
| ${value}= | Read from cell | (1, 5) | |
| | | | |
| # A1 Notation. | | | |
| ${value}= | Read from cell | B2 | # no prefix for a1 notations is also ok |
| ${value}= | Read from cell | a1:CC2 | # with prefix is fine as well |

Note that the prefixes `coords` and `a1` are optional. Without a prefix the library is still capable of resolving which locator form you intended to use. Arguably though, using them is more explicit and therefore improves readability.

The `cell_obj` argument can be used to pass an *OpenPyXL Cell* object to read from. This is not intended for typical use.

| | | |
|---|---|---|
| | | By default the value read from the cell is obtained untouched, verbatim. To trim the surrounding whitespace you can pass the argument `trim=${TRUE}`. |
| **Read Sheet Data** | *column_names=None*, *get_column_names_from_header_row=False*, *cell_range=None*, *trim=False* | Reads all the data from the active sheet.<br><br>This keyword can output the sheet data in two formats:<br>▪ *As a list of dictionaries*. In the case column names are supplied or obtained (see relevant parameters described below), the rows will be represented through dictionaries, of which the keys will correspond to the column names.<br>▪ *As a list of lists*. If no column names are provided or obtained, each row will be read from the sheet as a list, and the returned data will, therefore, be a list of all such lists.<br><br>To use column names the following two parameters can be used:<br>▪ If `column_names` is provided it is expected to be a list which will<br><br>be used to name the columns in the supplied order.<br>▪ If `get_column_names_from_header_row` is `True`, the column names<br><br>will be read from the first row in the sheet. In this case, the first row will not be read as part of the sheet data.<br><br>*NOTE*: If both parameters are supplied, the *column_names*` list will have precedence. You will get a warning in your log when this situation occurs though.<br><br>Use `cell_range` if you want to get data from only that range in the sheet, rather than all of the data in it. The expected input form is in *A1 Notation*. For example: `A1:B3`.<br><br>If `trim` is `True`, all cell values are trimmed, i.e. the surrounding whitespace is removed.<br><br>Examples:<br><br>Read entire sheet with column names from header row<br><br>Open workbook ${PROPER EXCEL FILE} # no alias provided: defaulting to file path<br>Switch sheet Sheet 1 (with header)<br>@{data sheet}= Read sheet data get_column_names_from_header_row=${TRUE}<br>:FOR ${row} IN @{data sheet}<br>\ Log list ${row}<br>Close workbook ${PROPER EXCEL FILE}<br><br>Read sheet range without column names (trimmed)<br><br>Open workbook ${PROPER EXCEL FILE} first excel file<br>Switch sheet Sheet 1 (with header)<br>@{data sheet}= Read sheet data cell_range=A1:B3 trim=${TRUE}<br>:FOR ${row} IN @{data sheet}<br>\ Log dictionary ${row}<br>Close workbook<br><br>For more examples check out the included test suite. |
| **Remove Sheet** | *sheet_name* | Removes the sheet identified by its name from the active workbook.<br><br>The `name` parameter must be used to supply the name of the sheet. If the sheet does not exist, a `SheetNotFoundException` will be raised. |
| **Save** | | Saves the changes to the currently active workbook.<br><br>*NOTE*: When manipulating sheets/cells, you are working with object representations in memory, not the factual data on disk. Only when you choose to make the changes persistent by calling this keyword, will those changes be saved to the file. |
| **Switch Sheet** | *sheet_name* | Switches to the sheet with the supplied name within the active workbook.<br><br>Please supply the `sheet_name` parameter to identify which sheet you want to switch to. |
| **Switch Workbook** | *alias* | Switches to the workbook identified by `alias`, i.e. make that the active workbook. |

| | | |
|---|---|---|
| | | *NOTE*: You can only switch to workbooks which are opened. This keyword won't do that for you, so make sure you've opened the workbook you want to switch to using *Open workbook*.<br><br>Examples:<br><br>Opening several workbooks and switching between them<br>Open workbook ${PROPER EXCEL FILE} alias=first excel file # supply alias with or without *alias*=: both is fine<br>Open workbook ${WEIRD EXCEL FILE} second excel file # when opening a workbook it is made the active one<br>Switch workbook first excel file<br>Switch sheet Sheet 2 (no header)<br>Close workbook alias=first excel file<br>Close workbook alias=second excel file |
| **Write To Cell** | *cell*, *value*, *number_format=None* | Writes a value to the supplied cell.<br><br>For an explanation of how to identify a cell, please see the *Read From Cell* keyword documentation. For the sake of convenience I will stick with A1 Notation.<br><br>Writing a value to a cell, then, is really straight-forward:<br><br>Write To Cell B1 Hello # this is ok!<br><br>It is possible to format the cell using the `number_format` parameter. In order for this to work properly with the data you're writing, you must make sure that the data type of the latter is compatible with what the number formatting expects. For example, to format a cell as a number that's rounded to two decimals, one should write data of a type number. To format a cell to hold a date-time value, a Python date-time object should be passed in for it to function.<br><br>Some examples:<br><br>Write To Cell B1 Hello # OK<br>Write To Cell B1 ${2} # OK<br>Write to cell A1 1.233 number_format=#.# # Bad<br>Write to cell A1 ${1.233} number_format=#.# # OK<br>Write to cell C1 2018-04-01 number_format=yyyy-dd-mm # Bad<br>${now} DateTime.Get current date<br>Write to cell D1 ${now} number_format=yyyy-dd-mm # OK<br>Write to cell D1$ {now} number_format=jjjj-dd-mm # Bad<br><br>*NOTE*: The `numer_format` parameter seems to assume the US locale, so make sure to delimit numbers with dots ("."), and format your dates using `yyyy` for example rather than `jjjj` (in Dutch). Excel will honour your own locale settings anyways, so don't worry about it. |

Altogether 16 keywords.
Generated by Libdoc on 2018-09-17 14:05:37.