xlwings - Make Excel Fly!

dev

Zoomer Analytics LLC

2020 02 15

Contents

1 **1**

2 Migrate to v0.11 (Add-in) **3**

2.1 Upgrade the xlwings Python package 3
2.2 Install the add-in 3
2.3 Upgrade existing workbooks 4

3 Migrate to v0.9 **5**

3.1 Full qualification: Using collections 5
3.2 Connecting to Books 5
3.3 Active Objects 6
3.4 Round vs. Square Brackets 6
3.5 Access the underlying Library/Engine 6
3.6 Cheat sheet 6

4 **9**

4.1 9
4.2 10
4.3 10
4.4 Python 10

5 **11**

5.1 1. Python Excel 11
5.2 2. Excel Python 12
5.3 3. UDFs: Windows 13

6 **15**

6.1 Python Excel 15
6.2 Excel Python (RunPython) 16
6.3 UDFs 16

7 Syntax Overview **17**

7.1 Active Objects 17

7.2 Full qualification . . . 18
7.3 Range indexing/slicing . . . 18
7.4 Range Shortcuts . . . 18
7.5 Object Hierarchy . . . 19

8 **21**
8.1 . . . 21
8.2 . . . 22
8.3 . . . 22
8.4 NumPy arrays . . . 23
8.5 Pandas DataFrames . . . 23
8.6 Pandas Series . . . 24

9 Add-in **25**
9.1 Run main . . . 25
9.2 Installation . . . 26
9.3 Anaconda/Miniconda . . . 26
9.4 Global Settings . . . 26
9.5 Global Config: Ribbon/Config File . . . 27
9.6 Workbook Directory Config: Config file . . . 27
9.7 Workbook Config: xlwings.conf Sheet . . . 28
9.8 Alternative: Standalone VBA module . . . 28

10 VBA: RunPython **29**
10.1 xlwings add-in . . . 29
10.2 Call Python with RunPython . . . 29
10.3 Function Arguments and Return Values . . . 30

11 VBA: User Defined Functions (UDFs) **31**
11.1 One-time Excel preparations . . . 31
11.2 Workbook preparation . . . 31
11.3 A simple UDF . . . 32
11.4 Array formulas: Get efficient . . . 33
11.5 Array formulas with NumPy and Pandas . . . 34
11.6 @xw.arg and @xw.ret decorators . . . 34
11.7 Dynamic Array Formulas . . . 35
11.8 Docstrings . . . 36
11.9 The vba keyword . . . 36
11.10 Macros . . . 36
11.11 Call UDFs from VBA . . . 37
11.12 Asynchronous UDFs . . . 37

12 Debugging **39**
12.1 RunPython . . . 40
12.2 UDF debug server . . . 40

13 Matplotlib **43**
13.1 Getting started . . . 43
13.2 Full integration with Excel . . . 43

13.3 Properties . . . 44
13.4 Getting a Matplotlib figure . . . 46

14 Converters and Options **47**
14.1 Default Converter . . . 48
14.2 Built-in Converters . . . 50
14.3 Custom Converter . . . 54

15 Threading and Multiprocessing **59**
15.1 Threading . . . 59
15.2 Multiprocessing . . . 60

16 Command Line Client **63**
16.1 Quickstart . . . 63
16.2 Add-in . . . 63
16.3 RunPython . . . 64

17 Missing Features **65**
17.1 Example: Workaround to use VBA's `Range.WrapText` . . . 65

18 xlwings with other Office Apps **67**
18.1 How To . . . 67
18.2 Config . . . 68

19 Deployment **69**
19.1 Zip files . . . 69
19.2 RunFrozenPython . . . 69

20 Extensions **71**
20.1 In-Excel SQL . . . 71

21 xlwings with R and Julia **73**
21.1 R . . . 73
21.2 Julia . . . 75

22 Troubleshooting **77**
22.1 Issue: dll not found . . . 77

23 REST API **79**
23.1 Quickstart . . . 79
23.2 Run the server . . . 81
23.3 Indexing . . . 81
23.4 Range Options . . . 81
23.5 Endpoint overview . . . 82
23.6 Endpoint details . . . 82

24 Python API **111**
24.1 Top-level functions . . . 111
24.2 Object model . . . 112
24.3 UDF decorators . . . 140

25 License **143**
25.1 xlwings 143
25.2 Dependencies 144

149

CHAPTER 1

https://training.zoomeranalytics.com/p/xlwings

xlwings :)

CHAPTER 2

Migrate to v0.11 (Add-in)

This migration guide shows you how you can start using the new xlwings add-in as opposed to the old xlwings VBA module (and the old add-in that consisted of just a single import button).

2.1 Upgrade the xlwings Python package

1. Check where xlwings is currently installed

```
>>> import xlwings
>>> xlwings.__path__
```

2. If you installed xlwings with pip, for once, you should first uninstall xlwings: `pip uninstall xlwings`
3. Check the directory that you got under 1): if there are any files left over, delete the `xlwings` folder and the remaining files manually
4. Install the latest xlwings version: `pip install xlwings`
5. Verify that you have >= 0.11 by doing

```
>>> import xlwings
>>> xlwings.__version__
```

2.2 Install the add-in

1. If you have the old xlwings addin installed, find the location and remove it or overwrite it with the new version (see next step). If you installed it via the xlwings command line client,

you should be able to do: `xlwings addin remove`.

2. Close Excel. Run `xlwings addin install` from a command prompt. Reopen Excel and check if the xlwings Ribbon appears. If not, copy `xlwings.xlam` (from your xlwings installation folder under `addin\xlwings.xlam` manually into the `XLSTART` folder. You can find the location of this folder under Options > Trust Center > Trust Center Settings… > Trusted Locations, under the description `Excel default location: User StartUp`. Restart Excel and you should see the add-in.

2.3 Upgrade existing workbooks

1. Make a backup of your Excel file
2. Open the file and go to the VBA Editor (`Alt-F11`)
3. Remove the xlwings VBA module
4. Add a reference to the xlwings addin, see *Installation*
5. If you want to use workbook specific settings, add a sheet `xlwings.conf`, see *Workbook Config: xlwings.conf Sheet*

Note: To import UDFs, you need to have the reference to the xlwings add-in set!

CHAPTER 3

Migrate to v0.9

The purpose of this document is to enable you a smooth experience when upgrading to xlwings v0.9.0 and above by laying out the concept and syntax changes in detail. If you want to get an overview of the new features and bug fixes, have a look at the release notes. Note that the syntax for User Defined Functions (UDFs) didn't change.

3.1 Full qualification: Using collections

The new object model allows to specify the Excel application instance if needed:

- **old**: `xw.Range('Sheet1', 'A1', wkb=xw.Workbook('Book1'))`
- **new**: `xw.apps[0].books['Book1'].sheets['Sheet1'].range('A1')`

See *Syntax Overview* for the details of the new object model.

3.2 Connecting to Books

- **old**: `xw.Workbook()`
- **new**: `xw.Book()` or via `xw.books` if you need to control the app instance.

See for the details.

3.3 Active Objects

```
# Active app (i.e. Excel instance)
>>> app = xw.apps.active

# Active book
>>> wb = xw.books.active  # in active app
>>> wb = app.books.active  # in specific app

# Active sheet
>>> sht = xw.sheets.active  # in active book
>>> sht = wb.sheets.active  # in specific book

# Range on active sheet
>>> xw.Range('A1')  # on active sheet of active book of active app
```

3.4 Round vs. Square Brackets

Round brackets follow Excel's behavior (i.e. 1-based indexing), while square brackets use Python's 0-based indexing/slicing.

As an example, the following all reference the same range:

```
xw.apps[0].books[0].sheets[0].range('A1')
xw.apps(1).books(1).sheets(1).range('A1')
xw.apps[0].books['Book1'].sheets['Sheet1'].range('A1')
xw.apps(1).books('Book1').sheets('Sheet1').range('A1')
```

3.5 Access the underlying Library/Engine

- **old**: `xw.Range('A1').xl_range` and `xl_sheet` etc.
- **new**: `xw.Range('A1').api`, same for all other objects

This returns a `pywin32` COM object on Windows and an `appscript` object on Mac.

3.6 Cheat sheet

Note that `sht` stands for a sheet object, like e.g. (in 0.9.0 syntax): `sht = xw.books['Book1'].sheets[0]`

	v0.9.0	v0.7.2
Active Excel instance	`xw.apps.active`	unsupported
New Excel instance	`app = xw.App()`	unsupported
Get app from book	`app = wb.app`	`app = xw.Application(wb)`
Target installation (Mac)	`app = xw.App(spec=...)`	`wb = xw.Workbook(app_tar`
Hide Excel Instance	`app = xw.App(visible=False)`	`wb = xw.Workbook(app_vis`
Selected Range	`app.selection`	`wb.get_selection()`
Calculation mode	`app.calculation = 'manual'`	`app.calculation = xw.con`
All books in app	`app.books`	unsupported
Fully qualified book	`app.books['Book1']`	unsupported
Active book in active app	`xw.books.active`	`xw.Workbook.active()`
New book in active app	`wb = xw.Book()`	`wb = xw.Workbook()`
New book in specific app	`wb = app.books.add()`	unsupported
All sheets in book	`wb.sheets`	`xw.Sheet.all(wb)`
Call a macro in an addin	`app.macro('MacroName')`	unsupported
First sheet of book wb	`wb.sheets[0]`	`xw.Sheet(1, wkb=wb)`
Active sheet	`wb.sheets.active`	`xw.Sheet.active(wkb=wb)`
Add sheet	`wb.sheets.add()`	`xw.Sheet.add(wkb=wb)`
Sheet count	`wb.sheets.count` or `len(wb.sheets)`	`xw.Sheet.count(wb)`
Add chart to sheet	`chart = wb.sheets[0].charts.add()`	`chart = xw.Chart.add(she`
Existing chart	`wb.sheets['Sheet 1'].charts[0]`	`xw.Chart('Sheet 1', 1)`
Chart Type	`chart.chart_type = '3d_area'`	`chart.chart_type = xw.co`
Add picture to sheet	`wb.sheets[0].pictures.add('path/to/pic')`	`xw.Picture.add('path/to/`
Existing picture	`wb.sheets['Sheet 1'].pictures[0]`	`xw.Picture('Sheet 1', 1)`
Matplotlib	`sht.pictures.add(fig, name='x', update=True)`	`xw.Plot(fig).show('MyPlo`
Table expansion	`sht.range('A1').expand('table')`	`xw.Range(sht, 'A1', wkb=`
Vertical expansion	`sht.range('A1').expand('down')`	`xw.Range(sht, 'A1', wkb=`
Horizontal expansion	`sht.range('A1').expand('right')`	`xw.Range(sht, 'A1', wkb=`
Set name of range	`sht.range('A1').name = 'name'`	`xw.Range(sht, 'A1', wkb=`
Get name of range	`sht.range('A1').name.name`	`xw.Range(sht, 'A1', wkb=`
mock caller	`xw.Book('file.xlsm').set_mock_caller()`	`xw.Workbook.set_mock_cal`

CHAPTER 4

xlwings pip:

```
pip install xlwings
```

conda:

```
conda install xlwings
```

conda xlwings conda-forge (: https://anaconda.org/conda-forge/xlwings) pip :

```
conda install -c conda-forge xlwings
```

: Mac Excel 2016 `conda` xlwings Anaconda `$ xlwings runpython install` VBA `RunPython` `pip` xlwings

4.1

- **Windows**: `pywin32`, `comtypes`

  Windows `conda` `pip` xlwings

- **Mac**: `psutil`, `appscript`

  On Mac, the dependencies are automatically being handled if xlwings is installed with `conda` or `pip`. However, with pip, the Xcode command line tools need to be available. Mac OS X 10.4 (*Tiger*) or later is required. The recommended Python distribution for Mac is Anaconda.

With `conda` on the other hand, you'll need to manually run the command `xlwings runpython install`.

4.2

- NumPy
- Pandas
- Matplotlib
- Pillow/PIL
- Flask (for REST API only)

xlwings

4.3

Add-in

4.4 Python

xlwings CE requires at least Python 3.5. Python 2.7 is only supported with xlwings PRO.

CHAPTER 5

xlwings

5.1 1. Python Excel

```
>>> import xlwings as xw
>>> wb = xw.Book()  # this will create a new workbook
>>> wb = xw.Book('FileName.xlsx')  # connect to an existing file in the current␣
↪working directory
>>> wb = xw.Book(r'C:\path\to\file.xlsx')  # on Windows: use raw strings to␣
↪escape backslashes
```

Excel PID PID `xw.apps.keys()`

```
>>> xw.apps[10559].books['FileName.xlsx']
```

```
>>> sht = wb.sheets['Sheet1']
```

```
>>> sht.range('A1').value = 'Foo 1'
>>> sht.range('A1').value
'Foo 1'
```

, Range expanding :

```
>>> sht.range('A1').value = [['Foo 1', 'Foo 2', 'Foo 3'], [10.0, 20.0, 30.0]]
>>> sht.range('A1').expand().value
[['Foo 1', 'Foo 2', 'Foo 3'], [10.0, 20.0, 30.0]]
```

Numpy Array Pandas DataFrame

```
>>> import pandas as pd
>>> df = pd.DataFrame([[1,2], [3,4]], columns=['a', 'b'])
>>> sht.range('A1').value = df
>>> sht.range('A1').options(pd.DataFrame, expand='table').value
       a    b
0.0  1.0  2.0
1.0  3.0  4.0
```

Matplotlib

```
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure()
>>> plt.plot([1, 2, 3, 4, 5])
[<matplotlib.lines.Line2D at 0x1071706a0>]
>>> sht.pictures.add(fig, name='MyPlot', update=True)
<Picture 'MyPlot' in <Sheet [Workbook4]Sheet1>>
```

`xw.Range`

```
>>> import xlwings as xw
>>> xw.Range('A1').value = 'Foo'
>>> xw.Range('A1').value
'Foo'
```

`xw.Range` Excel book sheet

5.2 2. Excel Python

You can call Python functions either by clicking the `Run` button (new in v0.16) in the add-in or from VBA using the `RunPython` function:

The `Run` button expects a function called `main` in a Python module with the same name as your workbook. The great thing about that approach is that you don't need your workbooks to be macro-enabled, you can save it as `xlsx`.

If you want to call any Python function no matter in what module it lives or what name it has, use `RunPython`:

```
Sub HelloWorld()
    RunPython ("import hello; hello.world()")
End Sub
```

Per default, `RunPython` expects `hello.py` in the same directory as the Excel file but you can change that via config. Refer to the calling Excel book by using `xw.Book.caller`:

```
# hello.py
import numpy as np
import xlwings as xw

def world():
    wb = xw.Book.caller()
    wb.sheets[0].range('A1').value = 'Hello World!'
```

xlwings xlwings `xlwings quickstart myproject`

Add-in

5.3 3. UDFs: Windows

Python Excel

```
import xlwings as xw

@xw.func
def hello(name):
    return 'Hello {0}'.format(name)
```

Pandas DataFrame

```
import xlwings as xw
import pandas as pd

@xw.func
@xw.arg('x', pd.DataFrame)
def correl2(x):
    # x arrives as DataFrame
    return x.corr()
```

xlwings import Excel *VBA: User Defined Functions (UDFs)*

CHAPTER 6

xw.Range() book

```
>>> import xlwings as xw
>>> xw.Range('A1').value = 'something'
```

6.1 Python Excel

xw.Book() xw.books

```
>>> app = xw.App()  # or something like xw.apps[10559] for existing apps, get␣
↪the available PIDs via xw.apps.keys()
>>> app.books['Book1']
```

	xw.Book	xw.books
	`xw.Book()`	`xw.books.add()`
	`xw.Book('Book1')`	`xw.books['Book1']`
	`xw.Book(r'C:/path/to/file.xlsx')`	`xw.books.open(r'C:/path/to/file.xlsx')`

: Windows () Python
r C:\\path\\to\\file.xlsx

6.2 Excel Python (RunPython)

VBA `RunPython` `xw.Book.caller()` *Call Python with RunPython* *Debugging* Python VBA Python

6.3 UDFs

Unlike `RunPython`, UDFs don't need a call to `xw.Book.caller()`, see *VBA: User Defined Functions (UDFs)*. However, it's available (restricted to read-only though), which sometimes proves to be useful.

CHAPTER 7

Syntax Overview

The xlwings object model is very similar to the one used by VBA.

All code samples below depend on the following import:

```
>>> import xlwings as xw
```

7.1 Active Objects

```
# Active app (i.e. Excel instance)
>>> app = xw.apps.active

# Active book
>>> wb = xw.books.active  # in active app
>>> wb = app.books.active  # in specific app

# Active sheet
>>> sht = xw.sheets.active  # in active book
>>> sht = wb.sheets.active  # in specific book

# Range on active sheet
>>> xw.Range('A1')  # on active sheet of active book of active app
```

A Range can be instantiated with A1 notation, a tuple of Excel's 1-based indices, a named range or two Range objects:

```
xw.Range('A1')
xw.Range('A1:C3')
xw.Range((1,1))
xw.Range((1,1), (3,3))
xw.Range('NamedRange')
xw.Range(xw.Range('A1'), xw.Range('B2'))
```

7.2 Full qualification

Round brackets follow Excel's behavior (i.e. 1-based indexing), while square brackets use Python's 0-based indexing/slicing. As an example, the following expressions all reference the same range:

```
xw.apps[763].books[0].sheets[0].range('A1')
xw.apps(10559).books(1).sheets(1).range('A1')
xw.apps[763].books['Book1'].sheets['Sheet1'].range('A1')
xw.apps(10559).books('Book1').sheets('Sheet1').range('A1')
```

Note that the apps keys are different for you as they are the process IDs (PID). You can get the list of your PIDs via `xw.apps.keys()`.

7.3 Range indexing/slicing

Range objects support indexing and slicing, a few examples:

```
>>> rng = xw.Book().sheets[0].range('A1:D5')
>>> rng[0, 0]
 <Range [Workbook1]Sheet1!$A$1>
>>> rng[1]
 <Range [Workbook1]Sheet1!$B$1>
>>> rng[:, 3:]
<Range [Workbook1]Sheet1!$D$1:$D$5>
>>> rng[1:3, 1:3]
<Range [Workbook1]Sheet1!$B$2:$C$3>
```

7.4 Range Shortcuts

Sheet objects offer a shortcut for range objects by using index/slice notation on the sheet object. This evaluates to either `sheet.range` or `sheet.cells` depending on whether you pass a string or indices/slices:

```
>>> sht = xw.Book().sheets['Sheet1']
>>> sht['A1']
```

(continues on next page)

( )

```
<Range [Book1]Sheet1!$A$1>
>>> sht['A1:B5']
<Range [Book1]Sheet1!$A$1:$B$5>
>>> sht[0, 1]
<Range [Book1]Sheet1!$B$1>
>>> sht[:10, :10]
<Range [Book1]Sheet1!$A$1:$J$10>
```

7.5 Object Hierarchy

The following shows an example of the object hierarchy, i.e. how to get from an app to a range object and all the way back:

```
>>> rng = xw.apps[10559].books[0].sheets[0].range('A1')
>>> rng.sheet.book.app
<Excel App 10559>
```

CHAPTER 8

xlwings options *Converters and Options*

xlwings

```
>>> import xlwings as xw
```

8.1

xlwings float, unicode, None datetime

```
>>> import datetime as dt
>>> sht = xw.Book().sheets[0]
>>> sht.range('A1').value = 1
>>> sht.range('A1').value
1.0
>>> sht.range('A2').value = 'Hello'
>>> sht.range('A2').value
'Hello'
>>> sht.range('A3').value is None
True
>>> sht.range('A4').value = dt.datetime(2000, 1, 1)
>>> sht.range('A4').value
datetime.datetime(2000, 1, 1, 0, 0)
```

8.2

- n x 1 1 x n Python

```
>>> sht = xw.Book().sheets[0]
>>> sht.range('A1').value = [[1],[2],[3],[4],[5]]  # Column orientation␣
↪(nested list)
>>> sht.range('A1:A5').value
[1.0, 2.0, 3.0, 4.0, 5.0]
>>> sht.range('A1').value = [1, 2, 3, 4, 5]
>>> sht.range('A1:E1').value
[1.0, 2.0, 3.0, 4.0, 5.0]
```

ndim 1

```
>>> sht.range('A1').options(ndim=1).value
[1.0]
```

: transpose : `sht.range('A1').options(transpose=True).value = [1,2,3,4]`

- n x 1 1 x n range ndim 2

```
>>> sht.range('A1:A5').options(ndim=2).value
[[1.0], [2.0], [3.0], [4.0], [5.0]]
>>> sht.range('A1:E1').options(ndim=2).value
[[1.0, 2.0, 3.0, 4.0, 5.0]]
```

- /

```
>>> sht.range('A10').value = [['Foo 1', 'Foo 2', 'Foo 3'], [10, 20, 30]]
>>> sht.range((10,1),(11,3)).value
[['Foo 1', 'Foo 2', 'Foo 3'], [10.0, 20.0, 30.0]]
```

: Excel Excel `sht.range('A1').value = [[1,2],[3,4]]` `sht.range('A1').value = [1, 2]` `sht.range('A2').value = [3, 4]`

8.3

`expand()` `options` `expand` `expand()` range() `options(expand='table')`

```
>>> sht = xw.Book().sheets[0]
>>> sht.range('A1').value = [[1,2], [3,4]]
>>> rng1 = sht.range('A1').expand('table')  # or just .expand()
>>> rng2 = sht.range('A1').options(expand='table')
>>> rng1.value
[[1.0, 2.0], [3.0, 4.0]]
>>> rng2.value
[[1.0, 2.0], [3.0, 4.0]]
>>> sht.range('A3').value = [5, 6]
>>> rng1.value
[[1.0, 2.0], [3.0, 4.0]]
>>> rng2.value
[[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]]
```

`'table'` `down`

: `expand()` `sht.range('NamedRange').expand().value`

8.4 NumPy arrays

NumPy `nan` `None` Numpy options `convert=np.array`

```
>>> import numpy as np
>>> sht = xw.Book().sheets[0]
>>> sht.range('A1').value = np.eye(3)
>>> sht.range('A1').options(np.array, expand='table').value
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

8.5 Pandas DataFrames

```
>>> sht = xw.Book().sheets[0]
>>> df = pd.DataFrame([[1.1, 2.2], [3.3, None]], columns=['one', 'two'])
>>> df
   one  two
0  1.1  2.2
1  3.3  NaN
>>> sht.range('A1').value = df
>>> sht.range('A1:C3').options(pd.DataFrame).value
```

(continues on next page)

( )

```
   one  two
0  1.1  2.2
1  3.3  NaN
# options: work for reading and writing
>>> sht.range('A5').options(index=False).value = df
>>> sht.range('A9').options(index=False, header=False).value = df
```

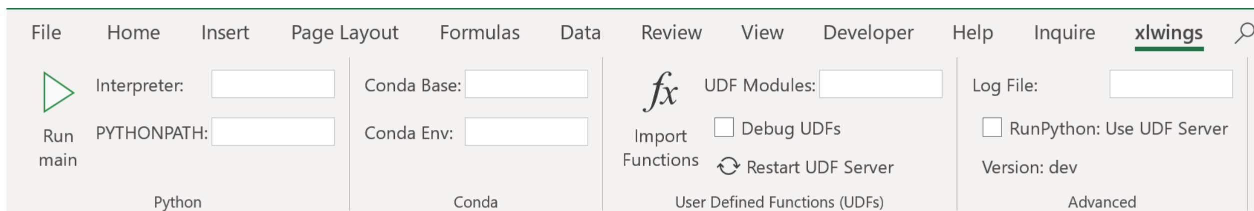
8.6 Pandas Series

```
>>> import pandas as pd
>>> import numpy as np
>>> sht = xw.Book().sheets[0]
>>> s = pd.Series([1.1, 3.3, 5., np.nan, 6., 8.], name='myseries')
>>> s
0    1.1
1    3.3
2    5.0
3    NaN
4    6.0
5    8.0
Name: myseries, dtype: float64
>>> sht.range('A1').value = s
>>> sht.range('A1:B7').options(pd.Series).value
0    1.1
1    3.3
2    5.0
3    NaN
4    6.0
5    8.0
Name: myseries, dtype: float64
```

: Numpy Array Pandas DataFrame Excel

CHAPTER 9

Add-in



The xlwings add-in is the preferred way to be able to use the `Run main` button, `RunPython` or `UDFs`. Note that you don't need an add-in if you just want to manipulate Excel from Python via xlwings.

: The ribbon of the add-in is compatible with Excel >= 2007 on Windows and >= 2016 on Mac. On Mac, all UDF related functionality is not available.

: The add-in is password protected with the password `xlwings`. For debugging or to add new extensions, you need to unprotect it.

9.1 Run main

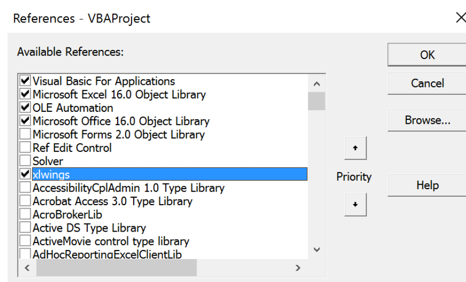
0.16.0 .

The `Run main` button is the easiest to run your Python code: It runs a function called `main` in a Python module that has the same name as your workbook. This allows you to save your workbook as `xlsx` without enabling macros. The `xlwings quickstart` command will create a workbook that will automatically work with the `Run` button.

9.2 Installation

To install the add-in, it's easiest to use the command line client: `xlwings addin install`. Technically, this copies the add-in from Python's installation directory to Excel's `XLSTART` folder. If you encounter issues, then you can also download the add-in (`xlwings.xlam`) from the GitHub Release page (make sure you download the same version as the version of the Python package). Once downloaded, you can install the add-in by going to `Developer > Excel Add-in > Browse`. If you don't see `Developer` as tab in your ribbon, make sure to activate the tab first under `File > Options > Customize Ribbon` (Mac: `Cmd + , > Ribbon & Toolbar`).

Then, to use `RunPython` or `UDFs` in a workbook, you need to set a reference to `xlwings` in the VBA editor, see screenshot (Windows: `Tools > References...`, Mac: it's on the lower left corner of the VBA editor). Note that when you create a workbook via `xlwings quickstart`, the reference is already set.



9.3 Anaconda/Miniconda

If you use Anaconda or Miniconda, you will need to set your `Conda Base` and `Conda Env` settings, as you will otherwise get errors when using `NumPy` etc. See next section.

9.4 Global Settings

While the defaults will often work out-of-the box, you can change the global settings directly in the add-in:

- `Interpreter`: This is the path to the Python interpreter. This works also with virtual or conda envs on Mac. If you use conda envs on Windows, then use `Conda Base` and `Conda Env` below instead. Examples: `"C:\Python35\pythonw.exe"` or `"/usr/local/bin/python3.5"`. An empty field defaults to `pythonw` that expects the interpreter to be set in the `PATH` on Windows or `.bash_profile` on Mac.
- `PYTHONPATH`: If the source file of your code is not found, add the path here.
- `Conda Base`: If you are on Windows and use Anaconda or Miniconda, then type here the path to your installation, e.g. `C:\Users\Username\Miniconda3` or `%USERPROFILE%\Anaconda`. NOTE that you need at least conda 4.6! You also need to set `Conda Env`, see next point.

- `Conda Env`: If you are on Windows and use Anaconda or Miniconda, type here the name of your conda env, e.g. `base` for the base installation or `myenv` for a conda env with the name `myenv`. Note that this requires you to either leave the `Interpreter` blank or set it to one of `python` or `pythonw`.
- `UDF Modules`: Names of Python modules (without .py extension) from which the UDFs are being imported. Separate multiple modules by ";". Example: `UDF_MODULES = "common_udfs;myproject"` The default imports a file in the same directory as the Excel spreadsheet with the same name but ending in `.py`.
- `Debug UDFs`: Check this box if you want to run the xlwings COM server manually for debugging, see *Debugging*.
- `RunPython: Use UDF Server`: Uses the same COM Server for RunPython as for UDFs. This will be faster, as the interpreter doesn't shut down after each call.
- `Restart UDF Server`: This shuts down the UDF Server/Python interpreter. It'll be restarted upon the next function call.

: If you use `Conda Base` and `Conda Env` with UDFs, you currently can't hide the command prompt that pops up. You can still control if the output is printed to the command prompt or not though by setting the `Interpreter` to `python` or `pythonw`, respectively.

9.5 Global Config: Ribbon/Config File

The settings in the xlwings Ribbon are stored in a config file that can also be manipulated externally. The location is

- Windows: `.xlwings\xlwings.conf` in your user folder
- macOS: `~/Library/Containers/com.microsoft.Excel/Data/xlwings.conf`

The format is as follows (keys are uppercase):

```
"INTERPRETER","pythonw"
"PYTHONPATH",""
```

9.6 Workbook Directory Config: Config file

The global settings of the Ribbon/Config file can be overridden for one or more workbooks by creating a `xlwings.conf` file in the workbook's directory.

9.7 Workbook Config: xlwings.conf Sheet

Workbook specific settings will override global (Ribbon) and workbook directory config files: Workbook specific settings are set by listing the config key/value pairs in a sheet with the name `xlwings.conf`. When you create a new project with `xlwings quickstart`, it'll already have such a sheet but you need to rename it to `xlwings.conf` to make it active.

	A	B
1	Interpreter	pythonw
2	PYTHONPATH	
3	UDF Modules	
4	Debug UDFs	FALSE
5	Log File	
6	Use UDF Server	FALSE

9.8 Alternative: Standalone VBA module

Sometimes it might be useful to run xlwings code without having to install an add-in first. To do so, you need to use the `standalone` option when creating a new project: `xlwings quickstart myproject --standalone`.

This will add the content of the add-in as a single VBA module so you don't need to set a reference to the add-in anymore. It will still read in the settings from your `xlwings.conf` if you don't override them by using a sheet with the name `xlwings.conf`.

CHAPTER 10

VBA: RunPython

10.1 xlwings add-in

To get access to `Run main` (new in v0.16) button or the `RunPython` VBA function, you'll need the xlwings addin (or VBA module), see *Add-in*.

For new projects, the easiest way to get started is by using the command line client with the quickstart command, see *Command Line Client* for details:

```
$ xlwings quickstart myproject
```

10.2 Call Python with RunPython

In the VBA Editor (`Alt-F11`), write the code below into a VBA module. `xlwings quickstart` automatically adds a new module with a sample call. If you rather want to start from scratch, you can add new module via `Insert > Module`.

```
Sub HelloWorld()
    RunPython ("import hello; hello.world()")
End Sub
```

This calls the following code in `hello.py`:

```
# hello.py
import numpy as np
import xlwings as xw

```

(continues on next page)

( )

```
def world():
    wb = xw.Book.caller()
    wb.sheets[0].range('A1').value = 'Hello World!'
```

You can then attach `HelloWorld` to a button or run it directly in the VBA Editor by hitting `F5`.

: Place `xw.Book.caller()` within the function that is being called from Excel and not outside as global variable. Otherwise it prevents Excel from shutting down properly upon exiting and leaves you with a zombie process when you use `OPTIMIZED_CONNECTION = True`.

10.3 Function Arguments and Return Values

While it's technically possible to include arguments in the function call within `RunPython`, it's not very convenient. Also, `RunPython` does not allow you to return values. To overcome these issue, use UDFs, see *VBA: User Defined Functions (UDFs)* - however, this is currently limited to Windows only.

CHAPTER 11

VBA: User Defined Functions (UDFs)

This tutorial gets you quickly started on how to write User Defined Functions.

:

- UDFs are currently only available on Windows.
- For details of how to control the behaviour of the arguments and return values, have a look at *Converters and Options*.
- For a comprehensive overview of the available decorators and their options, check out the corresponding API docs: *UDF decorators*.

11.1 One-time Excel preparations

1) Enable `Trust access to the VBA project object model` under `File > Options > Trust Center > Trust Center Settings > Macro Settings`

2) Install the add-in via command prompt: `xlwings addin install` (see *Add-in*).

11.2 Workbook preparation

The easiest way to start a new project is to run `xlwings quickstart myproject` on a command prompt (see *Command Line Client*). This automatically adds the xlwings reference to the generated workbook.

11.3 A simple UDF

The default addin settings expect a Python source file in the way it is created by `quickstart`:

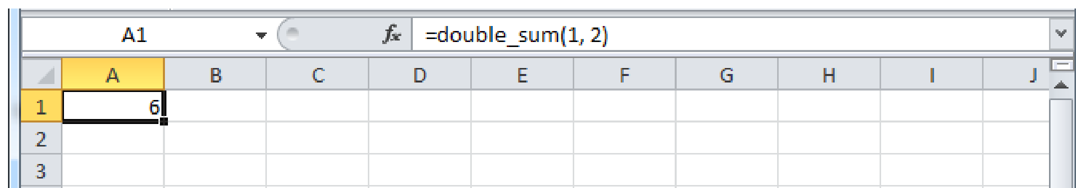
- in the same directory as the Excel file
- with the same name as the Excel file, but with a `.py` ending instead of `.xlsm`.

Alternatively, you can point to a specific module via `UDF Modules` in the xlwings ribbon.

Let's assume you have a Workbook `myproject.xlsm`, then you would write the following code in `myproject.py`:

```
import xlwings as xw

@xw.func
def double_sum(x, y):
    """Returns twice the sum of the two arguments"""
    return 2 * (x + y)
```

- Now click on `Import Python UDFs` in the xlwings tab to pick up the changes made to `myproject.py`.
- Enter the formula `=double_sum(1, 2)` into a cell and you will see the correct result:



- The docstring (in triple-quotes) will be shown as function description in Excel.

:

- You only need to re-import your functions if you change the function arguments or the function name.
- Code changes in the actual functions are picked up automatically (i.e. at the next calculation of the formula, e.g. triggered by `Ctrl-Alt-F9`), but changes in imported modules are not. This is the very behaviour of how Python imports work. If you want to make sure everything is in a fresh state, click `Restart UDF Server`.
- The `@xw.func` decorator is only used by xlwings when the function is being imported into Excel. It tells xlwings for which functions it should create a VBA wrapper function, otherwise it has no effect on how the functions behave in Python.

11.4 Array formulas: Get efficient

Calling one big array formula in Excel is much more efficient than calling many single-cell formulas, so it's generally a good idea to use them, especially if you hit performance problems.

You can pass an Excel Range as a function argument, as opposed to a single cell and it will show up in Python as list of lists.

For example, you can write the following function to add 1 to every cell in a Range:

```
@xw.func
def add_one(data):
    return [[cell + 1 for cell in row] for row in data]
```

To use this formula in Excel,

- Click on `Import Python UDFs` again
- Fill in the values in the range `A1:B2`
- Select the range `D1:E2`
- Type in the formula `=add_one(A1:B2)`
- Press `Ctrl+Shift+Enter` to create an array formula. If you did everything correctly, you'll see the formula surrounded by curly braces as in this screenshot:

	A	B	C	D	E	F	G	H	I	J
1	1	2		2	3					
2	3	4		4	5					
3										

D1 {=add_one(A1:B2)}

11.4.1 Number of array dimensions: ndim

The above formula has the issue that it expects a "two dimensional" input, e.g. a nested list of the form `[[1, 2], [3, 4]]`. Therefore, if you would apply the formula to a single cell, you would get the following error: `TypeError: 'float' object is not iterable`.

To force Excel to always give you a two-dimensional array, no matter whether the argument is a single cell, a column/row or a two-dimensional Range, you can extend the above formula like this:

```
@xw.func
@xw.arg('data', ndim=2)
def add_one(data):
    return [[cell + 1 for cell in row] for row in data]
```

11.5 Array formulas with NumPy and Pandas

Often, you'll want to use NumPy arrays or Pandas DataFrames in your UDF, as this unlocks the full power of Python's ecosystem for scientific computing.

To define a formula for matrix multiplication using numpy arrays, you would define the following function:

```
import xlwings as xw
import numpy as np

@xw.func
@xw.arg('x', np.array, ndim=2)
@xw.arg('y', np.array, ndim=2)
def matrix_mult(x, y):
    return x @ y
```

: If you are not on Python >= 3.5 with NumPy >= 1.10, use `x.dot(y)` instead of `x @ y`.

A great example of how you can put Pandas at work is the creation of an array-based `CORREL` formula. Excel's version of `CORREL` only works on 2 datasets and is cumbersome to use if you want to quickly get the correlation matrix of a few time-series, for example. Pandas makes the creation of an array-based `CORREL2` formula basically a one-liner:

```
import xlwings as xw
import pandas as pd

@xw.func
@xw.arg('x', pd.DataFrame, index=False, header=False)
@xw.ret(index=False, header=False)
def CORREL2(x):
    """Like CORREL, but as array formula for more than 2 data sets"""
    return x.corr()
```

11.6 @xw.arg and @xw.ret decorators

These decorators are to UDFs what the `options` method is to `Range` objects: they allow you to apply converters and their options to function arguments (`@xw.arg`) and to the return value (`@xw.ret`). For example, to convert the argument `x` into a pandas DataFrame and suppress the index when returning it, you would do the following:

```
@xw.func
@xw.arg('x', pd.DataFrame)
@xw.ret(index=False)
```

(continues on next page)

( )

```
def myfunction(x):
    # x is a DataFrame, do something with it
    return x
```

For further details see the *Converters and Options* documentation.

11.7 Dynamic Array Formulas

: If your version of Excel supports the new native dynamic arrays, then you don't have to do anything special, and you shouldn't use the `expand` decorator! To check if your version of Excel supports it, see if you have the `=UNIQUE()` formula available. Native dynamic arrays were introduced in Office 365 Insider Fast at the end of September 2018.

As seen above, to use Excel's array formulas, you need to specify their dimensions up front by selecting the result array first, then entering the formula and finally hitting `Ctrl-Shift-Enter`. In practice, it often turns out to be a cumbersome process, especially when working with dynamic arrays such as time series data. Since v0.10, xlwings offers dynamic UDF expansion:

This is a simple example that demonstrates the syntax and effect of UDF expansion:

```
import numpy as np

@xw.func
@xw.ret(expand='table')
def dynamic_array(r, c):
    return np.random.randn(int(r), int(c))
```

B4: =dynamic_array(B2,C2)

	A	B	C	D	E
1		rows:	columns:		
2		5	2		
3					
4		2.01156647	-0.0985618		
5		-0.2152179	-0.7541961		
6		0.37168657	-0.1978662		
7		-1.0643897	1.37592295		
8		0.5272535	-0.0508628		
9					

B4: =dynamic_array(B2,C2)

	A	B	C	D	E	F
1		rows:	columns:			
2		2	5			
3						
4		-0.6788379	-1.0009999	-0.6342434	-0.9362773	1.02582914
5		-2.1803953	0.18511092	0.3121721	0.20600051	0.3799863
6						

:

- Expanding array formulas will overwrite cells without prompting
- Pre v0.15.0 doesn't allow to have volatile functions as arguments, e.g. you cannot use functions like `=TODAY()` as arguments. Starting with v0.15.0, you can use volatile functions as input, but the UDF will be called more than 1x.
- Dynamic Arrays have been refactored with v0.15.0 to be proper legacy arrays: To edit a dynamic array with xlwings >= v0.15.0, you need to hit `Ctrl-Shift-Enter` while in the top left cell. Note that you don't have to do that when you enter the formula for the first time.

11.8 Docstrings

The following sample shows how to include docstrings both for the function and for the arguments x and y that then show up in the function wizard in Excel:

```
import xlwings as xw

@xw.func
@xw.arg('x', doc='This is x.')
@xw.arg('y', doc='This is y.')
def double_sum(x, y):
    """Returns twice the sum of the two arguments"""
    return 2 * (x + y)
```

11.9 The vba keyword

It's often helpful to get the address of the calling cell. Right now, one of the easiest ways to accomplish this is to use the `vba` keyword. `vba`, in fact, allows you to access any available VBA expression e.g. `Application`. Note, however, that currently you're acting directly on the pywin32 COM object:

```
@xw.func
@xw.arg('xl_app', vba='Application')
def get_caller_address(xl_app):
    return xl_app.Caller.Address
```

11.10 Macros

On Windows, as alternative to calling macros via *RunPython*, you can also use the `@xw.sub` decorator:

```
import xlwings as xw

@xw.sub
def my_macro():
    """Writes the name of the Workbook into Range("A1") of Sheet 1"""
    wb = xw.Book.caller()
    wb.sheets[0].range('A1').value = wb.name
```

After clicking on `Import Python UDFs`, you can then use this macro by executing it via `Alt + F8` or by binding it e.g. to a button. To to the latter, make sure you have the `Developer` tab selected under `File > Options > Customize Ribbon`. Then, under the `Developer` tab, you can insert a button via `Insert > Form Controls`. After drawing the button, you will be prompted to assign a macro to it and you can select `my_macro`.

11.11 Call UDFs from VBA

Imported functions can also be used from VBA. For example, for a function returning a 2d array:

```
Sub MySub()

Dim arr() As Variant
Dim i As Long, j As Long

    arr = my_imported_function(...)

    For j = LBound(arr, 2) To UBound(arr, 2)
        For i = LBound(arr, 1) To UBound(arr, 1)
            Debug.Print "(" & i & "," & j & ")", arr(i, j)
        Next i
    Next j

End Sub
```

11.12 Asynchronous UDFs

v0.14.0 .

xlwings offers an easy way to write asynchronous functions in Excel. Asynchronous functions return immediately with `#N/A waiting...`. While the function is waiting for its return value, you can use Excel to do other stuff and whenever the return value is available, the cell value will be updated.

The only available mode is currently `async_mode='threading'`, meaning that it's useful for I/O-bound tasks, for example when you fetch data from an API over the web.

You make a function asynchronous simply by giving it the respective argument in the function decorator. In this example, the time consuming I/O-bound task is simulated by using `time.sleep`:

```
import xlwings as xw
import time

@xw.func(async_mode='threading')
def myfunction(a):
    time.sleep(5)  # long running tasks
    return a
```

You can use this function like any other xlwings function, simply by putting `=myfunction("abcd")` into a cell (after you have imported the function, off course).

Note that xlwings doesn't use the native asynchronous functions that were introduced with Excel 2010, so xlwings asynchronous functions are supported with any version of Excel.

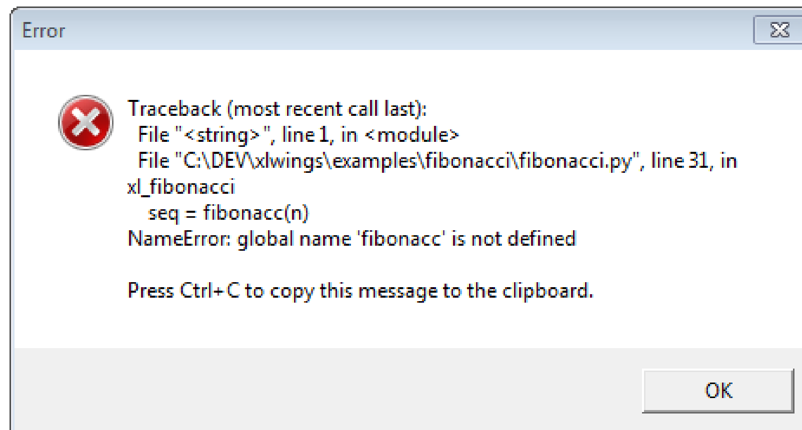
CHAPTER 12

Debugging

Since xlwings runs in every Python environment, you can use your preferred way of debugging.

- **RunPython**: When calling Python through `RunPython`, you can set a `mock_caller` to make it easy to switch back and forth between calling the function from Excel and Python.
- **UDFs**: For debugging User Defined Functions, xlwings offers a convenient debugging server

To begin with, Excel will show Python errors in a Message Box:



: On Mac, if the `import` of a module/package fails before `xlwings` is imported, the popup will not be shown and the StatusBar will not be reset. However, the error will still be logged in the log file (`/Users/<User>/Library/Containers/com.microsoft.Excel/Data/xlwings.log`).

12.1 RunPython

Consider the following sample code of your Python source code `my_module.py`:

```python
# my_module.py
import os
import xlwings as xw

def my_macro():
    wb = xw.Book.caller()
    wb.sheets[0].range('A1').value = 1

if __name__ == '__main__':
    # Expects the Excel file next to this source file, adjust accordingly.
    xw.Book('myfile.xlsm').set_mock_caller()
    my_macro()
```

`my_macro()` can now easily be run from Python for debugging and from Excel via `RunPython` without having to change the source code:

```vba
Sub my_macro()
    RunPython ("import my_module; my_module.my_macro()")
End Sub
```

12.2 UDF debug server

Windows only: To debug UDFs, just check the `Debug UDFs` in the *Add-in*, at the top of the xlwings VBA module. Then add the following lines at the end of your Python source file and run it. Depending on which IDE you use, you might need to run the code in debug mode (e.g. in case you're using PyCharm or PyDev):

```python
if __name__ == '__main__':
    xw.serve()
```

When you recalculate the Sheet (`Ctrl-Alt-F9`), the code will stop at breakpoints or output any print calls that you may have.

The following screenshot shows the code stopped at a breakpoint in the community version of PyCharm:

: When running the debug server from a command prompt, there is currently no gracious way to terminate it, but closing the command prompt will kill it.

CHAPTER 13

Matplotlib

Using *pictures.add()*, it is easy to paste a Matplotlib plot as picture in Excel.

13.1 Getting started

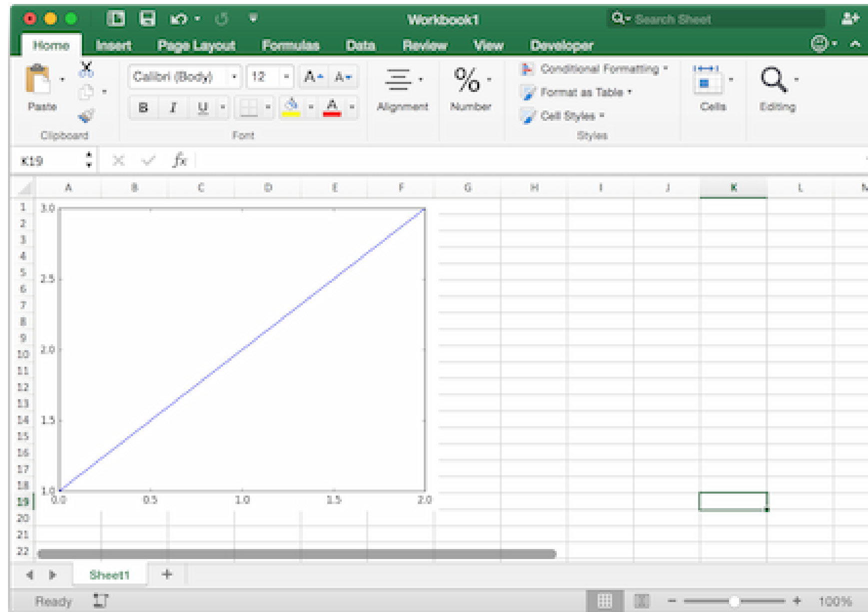
The easiest sample boils down to:

```
>>> import matplotlib.pyplot as plt
>>> import xlwings as xw

>>> fig = plt.figure()
>>> plt.plot([1, 2, 3])

>>> sht = xw.Book().sheets[0]
>>> sht.pictures.add(fig, name='MyPlot', update=True)
```

: If you set `update=True`, you can resize and position the plot on Excel: subsequent calls to `pictures.add()` with the same name (`'MyPlot'`) will update the picture without changing its position or size.

13.2 Full integration with Excel

Calling the above code with *RunPython* and binding it e.g. to a button is straightforward and works cross-platform.



However, on Windows you can make things feel even more integrated by setting up a *UDF* along the following lines:

```
@xw.func
def myplot(n):
    sht = xw.Book.caller().sheets.active
    fig = plt.figure()
    plt.plot(range(int(n)))
    sht.pictures.add(fig, name='MyPlot', update=True)
    return 'Plotted with n={}'.format(n)
```

If you import this function and call it from cell B2, then the plot gets automatically updated when cell B1 changes:

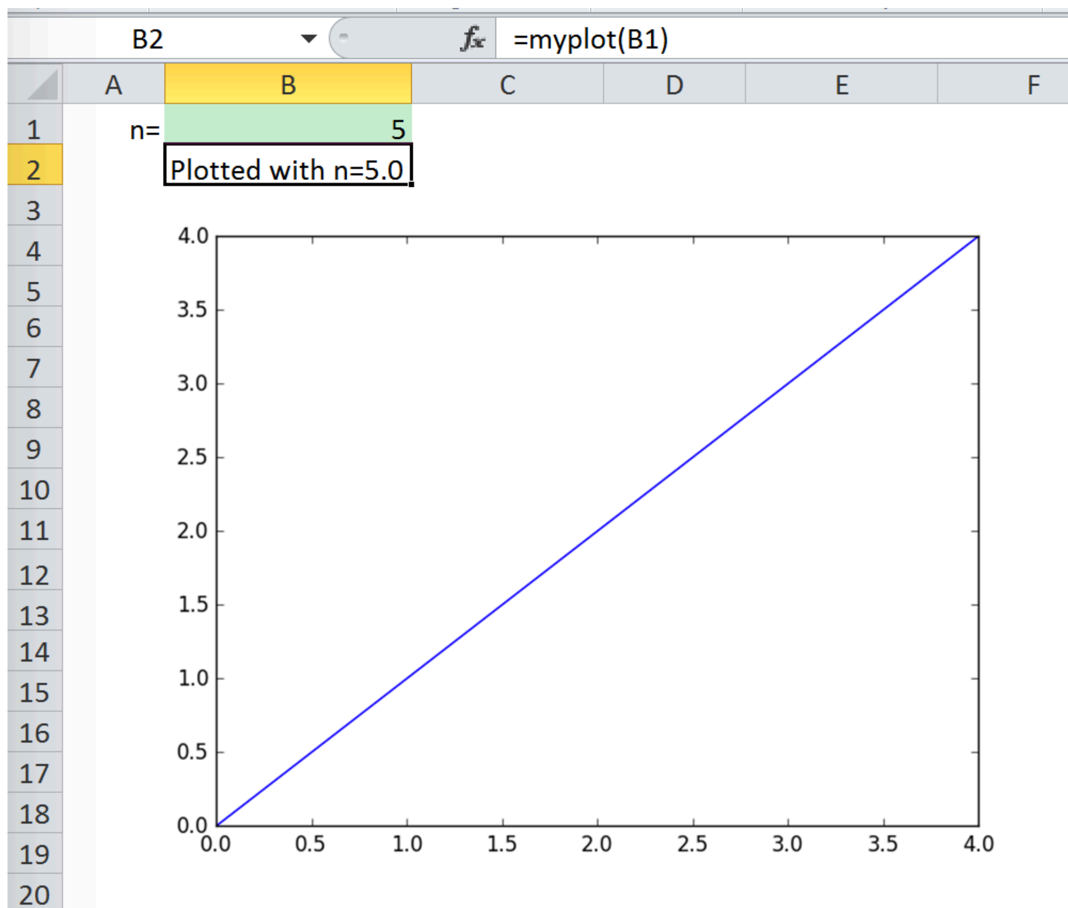
13.3 Properties

Size, position and other properties can either be set as arguments within *pictures.add()*, or by manipulating the picture object that is returned, see *xlwings.Picture()*.

For example:

```
>>> sht = xw.Book().sheets[0]
>>> sht.pictures.add(fig, name='MyPlot', update=True,
                     left=sht.range('B5').left, top=sht.range('B5').top)
```

or:


B2
=myplot(B1)
A
B
C
D
E
F
n=
5
Plotted with n=5.0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

```
>>> plot = sht.pictures.add(fig, name='MyPlot', update=True)
>>> plot.height /= 2
>>> plot.width /= 2
```

13.4 Getting a Matplotlib figure

Here are a few examples of how you get a matplotlib `figure` object:

- via PyPlot interface:

  ```
  import matplotlib.pyplot as plt
  fig = plt.figure()
  plt.plot([1, 2, 3, 4, 5])
  ```

  or:

  ```
  import matplotlib.pyplot as plt
  plt.plot([1, 2, 3, 4, 5])
  fig = plt.gcf()
  ```

- via object oriented interface:

  ```
  from matplotlib.figure import Figure
  fig = Figure(figsize=(8, 6))
  ax = fig.add_subplot(111)
  ax.plot([1, 2, 3, 4, 5])
  ```

- via Pandas:

  ```
  import pandas as pd
  import numpy as np

  df = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c', 'd'])
  ax = df.plot(kind='bar')
  fig = ax.get_figure()
  ```

CHAPTER 14

Converters and Options

Introduced with v0.7.0, converters define how Excel ranges and their values are converted both during **reading** and **writing** operations. They also provide a consistent experience across **xlwings.Range** objects and **User Defined Functions** (UDFs).

Converters are explicitely set in the `options` method when manipulating `Range` objects or in the `@xw.arg` and `@xw.ret` decorators when using UDFs. If no converter is specified, the default converter is applied when reading. When writing, xlwings will automatically apply the correct converter (if available) according to the object's type that is being written to Excel. If no converter is found for that type, it falls back to the default converter.

All code samples below depend on the following import:

```
>>> import xlwings as xw
```

Syntax:

	xw.Range	**UDFs**
reading	`xw.Range.options(convert=None, **kwargs).value`	`@arg('x', convert=None, **kwargs)`
writing	`xw.Range.options(convert=None, **kwargs).value = myvalue`	`@ret(convert=None, **kwargs)`

: Keyword arguments (`kwargs`) may refer to the specific converter or the default converter. For example, to set the `numbers` option in the default converter and the `index` option in the DataFrame converter, you would write:

```
xw.Range('A1:C3').options(pd.DataFrame, index=False, numbers=int).value
```

14.1 Default Converter

If no options are set, the following conversions are performed:

- single cells are read in as `floats` in case the Excel cell holds a number, as `unicode` in case it holds text, as `datetime` if it contains a date and as `None` in case it is empty.
- columns/rows are read in as lists, e.g. `[None, 1.0, 'a string']`
- 2d cell ranges are read in as list of lists, e.g. `[[None, 1.0, 'a string'], [None, 2.0, 'another string']]`

The following options can be set:

- **ndim**

  Force the value to have either 1 or 2 dimensions regardless of the shape of the range:

```
>>> import xlwings as xw
>>> sht = xw.Book().sheets[0]
>>> sht.range('A1').value = [[1, 2], [3, 4]]
>>> sht.range('A1').value
1.0
>>> sht.range('A1').options(ndim=1).value
[1.0]
>>> sht.range('A1').options(ndim=2).value
[[1.0]]
>>> sht.range('A1:A2').value
[1.0 3.0]
>>> sht.range('A1:A2').options(ndim=2).value
[[1.0], [3.0]]
```

- **numbers**

  By default cells with numbers are read as `float`, but you can change it to `int`:

```
>>> sht.range('A1').value = 1
>>> sht.range('A1').value
1.0
>>> sht.range('A1').options(numbers=int).value
1
```

  Alternatively, you can specify any other function or type which takes a single float argument.

  Using this on UDFs looks like this:

```
@xw.func
@xw.arg('x', numbers=int)
def myfunction(x):
    # all numbers in x arrive as int
    return x
```

Note: Excel always stores numbers internally as floats, which is the reason why the *int* converter rounds numbers first before turning them into integers. Otherwise it could happen that e.g. 5 might be returned as 4 in case it is represented as a floating point number that is slightly smaller than 5. Should you require Python's original *int* in your converter, use *raw int* instead.

- **dates**

  By default cells with dates are read as `datetime.datetime`, but you can change it to `datetime.date`:

  - Range:

    ```
    >>> import datetime as dt
    >>> sht.range('A1').options(dates=dt.date).value
    ```

  - UDFs: `@xw.arg('x', dates=dt.date)`

  Alternatively, you can specify any other function or type which takes the same keyword arguments as `datetime.datetime`, for example:

  ```
  >>> my_date_handler = lambda year, month, day, **kwargs: "%04i-%02i-%02i" %
  ↪(year, month, day)
  >>> sht.range('A1').options(dates=my_date_handler).value
  '2017-02-20'
  ```

- **empty**

  Empty cells are converted per default into `None`, you can change this as follows:

  - Range: `>>> sht.range('A1').options(empty='NA').value`
  - UDFs: `@xw.arg('x', empty='NA')`

- **transpose**

  This works for reading and writing and allows us to e.g. write a list in column orientation to Excel:

  - Range: `sht.range('A1').options(transpose=True).value = [1, 2, 3]`
  - UDFs:

    ```
    @xw.arg('x', transpose=True)
    @xw.ret(transpose=True)
    def myfunction(x):
        # x will be returned unchanged as transposed both when reading and
    ↪writing
        return x
    ```

- **expand**

  This works the same as the Range properties `table`, `vertical` and `horizontal` but is only evaluated when getting the values of a Range:

```
>>> import xlwings as xw
>>> sht = xw.Book().sheets[0]
>>> sht.range('A1').value = [[1,2], [3,4]]
>>> rng1 = sht.range('A1').expand()
>>> rng2 = sht.range('A1').options(expand='table')
>>> rng1.value
[[1.0, 2.0], [3.0, 4.0]]
>>> rng2.value
[[1.0, 2.0], [3.0, 4.0]]
>>> sht.range('A3').value = [5, 6]
>>> rng1.value
[[1.0, 2.0], [3.0, 4.0]]
>>> rng2.value
[[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]]
```

: The `expand` method is only available on `Range` objects as UDFs only allow to manipulate the calling cells.

14.2 Built-in Converters

xlwings offers several built-in converters that perform type conversion to **dictionaries**, **NumPy arrays**, **Pandas Series** and **DataFrames**. These build on top of the default converter, so in most cases the options described above can be used in this context, too (unless they are meaningless, for example the `ndim` in the case of a dictionary).

It is also possible to write and register custom converter for additional types, see below.

The samples below can be used with both `xlwings.Range` objects and UDFs even though only one version may be shown.

14.2.1 Dictionary converter

The dictionary converter turns two Excel columns into a dictionary. If the data is in row orientation, use `transpose`:

	A	B
1	a	1
2	b	2
3		
4	a	b
5	1	2

```
>>> sht = xw.sheets.active
>>> sht.range('A1:B2').options(dict).value
{'a': 1.0, 'b': 2.0}
>>> sht.range('A4:B5').options(dict, transpose=True).value
{'a': 1.0, 'b': 2.0}
```

Note: instead of `dict`, you can also use `OrderedDict` from `collections`.

14.2.2 Numpy array converter

options: `dtype=None, copy=True, order=None, ndim=None`

The first 3 options behave the same as when using `np.array()` directly. Also, `ndim` works the same as shown above for lists (under default converter) and hence returns either numpy scalars, 1d arrays or 2d arrays.

Example:

```
>>> import numpy as np
>>> sht = xw.Book().sheets[0]
>>> sht.range('A1').options(transpose=True).value = np.array([1, 2, 3])
>>> sht.range('A1:A3').options(np.array, ndim=2).value
array([[ 1.],
       [ 2.],
       [ 3.]])
```

14.2.3 Pandas Series converter

options: `dtype=None, copy=False, index=1, header=True`

The first 2 options behave the same as when using `pd.Series()` directly. `ndim` doesn't have an effect on Pandas series as they are always expected and returned in column orientation.

`index`: int or Boolean

When reading, it expects the number of index columns shown in Excel.

When writing, include or exclude the index by setting it to `True` or `False`.

`header`: Boolean

When reading, set it to `False` if Excel doesn't show either index or series names.

When writing, include or exclude the index and series names by setting it to `True` or `False`.

For `index` and `header`, `1` and `True` may be used interchangeably.

Example:

```
>>> sht = xw.Book().sheets[0]
>>> s = sht.range('A1').options(pd.Series, expand='table').value
```

(continues on next page)

	A	B	C	D	E
1	date	series name		01/01/01	1
2	01/01/01	1		02/01/01	2
3	02/01/01	2		03/01/01	3
4	03/01/01	3		04/01/01	4
5	04/01/01	4		05/01/01	5
6	05/01/01	5		06/01/01	6
7	06/01/01	6			

( )

```
>>> s
date
2001-01-01    1
2001-01-02    2
2001-01-03    3
2001-01-04    4
2001-01-05    5
2001-01-06    6
Name: series name, dtype: float64
>>> sht.range('D1', header=False).value = s
```

14.2.4 Pandas DataFrame converter

options: `dtype=None, copy=False, index=1, header=1`

The first 2 options behave the same as when using `pd.DataFrame()` directly. `ndim` doesn't have an effect on Pandas DataFrames as they are automatically read in with `ndim=2`.

`index`: int or Boolean

When reading, it expects the number of index columns shown in Excel.

When writing, include or exclude the index by setting it to `True` or `False`.

`header`: int or Boolean

When reading, it expects the number of column headers shown in Excel.

When writing, include or exclude the index and series names by setting it to `True` or `False`.

For `index` and `header`, `1` and `True` may be used interchangeably.

Example:

```
>>> sht = xw.Book().sheets[0]
>>> df = sht.range('A1:D5').options(pd.DataFrame, header=2).value
>>> df
    a     b
    c  d  e
ix
```

(continues on next page)

B13 | f_x {=myfunction(A1:D5)}

	A	B	C	D
1		**a**	**a**	**b**
2	**ix**	**c**	**d**	**e**
3	**10**	1	2	3
4	**20**	4	5	6
5	**30**	7	8	9
6				
7		a	a	b
8		c	d	e
9		1	2	3
10		4	5	6
11		7	8	9
12				
13		a	a	b
14		c	d	e
15		1	2	3
16		4	5	6
17		7	8	9
18				

( )

```
10  1  2  3
20  4  5  6
30  7  8  9

# Writing back using the defaults:
>>> sht.range('A1').value = df

# Writing back and changing some of the options, e.g. getting rid of the index:
>>> sht.range('B7').options(index=False).value = df
```

The same sample for **UDF** (starting in `Range('A13')` on screenshot) looks like this:

```
@xw.func
@xw.arg('x', pd.DataFrame, header=2)
@xw.ret(index=False)
def myfunction(x):
    # x is a DataFrame, do something with it
    return x
```

14.2.5 xw.Range and raw converters

Technically speaking, these are no-converters .

- If you need access to the `xlwings.Range` object directly, you can do:

```
@xw.func
@xw.arg('x', xw.Range)
```

(continues on next page)

( )

```
def myfunction(x):
   return x.formula
```

This returns x as `xlwings.Range` object, i.e. without applying any converters or options.

- The `raw` converter delivers the values unchanged from the underlying libraries (`pywin32` on Windows and `appscript` on Mac), i.e. no sanitizing/cross-platform harmonizing of values are being made. This might be useful in a few cases for efficiency reasons. E.g:

```
>>> sht.range('A1:B2').value
[[1.0, 'text'], [datetime.datetime(2016, 2, 1, 0, 0), None]]

>>> sht.range('A1:B2').options('raw').value  # or sht.range('A1:B2').raw_
↪value
((1.0, 'text'), (pywintypes.datetime(2016, 2, 1, 0, 0, tzinfo=TimeZoneInfo(
↪'GMT Standard Time', True)), None))
```

14.3 Custom Converter

Here are the steps to implement your own converter:

- Inherit from `xlwings.conversion.Converter`
- Implement both a `read_value` and `write_value` method as static- or classmethod:
  - In `read_value`, `value` is what the base converter returns: hence, if no `base` has been specified it arrives in the format of the default converter.
  - In `write_value`, `value` is the original object being written to Excel. It must be returned in the format that the base converter expects. Again, if no `base` has been specified, this is the default converter.

The `options` dictionary will contain all keyword arguments specified in the `xw.Range.options` method, e.g. when calling `xw.Range('A1').options(myoption='some value')` or as specified in the `@arg` and `@ret` decorator when using UDFs. Here is the basic structure:

```
from xlwings.conversion import Converter

class MyConverter(Converter):

    @staticmethod
    def read_value(value, options):
        myoption = options.get('myoption', default_value)
        return_value = value  # Implement your conversion here
        return return_value

    @staticmethod
```

(continues on next page)

( )

```
    def write_value(value, options):
        myoption = options.get('myoption', default_value)
        return_value = value  # Implement your conversion here
        return return_value
```

- Optional: set a `base` converter (`base` expects a class name) to build on top of an existing converter, e.g. for the built-in ones: `DictCoverter`, `NumpyArrayConverter`, `PandasDataFrameConverter`, `PandasSeriesConverter`
- Optional: register the converter: you can **(a)** register a type so that your converter becomes the default for this type during write operations and/or **(b)** you can register an alias that will allow you to explicitly call your converter by name instead of just by class name

The following examples should make it much easier to follow - it defines a DataFrame converter that extends the built-in DataFrame converter to add support for dropping nan's:

```
from xlwings.conversion import Converter, PandasDataFrameConverter

class DataFrameDropna(Converter):

    base = PandasDataFrameConverter

    @staticmethod
    def read_value(builtin_df, options):
        dropna = options.get('dropna', False)  # set default to False
        if dropna:
            converted_df = builtin_df.dropna()
        else:
            converted_df = builtin_df
        # This will arrive in Python when using the DataFrameDropna converter␣
↪for reading
        return converted_df

    @staticmethod
    def write_value(df, options):
        dropna = options.get('dropna', False)
        if dropna:
            converted_df = df.dropna()
        else:
            converted_df = df
        # This will be passed to the built-in PandasDataFrameConverter when␣
↪writing
        return converted_df
```

Now let's see how the different converters can be applied:

```
# Fire up a Workbook and create a sample DataFrame
```

(continues on next page)

( )

```
sht = xw.Book().sheets[0]
df = pd.DataFrame([[1.,10.],[2.,np.nan], [3., 30.]])
```

- Default converter for DataFrames:

```
# Write
sht.range('A1').value = df

# Read
sht.range('A1:C4').options(pd.DataFrame).value
```

- DataFrameDropna converter:

```
# Write
sht.range('A7').options(DataFrameDropna, dropna=True).value = df

# Read
sht.range('A1:C4').options(DataFrameDropna, dropna=True).value
```

- Register an alias (optional):

```
DataFrameDropna.register('df_dropna')

# Write
sht.range('A12').options('df_dropna', dropna=True).value = df

# Read
sht.range('A1:C4').options('df_dropna', dropna=True).value
```

- Register DataFrameDropna as default converter for DataFrames (optional):

```
DataFrameDropna.register(pd.DataFrame)

# Write
sht.range('A13').options(dropna=True).value = df

# Read
sht.range('A1:C4').options(pd.DataFrame, dropna=True).value
```

These samples all work the same with UDFs, e.g.:

```
@xw.func
@arg('x', DataFrameDropna, dropna=True)
@ret(DataFrameDropna, dropna=True)
def myfunction(x):
    # ...
    return x
```

: Python objects run through multiple stages of a transformation pipeline when they are being written to Excel. The same holds true in the other direction, when Excel/COM objects are being read into Python.

Pipelines are internally defined by `Accessor` classes. A Converter is just a special Accessor which converts to/from a particular type by adding an extra stage to the pipeline of the default Accessor. For example, the `PandasDataFrameConverter` defines how a list of list (as delivered by the default Accessor) should be turned into a Pandas DataFrame.

The `Converter` class provides basic scaffolding to make the task of writing a new Converter easier. If you need more control you can subclass `Accessor` directly, but this part requires more work and is currently undocumented.

CHAPTER 15

Threading and Multiprocessing

0.13.0 .

15.1 Threading

While xlwings is not technically thread safe, it's still easy to use it in threads as long as you have at least v0.13.0 and stick to a simple rule: Do not pass xlwings objects to threads. This rule isn't a requirement on macOS, but it's still recommended if you want your programs to be cross-platform.

Consider the following example that will **NOT** work:

```
import threading
from queue import Queue
import xlwings as xw

num_threads = 4


def write_to_workbook():
    while True:
        rng = q.get()
        rng.value = rng.address
        print(rng.address)
        q.task_done()


q = Queue()
```

(continues on next page)

( )

```
for i in range(num_threads):
    t = threading.Thread(target=write_to_workbook)
    t.daemon = True
    t.start()

for cell in ['A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'A9', 'A10']:
    # THIS DOESN'T WORK - passing xlwings objects to threads will fail!
    rng = xw.Book('Book1.xlsx').sheets[0].range(cell)
    q.put(rng)

q.join()
```

To make it work, you simply have to fully qualify the cell reference in the thread instead of passing a `Book` object:

```
import threading
from queue import Queue
import xlwings as xw

num_threads = 4


def write_to_workbook():
    while True:
        cell_ = q.get()
        xw.Book('Book1.xlsx').sheets[0].range(cell_).value = cell_
        print(cell_)
        q.task_done()


q = Queue()

for i in range(num_threads):
    t = threading.Thread(target=write_to_workbook)
    t.daemon = True
    t.start()

for cell in ['A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'A9', 'A10']:
    q.put(cell)

q.join()
```

15.2 Multiprocessing

: Multiprocessing is only support on Windows!

The same rules apply to multiprocessing as for threading, here's a working example:

```
from multiprocessing import Pool
import xlwings as xw


def write_to_workbook(cell):
    xw.Book('Book1.xlsx').sheets[0].range(cell).value = cell
    print(cell)


if __name__ == '__main__':
    with Pool(4) as p:
        p.map(write_to_workbook,
              ['A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'A9', 'A10'])
```

CHAPTER 16

Command Line Client

xlwings comes with a command line client that makes it easy to set up workbooks and install the add-in. On Windows, type the commands into a `Command Prompt`, on Mac, type them into a `Terminal`.

16.1 Quickstart

- `xlwings quickstart myproject`

This command is by far the fastest way to get off the ground: It creates a new folder `myproject` with an Excel workbook that already has the reference to the xlwings addin and a Python file, ready to be used right away:

```
myproject
  |--myproject.xlsm
  |--myproject.py
```

If you want to use xlwings via VBA module instead of addin, use the `--standalone` or `-s` flag:

`xlwings quickstart myproject --standalone`

16.2 Add-in

The *addin* command makes it easy on Windows to install/remove the addin. On Mac, you need to install it manually, but `xlwings addin install` will show you how to do it.

: Excel needs to be closed before installing/updating the add-in via command line. If you're still getting an error, start the Task Manager and make sure there are no `EXCEL.EXE` processes left.

- `xlwings addin install`: Copies the xlwings add-in to the XLSTART folder
- `xlwings addin update`: Replaces the current add-in with the latest one
- `xlwings addin remove`: Removes the add-in from the XLSTART folder
- `xlwings addin status`: Shows if the add-in is installed together with the installation path

After installing the add-in, it will be available as xlwings tab on the Excel Ribbon.

0.6.0 .

16.3 RunPython

Only required if you are on Mac, are using Excel 2016 and have xlwings installed via conda or as part of Anaconda. To enable the `RunPython` calls in VBA, run this one time:

`xlwings runpython install`

Alternatively, install xlwings with `pip`.

0.7.0 .

CHAPTER 17

Missing Features

If you're missing a feature in xlwings, do the following:

1) Most importantly, open an issue on GitHub. If it's something bigger or if you want support from other users, consider opening a feature request. Adding functionality should be user driven, so only if you tell us about what you're missing, it's eventually going to find its way into the library. By the way, we also appreciate pull requests!

2) Workaround: in essence, xlwings is just a smart wrapper around pywin32 on Windows and appscript on Mac. You can access the underlying objects by calling the `api` property:

```
>>> sht = xw.Book().sheets[0]
>>> sht.api
<COMObject <unknown>>  # Windows/pywin32
app(pid=2319).workbooks['Workbook1'].worksheets[1]  # Mac/appscript
```

   This works accordingly for the other objects like `sht.range('A1').api` etc.

   The underlying objects will offer you pretty much everything you can do with VBA, using the syntax of pywin32 (which pretty much feels like VBA) and appscript (which doesn't feel like VBA). But apart from looking ugly, keep in mind that **it makes your code platform specific (!)**, i.e. even if you go for option 2), you should still follow option 1) and open an issue so the feature finds it's way into the library (cross-platform and with a Pythonic syntax).

17.1 Example: Workaround to use VBA's `Range.WrapText`

```
# Windows
sht.range('A1').api.WrapText = True

# Mac
sht.range('A1').api.wrap_text.set(True)
```

CHAPTER 18

xlwings with other Office Apps

xlwings can also be used to call Python functions from VBA within Office apps other than Excel (like Outlook, Access etc.).

: New in v0.12.0 and still in a somewhat early stage that involves a bit of manual work. Currently, this functionality is only available on Windows for UDFs. The `RunPython` functionality is currently not supported.

18.1 How To

1) As usual, write your Python function and import it into Excel (see *VBA: User Defined Functions (UDFs)*).
2) Press `Alt-F11` to get into the VBA editor, then right-click on the `xlwings_udfs` VBA module and select `Export File...`. Save the `xlwings_udfs.bas` file somewhere.
3) Switch into the other Office app, e.g. Microsoft Access and click again `Alt-F11` to get into the VBA editor. Right-click on the VBA Project and `Import File...`, then select the file that you exported in the previous step. Once imported, replace the app name in the first line to the one that you are using, i.e. `Microsoft Access` or `Microsoft Outlook` etc. so that the first line then reads: `#Const App = "Microsoft Access"`
4) Now import the standalone xlwings VBA module (`xlwings.bas`). You can find it in your xlwings installation folder. To know where that is, do:

```
>>> import xlwings as xw
>>> xlwings.__path__
```

And finally do the same as in the previous step and replace the App name in the first line with the name of the corresponding app that you are using. You are now able to call the Python function from VBA.

18.2 Config

The other Office apps will use the same global config file as you are editing via the Excel ribbon add-in. When it makes sense, you'll be able to use the directory config file (e.g. you can put it next to your Access or Word file) or you can hardcode the path to the config file in the VBA standalone module, e.g. in the function `GetDirectoryConfigFilePath` (e.g. suggested when using Outlook that doesn't really have the same concept of files like the other Office apps). NOTE: For Office apps without file concept, you need to make sure that the `PYTHONPATH` points to the directory with the Python source file. For details on the different config options, see *Config*.

CHAPTER 19

Deployment

19.1 Zip files

0.15.2 .

To make it easier to distribute, you can zip up your Python code into a zip file. If you use UDFs, this will disable the automatic code reload, so this is a feature meant for distribution, not development. In practice, this means that when your code is inside a zip file, you'll have to click on re-import to get any changes.

If you name your zip file like your Excel file (but with `.zip` extension) and place it in the same folder as your Excel workbook, xlwings will automatically find it (similar to how it works with a single python file).

If you want to use a different directory, make sure to add it to the `PYTHONPATH` in your config (Ribbon or config file):

```
PYTHONPATH, "C:\path\to\myproject.zip"
```

19.2 RunFrozenPython

0.15.2 .

You can use a freezer like PyInstaller, cx_Freeze, py2exe etc. to freeze your Python module into an executable so that the recipient doesn't have to install a full Python distribution.

:

- This does not work with UDFs.

- Currently only available on Windows, but support for Mac should be easy to add.
- You need at least 0.15.2 to support arguments whereas the syntax changed in 0.15.6

Use it as follows:

```
Sub MySample()
    RunFrozenPython "C:\path\to\dist\myproject\myproject.exe", "arg1 arg2"
End Sub
```

CHAPTER 20

Extensions

It's easy to extend the xlwings add-in with own code like UDFs or RunPython macros, so that they can be deployed without end users having to import or write the functions themselves. Just add another VBA module to the xlwings addin with the respective code.

UDF extensions can be used from every workbook without having to set a reference.

20.1 In-Excel SQL

The xlwings addin comes with a built-in extension that adds in-Excel SQL syntax (sqlite dialect):

```
=sql(SQL Statement, table a, table b, ...)
```

A16 =sql(A14,A1:D11,G1:H8)

	A	B	C	D	E	F	G	H
1	id	first_name	last_name	age			id	email
2	1	Mariam	Alt	12			1	Mariam@Alt
3	2	Shenita	Truelove	55			2	Shenita@Truelove
4	3	Evelyn	Braddy	30			3	Evelyn@Braddy
5	4	Shery	Sam	35			5	Rogello@Mote
6	5	Rogello	Mote	88			6	Solomon@Okamura
7	6	Solomon	Okamura	33			8	Latashia@Alire
8	7	Jessica	Buelow	10			9	Roselee@Tarwater
9	8	Latashia	Alire	19				
10	9	Roselee	Tarwater	28				
11	10	Kiera	Saulsbury	55				
12								
13								
14	SELECT a.id, a.first_name, a.last_name, b.email FROM a INNER JOIN b ON a.id = b.id							
15								
16	id	first_name	last_name	email				
17	1	Mariam	Alt	Mariam@Alt				
18	2	Shenita	Truelove	Shenita@Truelove				
19	3	Evelyn	Braddy	Evelyn@Braddy				
20	5	Rogello	Mote	Rogello@Mote				
21	6	Solomon	Okamura	Solomon@Okamura				
22	8	Latashia	Alire	Latashia@Alire				
23	9	Roselee	Tarwater	Roselee@Tarwater				

As this extension uses UDFs, it's only available on Windows right now.

CHAPTER 21

xlwings with R and Julia

While xlwings is a pure Python package, there are cross-language packages that allow for a relative straightforward use from/with other languages. This means, however, that you'll always need to have Python with xlwings installed in addition to R or Julia. We recommend the Anaconda distribution, see also .

21.1 R

The R instructions are for Windows, but things work accordingly on Mac except that calling the R functions as User Defined Functions is not supported at the moment (but `RunPython` works, see *Call Python with RunPython*).

Setup:

- Install R and Python
- Add `R_HOME` environment variable to base directory of installation, .e.g `C:\Program Files\R\R-x.x.x`
- Add `R_USER` environment variable to user folder, e.g. `C:\Users\<user>`
- Add `C:\Program Files\R\R-x.x.x\bin` to `PATH`
- Restart Windows because of the environment variables (!)

21.1.1 Simple functions with R

Original R function that we want to access from Excel (saved in `r_file.R`):

```
myfunction <- function(x, y){
    return(x * y)
}
```

Python wrapper code:

```
import xlwings as xw
import rpy2.robjects as robjects
# you might want to use some relative path or place the file in R's current␣
↪working dir
robjects.r.source(r"C:\path\to\r_file.R")

@xw.func
def myfunction(x, y):
    myfunc = robjects.r['myfunction']
    return tuple(myfunc(x, y))
```

After importing this function (see: *VBA: User Defined Functions (UDFs)*), it will be available as UDF from Excel.

21.1.2 Array functions with R

Original R function that we want to access from Excel (saved in `r_file.R`):

```
array_function <- function(m1, m2){
  # Matrix multiplication
  return(m1 %*% m2)
}
```

Python wrapper code:

```
import xlwings as xw
import numpy as np
import rpy2.robjects as robjects
from rpy2.robjects import numpy2ri

robjects.r.source(r"C:\path\to\r_file.R")
numpy2ri.activate()

@xw.func
@xw.arg("x", np.array, ndim=2)
@xw.arg("y", np.array, ndim=2)
def array_function(x, y):
    array_func = robjects.r['array_function']
    return np.array(array_func(x, y))
```

After importing this function (see: *VBA: User Defined Functions (UDFs)*), it will be available as UDF from Excel.

21.2 Julia

Setup:

- Install Julia and Python
- Run `Pkg.add("PyCall")` from an interactive Julia interpreter

xlwings can then be called from Julia with the following syntax (the colons take care of automatic type conversion):

```
julia> using PyCall
julia> @pyimport xlwings as xw

julia> xw.Book()
PyObject <Book [Workbook1]>

julia> xw.Range("A1")[:value] = "Hello World"
julia> xw.Range("A1")[:value]
"Hello World"
```

CHAPTER 22

Troubleshooting

22.1 Issue: dll not found

Solution:

1) xlwings32-<version>.dll and xlwings64-<version>.dll are both in the same directory as your python.exe. If not, something went wrong with your installation. Reinstall it with pip or conda, see .
2) Check your Interpreter in the add-in or config sheet. If it is empty, then you need to be able to open a windows command prompt and type python to start an interactive Python session. If you get the error 'python' is not recognized as an internal or external command, operable program or batch file., then you have two options: Either add the path of where your python.exe lives to your Windows path (see https://www.computerhope.com/issues/ch000549.htm) or set the full path to your interpreter in the add-in or your config sheet, e.g. C:\Users\MyUser\anaconda\pythonw.exe

CHAPTER 23

REST API

0.13.0 .

23.1 Quickstart

xlwings offers an easy way to expose an Excel workbook via REST API both on Windows and macOS. This can be useful when you have a workbook running on a single computer and want to access it from another computer. Or you can build a Linux based web app that can interact with a legacy Excel application while you are in the progress of migrating the Excel functionality into your web app (if you need help with that, give us a shout).

You can run the REST API server from a command prompt or terminal as follows (this requires Flask>=1.0, so make sure to `pip install Flask`):

```
xlwings restapi run
```

Then perform a GET request e.g. via PowerShell on Windows or Terminal on Mac (while having an unsaved Book1 open). Note that you need to run the server and the GET request from two separate terminals (or you can use something more convenient like Postman or Insomnia for testing the API):

```
$ curl "http://127.0.0.1:5000/book/book1/sheets/0/range/A1:B2"
{
  "address": "$A$1:$B$2",
  "color": null,
  "column": 1,
  "column_width": 10.0,
  "count": 4,
```

(continues on next page)

( )


```
  "current_region": "$A$1:$B$2",
  "formula": [
    [
      "1",
      "2"
    ],
    [
      "3",
      "4"
    ]
  ],
  "formula_array": null,
  "height": 32.0,
  "last_cell": "$B$2",
  "left": 0.0,
  "name": null,
  "number_format": "General",
  "row": 1,
  "row_height": 16.0,
  "shape": [
    2,
    2
  ],
  "size": 4,
  "top": 0.0,
  "value": [
    [
      1.0,
      2.0
    ],
    [
      3.0,
      4.0
    ]
  ],
  "width": 130.0
}
```


In the command prompt where your server is running, press `Ctrl-C` to shut it down again.

The xlwings REST API is a thin wrapper around the *Python API* which makes it very easy if you have worked previously with xlwings. It also means that the REST API does require the Excel application to be up and running which makes it a great choice if the data in your Excel workbook is constantly changing as the REST API will always deliver the current state of the workbook without the need of saving it first.

: Currently, we only provide the GET methods to read the workbook. If you are also interested in the POST methods to edit the workbook, let us know via GitHub issues. Some other things will also need improvement, most notably exception handling.

23.2 Run the server

`xlwings restapi run` will run a Flask development server on http://127.0.0.1:5000. You can provide `--host` and `--port` as command line args and it also respects the Flask environment variables like `FLASK_ENV=development`.

If you want to have more control, you can run the server directly with Flask, see the Flask docs for more details:

```
set FLASK_APP=xlwings.rest.api
flask run
```

If you are on Mac, use `export FLASK_APP=xlwings.rest.api` instead of `set FLASK_APP=xlwings.rest.api`.

For production, you can use any WSGI HTTP Server like gunicorn (on Mac) or waitress (on Mac/Windows) to serve the API. For example, with gunicorn you would do: `gunicorn xlwings.rest.api:api`. Or with waitress (adjust the host accordingly if you want to make the api accessible from outside of localhost):

```
from xlwings.rest.api import api
from waitress import serve
serve(wsgiapp, host='127.0.0.1', port=5000)
```

23.3 Indexing

While the Python API offers Python's 0-based indexing (e.g. `xw.books[0]`) as well as Excel's 1-based indexing (e.g. `xw.books(1)`), the REST API only offers 0-based indexing, e.g. `/books/0`.

23.4 Range Options

The REST API accepts Range options as query parameters, see *`xlwings.Range.options()`* e.g.

`/book/book1/sheets/0/range/A1?expand=table&transpose=true`

Remember that `options` only affect the `value` property.

23.5 Endpoint overview

End-point	Corre-sponds to	Short Description
/book	*Book*	Finds your workbook across all open instances of Excel and will open it if it can't find it
/books	*Books*	Books collection of the active Excel instance
/apps	*Apps*	This allows you to specify the Excel instance you want to work with

23.6 Endpoint details

23.6.1 /book

GET /book/<fullname_or_name>

Example response:

```
{
  "app": 1104,
  "fullname": "C:\\Users\\felix\\DEV\\xlwings\\scripts\\Book1.xlsx",
  "name": "Book1.xlsx",
  "names": [
    "Sheet1!myname1",
    "myname2"
  ],
  "selection": "Sheet2!$A$1",
  "sheets": [
    "Sheet1",
    "Sheet2"
  ]
}
```

GET /book/<fullname_or_name>/names

Example response:

```
{
  "names": [
    {
      "name": "Sheet1!myname1",
      "refers_to": "=Sheet1!$B$2:$C$3"
    },
    {
      "name": "myname2",
```

(continues on next page)

( )

```
      "refers_to": "=Sheet1!$A$1"
    }
  ]
}
```

GET /book/<fullname_or_name>/names/<name>

Example response:

```
{
  "name": "myname2",
  "refers_to": "=Sheet1!$A$1"
}
```

GET /book/<fullname_or_name>/names/<name>/range

Example response:

```
{
  "address": "$A$1",
  "color": null,
  "column": 1,
  "column_width": 8.47,
  "count": 1,
  "current_region": "$A$1:$B$2",
  "formula": "=1+1.1",
  "formula_array": "=1+1,1",
  "height": 14.25,
  "last_cell": "$A$1",
  "left": 0.0,
  "name": "myname2",
  "number_format": "General",
  "row": 1,
  "row_height": 14.3,
  "shape": [
    1,
    1
  ],
  "size": 1,
  "top": 0.0,
  "value": 2.1,
  "width": 51.0
}
```

GET /book/<fullname_or_name>/sheets

Example response:

```
{
  "sheets": [
    {
      "charts": [
        "Chart 1"
      ],
      "name": "Sheet1",
      "names": [
        "Sheet1!myname1"
      ],
      "pictures": [
        "Picture 3"
      ],
      "shapes": [
        "Chart 1",
        "Picture 3"
      ],
      "used_range": "$A$1:$B$2"
    },
    {
      "charts": [],
      "name": "Sheet2",
      "names": [],
      "pictures": [],
      "shapes": [],
      "used_range": "$A$1"
    }
  ]
}
```


`GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>`

Example response:


```
{
  "charts": [
    "Chart 1"
  ],
  "name": "Sheet1",
  "names": [
    "Sheet1!myname1"
  ],
  "pictures": [
    "Picture 3"
  ],
  "shapes": [
    "Chart 1",
```


(continues on next page)

( )


```
    "Picture 3"
  ],
  "used_range": "$A$1:$B$2"
}
```


GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/charts

Example response:


```
{
  "charts": [
    {
      "chart_type": "line",
      "height": 211.0,
      "left": 0.0,
      "name": "Chart 1",
      "top": 0.0,
      "width": 355.0
    }
  ]
}
```


GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/charts/<chart_name_or_ix>

Example response:


```
{
  "chart_type": "line",
  "height": 211.0,
  "left": 0.0,
  "name": "Chart 1",
  "top": 0.0,
  "width": 355.0
}
```


GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/names

Example response:


```
{
  "names": [
    {
      "name": "Sheet1!myname1",
      "refers_to": "=Sheet1!$B$2:$C$3"
    }
  ]
}
```


GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/names/<sheet_scope_name>

Example response:

```
{
  "name": "Sheet1!myname1",
  "refers_to": "=Sheet1!$B$2:$C$3"
}
```

`GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/names/<sheet_scope_name>/range`

Example response:

```
{
  "address": "$B$2:$C$3",
  "color": null,
  "column": 2,
  "column_width": 8.47,
  "count": 4,
  "current_region": "$A$1:$B$2",
  "formula": [
    [
      "",
      ""
    ],
    [
      "",
      ""
    ]
  ],
  "formula_array": "",
  "height": 28.5,
  "last_cell": "$C$3",
  "left": 51.0,
  "name": "Sheet1!myname1",
  "number_format": "General",
  "row": 2,
  "row_height": 14.3,
  "shape": [
    2,
    2
  ],
  "size": 4,
  "top": 14.25,
  "value": [
    [
      null,
      null
    ],
    [
```

(continues on next page)

( )

```
      null,
      null
    ]
  ],
  "width": 102.0
}
```

GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/pictures

Example response:

```
{
  "pictures": [
    {
      "height": 100.0,
      "left": 0.0,
      "name": "Picture 3",
      "top": 0.0,
      "width": 100.0
    }
  ]
}
```

GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/pictures/<picture_name_or_ix>

Example response:

```
{
  "height": 100.0,
  "left": 0.0,
  "name": "Picture 3",
  "top": 0.0,
  "width": 100.0
}
```

GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/range

Example response:

```
{
  "address": "$A$1:$B$2",
  "color": null,
  "column": 1,
  "column_width": 8.47,
  "count": 4,
  "current_region": "$A$1:$B$2",
  "formula": [
    [
```

(continues on next page)

(   )

```
      "=1+1.1",
      "a string"
    ],
    [
      "43395.0064583333",
      ""
    ]
  ],
  "formula_array": null,
  "height": 28.5,
  "last_cell": "$B$2",
  "left": 0.0,
  "name": null,
  "number_format": null,
  "row": 1,
  "row_height": 14.3,
  "shape": [
    2,
    2
  ],
  "size": 4,
  "top": 0.0,
  "value": [
    [
      2.1,
      "a string"
    ],
    [
      "Mon, 22 Oct 2018 00:09:18 GMT",
      null
    ]
  ],
  "width": 102.0
}
```

GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/range/<address>

Example response:

```
{
  "address": "$A$1:$B$2",
  "color": null,
  "column": 1,
  "column_width": 8.47,
  "count": 4,
  "current_region": "$A$1:$B$2",
```

(continues on next page)

( )


```
  "formula": [
    [
      "=1+1.1",
      "a string"
    ],
    [
      "43395.0064583333",
      ""
    ]
  ],
  "formula_array": null,
  "height": 28.5,
  "last_cell": "$B$2",
  "left": 0.0,
  "name": null,
  "number_format": null,
  "row": 1,
  "row_height": 14.3,
  "shape": [
    2,
    2
  ],
  "size": 4,
  "top": 0.0,
  "value": [
    [
      2.1,
      "a string"
    ],
    [
      "Mon, 22 Oct 2018 00:09:18 GMT",
      null
    ]
  ],
  "width": 102.0
}
```


`GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/shapes`

Example response:


```
{
  "shapes": [
    {
      "height": 211.0,
      "left": 0.0,
```


(continues on next page)

( )


```
      "name": "Chart 1",
      "top": 0.0,
      "type": "chart",
      "width": 355.0
    },
    {
      "height": 100.0,
      "left": 0.0,
      "name": "Picture 3",
      "top": 0.0,
      "type": "picture",
      "width": 100.0
    }
  ]
}
```


`GET /book/<fullname_or_name>/sheets/<sheet_name_or_ix>/shapes/<shape_name_or_ix>`

Example response:


```
{
  "height": 211.0,
  "left": 0.0,
  "name": "Chart 1",
  "top": 0.0,
  "type": "chart",
  "width": 355.0
}
```


23.6.2 /books

`GET /books`

Example response:


```
{
  "books": [
    {
      "app": 1104,
      "fullname": "Book1",
      "name": "Book1",
      "names": [],
      "selection": "Sheet2!$A$1",
      "sheets": [
        "Sheet1"
      ]
```


(continues on next page)

( )

```
    },
    {
      "app": 1104,
      "fullname": "C:\\Users\\felix\\DEV\\xlwings\\scripts\\Book1.xlsx",
      "name": "Book1.xlsx",
      "names": [
        "Sheet1!myname1",
        "myname2"
      ],
      "selection": "Sheet2!$A$1",
      "sheets": [
        "Sheet1",
        "Sheet2"
      ]
    },
    {
      "app": 1104,
      "fullname": "Book4",
      "name": "Book4",
      "names": [],
      "selection": "Sheet2!$A$1",
      "sheets": [
        "Sheet1"
      ]
    }
  ]
}
```

`GET /books/<book_name_or_ix>`

Example response:

```
{
  "app": 1104,
  "fullname": "C:\\Users\\felix\\DEV\\xlwings\\scripts\\Book1.xlsx",
  "name": "Book1.xlsx",
  "names": [
    "Sheet1!myname1",
    "myname2"
  ],
  "selection": "Sheet2!$A$1",
  "sheets": [
    "Sheet1",
    "Sheet2"
  ]
}
```

`GET /books/<book_name_or_ix>/names`

Example response:


```
{
  "names": [
    {
      "name": "Sheet1!myname1",
      "refers_to": "=Sheet1!$B$2:$C$3"
    },
    {
      "name": "myname2",
      "refers_to": "=Sheet1!$A$1"
    }
  ]
}
```


`GET /books/<book_name_or_ix>/names/<name>`

Example response:


```
{
  "name": "myname2",
  "refers_to": "=Sheet1!$A$1"
}
```


`GET /books/<book_name_or_ix>/names/<name>/range`

Example response:


```
{
  "address": "$A$1",
  "color": null,
  "column": 1,
  "column_width": 8.47,
  "count": 1,
  "current_region": "$A$1:$B$2",
  "formula": "=1+1.1",
  "formula_array": "=1+1,1",
  "height": 14.25,
  "last_cell": "$A$1",
  "left": 0.0,
  "name": "myname2",
  "number_format": "General",
  "row": 1,
  "row_height": 14.3,
  "shape": [
    1,
    1
```


(continues on next page)

( )


```
  ],
  "size": 1,
  "top": 0.0,
  "value": 2.1,
  "width": 51.0
}
```


`GET /books/<book_name_or_ix>/sheets`

Example response:


```
{
  "sheets": [
    {
      "charts": [
        "Chart 1"
      ],
      "name": "Sheet1",
      "names": [
        "Sheet1!myname1"
      ],
      "pictures": [
        "Picture 3"
      ],
      "shapes": [
        "Chart 1",
        "Picture 3"
      ],
      "used_range": "$A$1:$B$2"
    },
    {
      "charts": [],
      "name": "Sheet2",
      "names": [],
      "pictures": [],
      "shapes": [],
      "used_range": "$A$1"
    }
  ]
}
```


`GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>`

Example response:


```
{
  "charts": [
```


(continues on next page)

( )

```
    "Chart 1"
  ],
  "name": "Sheet1",
  "names": [
    "Sheet1!myname1"
  ],
  "pictures": [
    "Picture 3"
  ],
  "shapes": [
    "Chart 1",
    "Picture 3"
  ],
  "used_range": "$A$1:$B$2"
}
```

`GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/charts`

Example response:

```
{
  "charts": [
    {
      "chart_type": "line",
      "height": 211.0,
      "left": 0.0,
      "name": "Chart 1",
      "top": 0.0,
      "width": 355.0
    }
  ]
}
```

`GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/charts/<chart_name_or_ix>`

Example response:

```
{
  "chart_type": "line",
  "height": 211.0,
  "left": 0.0,
  "name": "Chart 1",
  "top": 0.0,
  "width": 355.0
}
```

`GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/names`

Example response:


```
{
  "names": [
    {
      "name": "Sheet1!myname1",
      "refers_to": "=Sheet1!$B$2:$C$3"
    }
  ]
}
```


`GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/names/<sheet_scope_name>`

Example response:


```
{
  "name": "Sheet1!myname1",
  "refers_to": "=Sheet1!$B$2:$C$3"
}
```


`GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/names/<sheet_scope_name>/range`

Example response:


```
{
  "address": "$B$2:$C$3",
  "color": null,
  "column": 2,
  "column_width": 8.47,
  "count": 4,
  "current_region": "$A$1:$B$2",
  "formula": [
    [
      "",
      ""
    ],
    [
      "",
      ""
    ]
  ],
  "formula_array": "",
  "height": 28.5,
  "last_cell": "$C$3",
  "left": 51.0,
  "name": "Sheet1!myname1",
  "number_format": "General",
  "row": 2,
```


(continues on next page)

( )

```
  "row_height": 14.3,
  "shape": [
    2,
    2
  ],
  "size": 4,
  "top": 14.25,
  "value": [
    [
      null,
      null
    ],
    [
      null,
      null
    ]
  ],
  "width": 102.0
}
```

`GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/pictures`

Example response:

```
{
  "pictures": [
    {
      "height": 100.0,
      "left": 0.0,
      "name": "Picture 3",
      "top": 0.0,
      "width": 100.0
    }
  ]
}
```

`GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/pictures/<picture_name_or_ix>`

Example response:

```
{
  "height": 100.0,
  "left": 0.0,
  "name": "Picture 3",
  "top": 0.0,
  "width": 100.0
}
```

```
GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/range
```

Example response:


```
{
  "address": "$A$1:$B$2",
  "color": null,
  "column": 1,
  "column_width": 8.47,
  "count": 4,
  "current_region": "$A$1:$B$2",
  "formula": [
    [
      "=1+1.1",
      "a string"
    ],
    [
      "43395.0064583333",
      ""
    ]
  ],
  "formula_array": null,
  "height": 28.5,
  "last_cell": "$B$2",
  "left": 0.0,
  "name": null,
  "number_format": null,
  "row": 1,
  "row_height": 14.3,
  "shape": [
    2,
    2
  ],
  "size": 4,
  "top": 0.0,
  "value": [
    [
      2.1,
      "a string"
    ],
    [
      "Mon, 22 Oct 2018 00:09:18 GMT",
      null
    ]
  ],
  "width": 102.0
}
```

```
GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/range/<address>
```

Example response:


```
{
  "address": "$A$1:$B$2",
  "color": null,
  "column": 1,
  "column_width": 8.47,
  "count": 4,
  "current_region": "$A$1:$B$2",
  "formula": [
    [
      "=1+1.1",
      "a string"
    ],
    [
      "43395.0064583333",
      ""
    ]
  ],
  "formula_array": null,
  "height": 28.5,
  "last_cell": "$B$2",
  "left": 0.0,
  "name": null,
  "number_format": null,
  "row": 1,
  "row_height": 14.3,
  "shape": [
    2,
    2
  ],
  "size": 4,
  "top": 0.0,
  "value": [
    [
      2.1,
      "a string"
    ],
    [
      "Mon, 22 Oct 2018 00:09:18 GMT",
      null
    ]
  ],
  "width": 102.0
}
```

GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/shapes

Example response:

```
{
  "shapes": [
    {
      "height": 211.0,
      "left": 0.0,
      "name": "Chart 1",
      "top": 0.0,
      "type": "chart",
      "width": 355.0
    },
    {
      "height": 100.0,
      "left": 0.0,
      "name": "Picture 3",
      "top": 0.0,
      "type": "picture",
      "width": 100.0
    }
  ]
}
```

GET /books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/shapes/<shape_name_or_ix>

Example response:

```
{
  "height": 211.0,
  "left": 0.0,
  "name": "Chart 1",
  "top": 0.0,
  "type": "chart",
  "width": 355.0
}
```

23.6.3 /apps

GET /apps

Example response:

```
{
  "apps": [
    {
      "books": [
```

(continues on next page)

( )

```
        "Book1",
        "C:\\Users\\felix\\DEV\\xlwings\\scripts\\Book1.xlsx",
        "Book4"
      ],
      "calculation": "automatic",
      "display_alerts": true,
      "pid": 1104,
      "screen_updating": true,
      "selection": "[Book1.xlsx]Sheet2!$A$1",
      "version": "16.0",
      "visible": true
    },
    {
      "books": [
        "Book2",
        "Book5"
      ],
      "calculation": "automatic",
      "display_alerts": true,
      "pid": 7920,
      "screen_updating": true,
      "selection": "[Book5]Sheet2!$A$1",
      "version": "16.0",
      "visible": true
    }
  ]
}
```

`GET /apps/<pid>`

Example response:

```
{
  "books": [
    "Book1",
    "C:\\Users\\felix\\DEV\\xlwings\\scripts\\Book1.xlsx",
    "Book4"
  ],
  "calculation": "automatic",
  "display_alerts": true,
  "pid": 1104,
  "screen_updating": true,
  "selection": "[Book1.xlsx]Sheet2!$A$1",
  "version": "16.0",
  "visible": true
}
```

`GET /apps/<pid>/books`

Example response:


```
{
  "books": [
    {
      "app": 1104,
      "fullname": "Book1",
      "name": "Book1",
      "names": [],
      "selection": "Sheet2!$A$1",
      "sheets": [
        "Sheet1"
      ]
    },
    {
      "app": 1104,
      "fullname": "C:\\Users\\felix\\DEV\\xlwings\\scripts\\Book1.xlsx",
      "name": "Book1.xlsx",
      "names": [
        "Sheet1!myname1",
        "myname2"
      ],
      "selection": "Sheet2!$A$1",
      "sheets": [
        "Sheet1",
        "Sheet2"
      ]
    },
    {
      "app": 1104,
      "fullname": "Book4",
      "name": "Book4",
      "names": [],
      "selection": "Sheet2!$A$1",
      "sheets": [
        "Sheet1"
      ]
    }
  ]
}
```


`GET /apps/<pid>/books/<book_name_or_ix>`

Example response:


```
{
```


(continues on next page)

( )

```
  "app": 1104,
  "fullname": "C:\\Users\\felix\\DEV\\xlwings\\scripts\\Book1.xlsx",
  "name": "Book1.xlsx",
  "names": [
    "Sheet1!myname1",
    "myname2"
  ],
  "selection": "Sheet2!$A$1",
  "sheets": [
    "Sheet1",
    "Sheet2"
  ]
}
```

`GET /apps/<pid>/books/<book_name_or_ix>/names`

Example response:

```
{
  "names": [
    {
      "name": "Sheet1!myname1",
      "refers_to": "=Sheet1!$B$2:$C$3"
    },
    {
      "name": "myname2",
      "refers_to": "=Sheet1!$A$1"
    }
  ]
}
```

`GET /apps/<pid>/books/<book_name_or_ix>/names/<name>`

Example response:

```
{
  "name": "myname2",
  "refers_to": "=Sheet1!$A$1"
}
```

`GET /apps/<pid>/books/<book_name_or_ix>/names/<name>/range`

Example response:

```
{
  "address": "$A$1",
  "color": null,
  "column": 1,
```

(continues on next page)

(continued from previous page)

```
  "column_width": 8.47,
  "count": 1,
  "current_region": "$A$1:$B$2",
  "formula": "=1+1.1",
  "formula_array": "=1+1,1",
  "height": 14.25,
  "last_cell": "$A$1",
  "left": 0.0,
  "name": "myname2",
  "number_format": "General",
  "row": 1,
  "row_height": 14.3,
  "shape": [
    1,
    1
  ],
  "size": 1,
  "top": 0.0,
  "value": 2.1,
  "width": 51.0
}
```

`GET /apps/<pid>/books/<book_name_or_ix>/sheets`

Example response:

```
{
  "sheets": [
    {
      "charts": [
        "Chart 1"
      ],
      "name": "Sheet1",
      "names": [
        "Sheet1!myname1"
      ],
      "pictures": [
        "Picture 3"
      ],
      "shapes": [
        "Chart 1",
        "Picture 3"
      ],
      "used_range": "$A$1:$B$2"
    },
    {
```

(continues on next page)

( )


```
      "charts": [],
      "name": "Sheet2",
      "names": [],
      "pictures": [],
      "shapes": [],
      "used_range": "$A$1"
    }
  ]
}
```


`GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>`

Example response:


```
{
  "charts": [
    "Chart 1"
  ],
  "name": "Sheet1",
  "names": [
    "Sheet1!myname1"
  ],
  "pictures": [
    "Picture 3"
  ],
  "shapes": [
    "Chart 1",
    "Picture 3"
  ],
  "used_range": "$A$1:$B$2"
}
```


`GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/charts`

Example response:


```
{
  "charts": [
    {
      "chart_type": "line",
      "height": 211.0,
      "left": 0.0,
      "name": "Chart 1",
      "top": 0.0,
      "width": 355.0
    }
  ]
```


(continues on next page)

```
}
```

GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/charts/<chart_name_or_ix>

Example response:

```
{
  "chart_type": "line",
  "height": 211.0,
  "left": 0.0,
  "name": "Chart 1",
  "top": 0.0,
  "width": 355.0
}
```

GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/names

Example response:

```
{
  "names": [
    {
      "name": "Sheet1!myname1",
      "refers_to": "=Sheet1!$B$2:$C$3"
    }
  ]
}
```

GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/names/<sheet_scope_name>

Example response:

```
{
  "name": "Sheet1!myname1",
  "refers_to": "=Sheet1!$B$2:$C$3"
}
```

GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/names/<sheet_scope_name>/rang

Example response:

```
{
  "address": "$B$2:$C$3",
  "color": null,
  "column": 2,
  "column_width": 8.47,
  "count": 4,
  "current_region": "$A$1:$B$2",
```

(continues on next page)

( )

```
  "formula": [
    [
      "",
      ""
    ],
    [
      "",
      ""
    ]
  ],
  "formula_array": "",
  "height": 28.5,
  "last_cell": "$C$3",
  "left": 51.0,
  "name": "Sheet1!myname1",
  "number_format": "General",
  "row": 2,
  "row_height": 14.3,
  "shape": [
    2,
    2
  ],
  "size": 4,
  "top": 14.25,
  "value": [
    [
      null,
      null
    ],
    [
      null,
      null
    ]
  ],
  "width": 102.0
}
```

GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/pictures

Example response:

```
{
  "pictures": [
    {
      "height": 100.0,
      "left": 0.0,
```

(continues on next page)

( )


```
      "name": "Picture 3",
      "top": 0.0,
      "width": 100.0
    }
  ]
}
```


`GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/pictures/<picture_name_or_ix>`

Example response:


```
{
  "height": 100.0,
  "left": 0.0,
  "name": "Picture 3",
  "top": 0.0,
  "width": 100.0
}
```


`GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/range`

Example response:


```
{
  "address": "$A$1:$B$2",
  "color": null,
  "column": 1,
  "column_width": 8.47,
  "count": 4,
  "current_region": "$A$1:$B$2",
  "formula": [
    [
      "=1+1.1",
      "a string"
    ],
    [
      "43395.0064583333",
      ""
    ]
  ],
  "formula_array": null,
  "height": 28.5,
  "last_cell": "$B$2",
  "left": 0.0,
  "name": null,
  "number_format": null,
  "row": 1,
```


(continues on next page)

( )

```
  "row_height": 14.3,
  "shape": [
    2,
    2
  ],
  "size": 4,
  "top": 0.0,
  "value": [
    [
      2.1,
      "a string"
    ],
    [
      "Mon, 22 Oct 2018 00:09:18 GMT",
      null
    ]
  ],
  "width": 102.0
}
```

```
GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/range/<address>
```

Example response:

```
{
  "address": "$A$1:$B$2",
  "color": null,
  "column": 1,
  "column_width": 8.47,
  "count": 4,
  "current_region": "$A$1:$B$2",
  "formula": [
    [
      "=1+1.1",
      "a string"
    ],
    [
      "43395.0064583333",
      ""
    ]
  ],
  "formula_array": null,
  "height": 28.5,
  "last_cell": "$B$2",
  "left": 0.0,
  "name": null,
```

(continues on next page)

( )


```
  "number_format": null,
  "row": 1,
  "row_height": 14.3,
  "shape": [
    2,
    2
  ],
  "size": 4,
  "top": 0.0,
  "value": [
    [
      2.1,
      "a string"
    ],
    [
      "Mon, 22 Oct 2018 00:09:18 GMT",
      null
    ]
  ],
  "width": 102.0
}
```


GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/shapes

Example response:


```
{
  "shapes": [
    {
      "height": 211.0,
      "left": 0.0,
      "name": "Chart 1",
      "top": 0.0,
      "type": "chart",
      "width": 355.0
    },
    {
      "height": 100.0,
      "left": 0.0,
      "name": "Picture 3",
      "top": 0.0,
      "type": "picture",
      "width": 100.0
    }
  ]
}
```

```
GET /apps/<pid>/books/<book_name_or_ix>/sheets/<sheet_name_or_ix>/shapes/<shape_name_or_ix>
```

Example response:


```
{
  "height": 211.0,
  "left": 0.0,
  "name": "Chart 1",
  "top": 0.0,
  "type": "chart",
  "width": 355.0
}
```

CHAPTER 24

Python API

24.1 Top-level functions

`xlwings.view(obj, sheet=None)`

Opens a new workbook and displays an object on its first sheet by default. If you provide a sheet object, it will clear the sheet before displaying the object on the existing sheet.

- **obj** (*any type with built-in converter*) – the object to display, e.g. numbers, strings, lists, numpy arrays, pandas dataframes
- **sheet** (*Sheet, default None*) – Sheet object. If none provided, the first sheet of a new workbook is used.

Examples

```
>>> import xlwings as xw
>>> import pandas as pd
>>> import numpy as np
>>> df = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c', 'd'])
>>> xw.view(df)
```

0.7.1 .

24.2 Object model

24.2.1 Apps

class xlwings.main.**Apps**(*impl*)

A collection of all **app** objects:

```
>>> import xlwings as xw
>>> xw.apps
Apps([<Excel App 1668>, <Excel App 1644>])
```

active

Returns the active app.

0.9.0 .

add()

Creates a new App. The new App becomes the active one. Returns an App object.

count

Returns the number of apps.

0.9.0 .

keys()

Provides the PIDs of the Excel instances that act as keys in the Apps collection.

0.13.0 .

24.2.2 App

class xlwings.**App**(*visible=None, spec=None, add_book=True, impl=None*)

An app corresponds to an Excel instance. New Excel instances can be fired up like so:

```
>>> import xlwings as xw
>>> app1 = xw.App()
>>> app2 = xw.App()
```

An app object is a member of the *apps* collection:

```
>>> xw.apps
Apps([<Excel App 1668>, <Excel App 1644>])
>>> xw.apps[1668]   # get the available PIDs via xw.apps.keys()
<Excel App 1668>
>>> xw.apps.active
<Excel App 1668>
```

- **visible** (*bool, default None*) – Returns or sets a boolean value that determines whether the app is visible. The default leaves the state unchanged or sets visible=True if the object doesn't exist yet.
- **spec** (*str, default None*) – Mac-only, use the full path to the Excel application, e.g. `/Applications/Microsoft Office 2011/Microsoft Excel` or `/Applications/Microsoft Excel`

  On Windows, if you want to change the version of Excel that xlwings talks to, go to `Control Panel > Programs and Features` and `Repair` the Office version that you want as default.

: On Mac, while xlwings allows you to run multiple instances of Excel, it's a feature that is not officially supported by Excel for Mac: Unlike on Windows, Excel will not ask you to open a read-only version of a file if it is already open in another instance. This means that you need to watch out yourself so that the same file is not being overwritten from different instances.

---

activate(*steal_focus=False*)
Activates the Excel app.

steal_focus (*bool, default False*) – If True, make frontmost application and hand over focus from Python to Excel.

0.9.0 .

api
Returns the native object (`pywin32` or `appscript` obj) of the engine being used.

0.9.0 .

books
A collection of all Book objects that are currently open.

0.9.0 .

calculate()
Calculates all open books.

0.3.6 .

calculation
Returns or sets a calculation value that represents the calculation mode. Modes: `'manual'`, `'automatic'`, `'semiautomatic'`

Examples

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> wb.app.calculation = 'manual'
```

0.9.0 .

display_alerts

The default value is True. Set this property to False to suppress prompts and alert messages while code is running; when a message requires a response, Excel chooses the default response.

0.9.0 .

hwnd

Returns the Window handle (Windows-only).

0.9.0 .

kill()

Forces the Excel app to quit by killing its process.

0.9.0 .

macro(*name*)

Runs a Sub or Function in Excel VBA that are not part of a specific workbook but e.g. are part of an add-in.

name (Name of Sub or Function with or without module name, e.g. `'Module1.MyMacro'` or `'MyMacro'`) –

Examples

This VBA function:

```
Function MySum(x, y)
    MySum = x + y
End Function
```

can be accessed like this:

```
>>> import xlwings as xw
>>> app = xw.App()
>>> my_sum = app.macro('MySum')
>>> my_sum(1, 2)
3
```

See also: *Book.macro()*

0.9.0 .

pid

Returns the PID of the app.

0.9.0 .

quit()

Quits the application without saving any workbooks.

0.3.3 .

range(*cell1*, *cell2=None*)
Range object from the active sheet of the active book, see *Range()*.

0.9.0 .

screen_updating
Turn screen updating off to speed up your script. You won't be able to see what the script is doing, but it will run faster. Remember to set the screen_updating property back to True when your script ends.

0.3.3 .

selection
Returns the selected cells as Range.

0.9.0 .

version
Returns the Excel version number object.

Examples

```
>>> import xlwings as xw
>>> xw.App().version
VersionNumber('15.24')
>>> xw.apps[10559].version.major
15
```

0.9.0 .

visible
Gets or sets the visibility of Excel to `True` or `False`.

0.3.3 .

24.2.3 Books

class `xlwings.main.Books`(*impl*)
A collection of all `book` objects:

```
>>> import xlwings as xw
>>> xw.books  # active app
Books([<Book [Book1]>, <Book [Book2]>])
>>> xw.apps[10559].books  # specific app, get the PIDs via xw.apps.keys()
Books([<Book [Book1]>, <Book [Book2]>])
```

0.9.0 .

`active`
Returns the active Book.

`add()`
Creates a new Book. The new Book becomes the active Book. Returns a Book object.

`open`(*fullname*, *update_links=None*, *read_only=None*, *format=None*, *password=None*, *write_res_password=None*, *ignore_read_only_recommended=None*, *origin=None*, *delimiter=None*, *editable=None*, *notify=None*, *converter=None*, *add_to_mru=None*, *local=None*, *corrupt_load=None*)
Opens a Book if it is not open yet and returns it. If it is already open, it doesn't raise an exception but simply returns the Book object.

- **fullname** (*str or path-like object*) – filename or fully qualified filename, e.g. `r'C:\path\to\file.xlsx'` or `'file.xlsm'`. Without a full path, it looks for the file in the current working directory.
- **Parameters** (*Other*) – see: *xlwings.Book()*

Book

Book that has been opened.

24.2.4 Book

`class xlwings.Book`(*fullname=None*, *update_links=None*, *read_only=None*, *format=None*, *password=None*, *write_res_password=None*, *ignore_read_only_recommended=None*, *origin=None*, *delimiter=None*, *editable=None*, *notify=None*, *converter=None*, *add_to_mru=None*, *local=None*, *corrupt_load=None*, *impl=None*)
A book object is a member of the `books` collection:

```
>>> import xlwings as xw
>>> xw.books[0]
<Book [Book1]>
```

The easiest way to connect to a book is offered by `xw.Book`: it looks for the book in all app instances and returns an error, should the same book be open in multiple instances. To connect to a book in the active app instance, use `xw.books` and to refer to a specific app, use:

```
>>> app = xw.App()  # or something like xw.apps[10559] for existing apps,␣
↪get the PIDs via xw.apps.keys()
>>> app.books['Book1']
```

	xw.Book	xw.books
New book	`xw.Book()`	`xw.books.add()`
Unsaved book	`xw.Book('Book1')`	`xw.books['Book1']`
Book by (full)name	`xw.Book(r'C:/path/to/file.xlsx')`	`xw.books.open(r'C:/path/to/file.xlsx')`

- **fullname** (*str or path-like object, default None*) – Full path or name (incl. xlsx, xlsm etc.) of existing workbook or name of an unsaved workbook. Without a full path, it looks for the file in the current working directory.
- **update_links** (*bool, default None*) – If this argument is omitted, the user is prompted to specify how links will be updated
- **read_only** (*bool, default False*) – True to open workbook in read-only mode
- **format** (*str*) – If opening a text file, this specifies the delimiter character
- **password** (*str*) – Password to open a protected workbook
- **write_res_password** (*str*) – Password to write to a write-reserved workbook
- **ignore_read_only_recommended** (*bool, default False*) – Set to `True` to mute the read-only recommended message
- **origin** (*int*) – For text files only. Specifies where it originated. Use XlPlatform constants.
- **delimiter** (*str*) – If format argument is 6, this specifies the delimiter.
- **editable** (*bool, default False*) – This option is only for legacy Microsoft Excel 4.0 addins.
- **notify** (*bool, default False*) – Notify the user when a file becomes available If the file cannot be opened in read/write mode.
- **converter** (*int*) – The index of the first file converter to try when opening the file.
- **add_to_mru** (*bool, default False*) – Add this workbook to the list of recently added workbooks.
- **local** (*bool, default False*) – If `True`, saves files against the language of Excel, otherwise against the language of VBA. Not supported on macOS.
- **corrupt_load** (*int, default xlNormalLoad*) – Can be one of xlNormalLoad, xlRepairFile or xlExtractData. Not supported on macOS.

activate(*steal_focus=False*)

Activates the book.

steal_focus (*bool, default False*) – If True, make frontmost window and hand over focus from Python to Excel.

api

Returns the native object (`pywin32` or `appscript` obj) of the engine being used.

0.9.0 .

app

Returns an app object that represents the creator of the book.

0.9.0 .

classmethod caller()

References the calling book when the Python function is called from Excel via `RunPython`. Pack it into the function being called from Excel, e.g.:

To be able to easily invoke such code from Python for debugging, use `xw.Book.set_mock_caller()`.

0.3.0 .

close()

Closes the book without saving it.

0.1.1 .

fullname

Returns the name of the object, including its path on disk, as a string. Read-only String.

macro(*name*)

Runs a Sub or Function in Excel VBA.

name (Name of Sub or Function with or without module name, e.g. `'Module1.MyMacro'` or `'MyMacro'`) –

Examples

This VBA function:

```
Function MySum(x, y)
    MySum = x + y
End Function
```

can be accessed like this:

```
>>> import xlwings as xw
>>> wb = xw.books.active
>>> my_sum = wb.macro('MySum')
>>> my_sum(1, 2)
3
```

See also: *App.macro()*

0.7.1 .

name

Returns the name of the book as str.

names

Returns a names collection that represents all the names in the specified book (including all sheet-specific names).

0.9.0 .

save(*path=None*)

Saves the Workbook. If a path is being provided, this works like SaveAs() in Excel. If no path is specified and if the file hasn't been saved previously, it's being saved in the current working directory with the current filename. Existing files are overwritten without prompting.

path (*str or path-like object, default None*) – Full path to the workbook

Example

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> wb.save()
>>> wb.save(r'C:\path\to\new_file_name.xlsx')
```

0.3.1 .

selection

Returns the selected cells as Range.

0.9.0 .

set_mock_caller()

Sets the Excel file which is used to mock `xw.Book.caller()` when the code is called from Python and not from Excel via `RunPython`.

Examples

```
# This code runs unchanged from Excel via RunPython and from Python␣
↪directly
import os
import xlwings as xw

def my_macro():
    sht = xw.Book.caller().sheets[0]
    sht.range('A1').value = 'Hello xlwings!'

if __name__ == '__main__':
    xw.Book('file.xlsm').set_mock_caller()
    my_macro()
```

0.3.1 .

sheets

Returns a sheets collection that represents all the sheets in the book.

0.9.0 .

24.2.5 Sheets

class xlwings.main.Sheets(*impl*)

A collection of all **sheet** objects:

```
>>> import xlwings as xw
>>> xw.sheets  # active book
Sheets([<Sheet [Book1]Sheet1>, <Sheet [Book1]Sheet2>])
>>> xw.Book('Book1').sheets  # specific book
Sheets([<Sheet [Book1]Sheet1>, <Sheet [Book1]Sheet2>])
```

0.9.0 .

active

Returns the active Sheet.

add(*name=None*, *before=None*, *after=None*)

Creates a new Sheet and makes it the active sheet.

- **name** (*str, default None*) – Name of the new sheet. If None, will default to Excel's default name.
- **before** (*Sheet, default None*) – An object that specifies the sheet before which the new sheet is added.
- **after** (*Sheet, default None*) – An object that specifies the sheet after which the new sheet is added.

24.2.6 Sheet

class xlwings.Sheet(*sheet=None*, *impl=None*)

A sheet object is a member of the *sheets* collection:

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> wb.sheets[0]
<Sheet [Book1]Sheet1>
>>> wb.sheets['Sheet1']
<Sheet [Book1]Sheet1>
>>> wb.sheets.add()
<Sheet [Book1]Sheet2>
```

0.9.0 .

activate()

Activates the Sheet and returns it.

api

Returns the native object (pywin32 or appscript obj) of the engine being used.

0.9.0 .

autofit(*axis=None*)

Autofits the width of either columns, rows or both on a whole Sheet.

axis (*string, default None*) –

- To autofit rows, use one of the following: rows or r
- To autofit columns, use one of the following: columns or c
- To autofit rows and columns, provide no arguments

Examples

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> wb.sheets['Sheet1'].autofit('c')
>>> wb.sheets['Sheet1'].autofit('r')
>>> wb.sheets['Sheet1'].autofit()
```

0.2.3 .

book

Returns the Book of the specified Sheet. Read-only.

cells

Returns a Range object that represents all the cells on the Sheet (not just the cells that are currently in use).

0.9.0 .

charts

See *Charts*

0.9.0 .

clear()

Clears the content and formatting of the whole sheet.

clear_contents()

Clears the content of the whole sheet but leaves the formatting.

delete()

Deletes the Sheet.

index

Returns the index of the Sheet (1-based as in Excel).

name

Gets or sets the name of the Sheet.

names
Returns a names collection that represents all the sheet-specific names (names defined with the SheetName! prefix).

0.9.0 .

pictures
See *Pictures*

0.9.0 .

range(*cell1*, *cell2=None*)
Returns a Range object from the active sheet of the active book, see *Range()*.

0.9.0 .

select()
Selects the Sheet. Select only works on the active book.

0.9.0 .

shapes
See *Shapes*

0.9.0 .

used_range
Used Range of Sheet.

xw.Range

0.13.0 .

24.2.7 Range

class xlwings.**Range**(*cell1=None*, *cell2=None*, ***options*)
Returns a Range object that represents a cell or a range of cells.

- **cell1** (*str or tuple or* Range) – Name of the range in the upper-left corner in A1 notation or as index-tuple or as name or as xw.Range object. It can also specify a range using the range operator (a colon), .e.g. A1:B2
- **cell2** (*str or tuple or* Range*, default None*) – Name of the range in the lower-right corner in A1 notation or as index-tuple or as name or as xw.Range object.

Examples

Active Sheet:

```
import xlwings as xw
xw.Range('A1')
xw.Range('A1:C3')
xw.Range((1,1))
xw.Range((1,1), (3,3))
xw.Range('NamedRange')
xw.Range(xw.Range('A1'), xw.Range('B2'))
```

Specific Sheet:

```
xw.books['MyBook.xlsx'].sheets[0].range('A1')
```

add_hyperlink(*address*, *text_to_display=None*, *screen_tip=None*)

Adds a hyperlink to the specified Range (single Cell)

- **address** (*str*) – The address of the hyperlink.
- **text_to_display** (*str, default None*) – The text to be displayed for the hyperlink. Defaults to the hyperlink address.
- **screen_tip** (*str, default None*) – The screen tip to be displayed when the mouse pointer is paused over the hyperlink. Default is set to <address> - Click once to follow. Click and hold to select this cell.

0.3.0 .

address

Returns a string value that represents the range reference. Use `get_address()` to be able to provide paramaters.

0.9.0 .

api

Returns the native object (`pywin32` or `appscript` obj) of the engine being used.

0.9.0 .

autofit()

Autofits the width and height of all cells in the range.

- To autofit only the width of the columns use `xw.Range('A1:B2').columns.autofit()`
- To autofit only the height of the rows use `xw.Range('A1:B2').rows.autofit()`

0.9.0 .

clear()

Clears the content and the formatting of a Range.

clear_contents()

Clears the content of a Range but leaves the formatting.

color

Gets and sets the background color of the specified Range.

To set the color, either use an RGB tuple `(0, 0, 0)` or a color constant. To remove the background, set the color to `None`, see Examples.

RGB

tuple

Examples

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> xw.Range('A1').color = (255,255,255)
>>> xw.Range('A2').color
(255, 255, 255)
>>> xw.Range('A2').color = None
>>> xw.Range('A2').color is None
True
```

0.3.0 .

column

Returns the number of the first column in the in the specified range. Read-only.

Integer

0.3.5 .

column_width

Gets or sets the width, in characters, of a Range. One unit of column width is equal to the width of one character in the Normal style. For proportional fonts, the width of the character 0 (zero) is used.

If all columns in the Range have the same width, returns the width. If columns in the Range have different widths, returns None.

column_width must be in the range: 0 <= column_width <= 255

Note: If the Range is outside the used range of the Worksheet, and columns in the Range have different widths, returns the width of the first column.

float

0.4.0 .

columns

Returns a *RangeColumns* object that represents the columns in the specified range.

0.9.0 .

copy(*destination=None*)

Copy a range to a destination range or clipboard.

destination (xlwings.Range) – xlwings Range to which the specified range will be copied. If omitted, the range is copied to the Clipboard.

None

count

Returns the number of cells.

current_region

This property returns a Range object representing a range bounded by (but not including) any combination of blank rows and blank columns or the edges of the worksheet. It corresponds to `Ctrl-*` on Windows and `Shift-Ctrl-Space` on Mac.

Range object

delete(*shift=None*)

Deletes a cell or range of cells.

shift (*str, default None*) – Use `left` or `up`. If omitted, Excel decides based on the shape of the range.

None

end(*direction*)

Returns a Range object that represents the cell at the end of the region that contains the source range. Equivalent to pressing Ctrl+Up, Ctrl+down, Ctrl+left, or Ctrl+right.

direction (*One of 'up', 'down', 'right', 'left'*) –

Examples

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> xw.Range('A1:B2').value = 1
>>> xw.Range('A1').end('down')
<Range [Book1]Sheet1!$A$2>
>>> xw.Range('B2').end('right')
<Range [Book1]Sheet1!$B$2>
```

0.9.0 .

expand(*mode='table'*)

Expands the range according to the mode provided. Ignores empty top-left cells (unlike `Range.end()`).

mode (*str, default 'table'*) – One of `'table'` (=down and right), `'down'`, `'right'`.

Range

Examples

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> xw.Range('A1').value = [[None, 1], [2, 3]]
>>> xw.Range('A1').expand().address
$A$1:$B$2
>>> xw.Range('A1').expand('right').address
$A$1:$B$1
```

0.9.0 .

formula
Gets or sets the formula for the given Range.

formula_array
Gets or sets an array formula for the given Range.

0.7.1 .

get_address(*row_absolute=True, column_absolute=True, include_sheetname=False, external=False*)
Returns the address of the range in the specified format. `address` can be used instead if none of the defaults need to be changed.

- **row_absolute** (*bool, default True*) – Set to True to return the row part of the reference as an absolute reference.
- **column_absolute** (*bool, default True*) – Set to True to return the column part of the reference as an absolute reference.
- **include_sheetname** (*bool, default False*) – Set to True to include the Sheet name in the address. Ignored if external=True.
- **external** (*bool, default False*) – Set to True to return an external reference with workbook and worksheet name.

str

Examples

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> xw.Range((1,1)).get_address()
'$A$1'
>>> xw.Range((1,1)).get_address(False, False)
'A1'
>>> xw.Range((1,1), (3,3)).get_address(True, False, True)
'Sheet1!A$1:C$3'
>>> xw.Range((1,1), (3,3)).get_address(True, False, external=True)
'[Book1]Sheet1!A$1:C$3'
```

0.2.3 .

height
Returns the height, in points, of a Range. Read-only.

float

0.4.0 .

hyperlink
Returns the hyperlink address of the specified Range (single Cell only)

Examples

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> xw.Range('A1').value
'www.xlwings.org'
>>> xw.Range('A1').hyperlink
'http://www.xlwings.org'
```

0.3.0 .

insert(*shift=None*, *copy_origin='format_from_left_or_above'*)
Insert a cell or range of cells into the sheet.

- **shift** (*str, default None*) – Use `right` or `down`. If omitted, Excel decides based on the shape of the range.
- **copy_origin** (*str, default format_from_left_or_above*) – Use `format_from_left_or_above` or `format_from_right_or_below`. Note that this is not supported on macOS.

None

last_cell

Returns the bottom right cell of the specified range. Read-only.

Range

Example

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> rng = xw.Range('A1:E4')
>>> rng.last_cell.row, rng.last_cell.column
(4, 5)
```

0.3.5 .

left

Returns the distance, in points, from the left edge of column A to the left edge of the range. Read-only.

float

0.6.0 .

merge(*across=False*)

Creates a merged cell from the specified Range object.

across (*bool, default False*) – True to merge cells in each row of the specified Range as separate merged cells.

merge_area

Returns a Range object that represents the merged Range containing the specified cell. If the specified cell isn't in a merged range, this property returns the specified cell.

merge_cells

Returns `True` if the Range contains merged cells, otherwise `False`

name

Sets or gets the name of a Range.

0.4.0 .

number_format

Gets and sets the number_format of a Range.

Examples

```
>>> import xlwings as xw
>>> wb = xw.Book()
>>> xw.Range('A1').number_format
'General'
>>> xw.Range('A1:C3').number_format = '0.00%'
>>> xw.Range('A1:C3').number_format
'0.00%'
```

0.2.3 .

offset(*row_offset=0*, *column_offset=0*)
Returns a Range object that represents a Range that's offset from the specified range.

Range object

Range

0.3.0 .

options(*convert=None*, ***options*)
Allows you to set a converter and their options. Converters define how Excel Ranges and their values are being converted both during reading and writing operations. If no explicit converter is specified, the base converter is being applied, see *Converters and Options*.

convert (*object, default None*) – A converter, e.g. `dict`, `np.array`, `pd.DataFrame`, `pd.Series`, defaults to default converter

- **ndim** (*int, default None*) – number of dimensions
- **numbers** (type*, default None*) – type of numbers, e.g. `int`
- **dates** (type*, default None*) – e.g. `datetime.date` defaults to `datetime.datetime`
- **empty** (*object, default None*) – transformation of empty cells
- **transpose** (*Boolean, default False*) – transpose values
- **expand** (*str, default None*) – One of `'table'`, `'down'`, `'right'`

  => For converter-specific options, see *Converters and Options*.

Range object

0.7.0 .

paste(*paste=None*, *operation=None*, *skip_blanks=False*, *transpose=False*)
Pastes a range from the clipboard into the specified range.

- **paste** (*str, default None*) – One of all_merging_conditional_formats, all, all_except_borders, all_using_source_theme, column_widths, comments, formats, formulas, formulas_and_number_formats, validation, values, values_and_number_formats.
- **operation** (*str, default None*) – One of add , divide , multiply , subtract .
- **skip_blanks** (*bool, default False*) – Set to True to skip over blank cells
- **transpose** (*bool, default False*) – Set to True to transpose rows and columns.

None

raw_value

Gets and sets the values directly as delivered from/accepted by the engine that is being used (pywin32 or appscript) without going through any of xlwings data cleaning/converting. This can be helpful if speed is an issue but naturally will be engine specific, i.e. might remove the cross-platform compatibility.

resize(*row_size=None*, *column_size=None*)

Resizes the specified Range

- **row_size** (*int > 0*) – The number of rows in the new range (if None, the number of rows in the range is unchanged).
- **column_size** (*int > 0*) – The number of columns in the new range (if None, the number of columns in the range is unchanged).

Range object

Range

0.3.0 .

row

Returns the number of the first row in the specified range. Read-only.

Integer

0.3.5 .

row_height

Gets or sets the height, in points, of a Range. If all rows in the Range have the same height, returns the height. If rows in the Range have different heights, returns None.

row_height must be in the range: 0 <= row_height <= 409.5

Note: If the Range is outside the used range of the Worksheet, and rows in the Range have different heights, returns the height of the first row.

float

0.4.0 .

rows
Returns a *RangeRows* object that represents the rows in the specified range.

0.9.0 .

select()
Selects the range. Select only works on the active book.

0.9.0 .

shape
Tuple of Range dimensions.

0.3.0 .

sheet
Returns the Sheet object to which the Range belongs.

0.9.0 .

size
Number of elements in the Range.

0.3.0 .

top
Returns the distance, in points, from the top edge of row 1 to the top edge of the range. Read-only.

float

0.6.0 .

unmerge()
Separates a merged area into individual cells.

value
Gets and sets the values for the given Range.

object

returned object depends on the converter being used, see *xlwings.Range.options()*

width
Returns the width, in points, of a Range. Read-only.

float

0.4.0 .

24.2.8 RangeRows

class xlwings.RangeRows(*rng*)

Represents the rows of a range. Do not construct this class directly, use *Range.rows* instead.

Example

```
import xlwings as xw

rng = xw.Range('A1:C4')

assert len(rng.rows) == 4  # or rng.rows.count

rng.rows[0].value = 'a'

assert rng.rows[2] == xw.Range('A3:C3')
assert rng.rows(2) == xw.Range('A2:C2')

for r in rng.rows:
    print(r.address)
```

autofit()

Autofits the height of the rows.

count

Returns the number of rows.

0.9.0 .

24.2.9 RangeColumns

class xlwings.RangeColumns(*rng*)

Represents the columns of a range. Do not construct this class directly, use *Range.columns* instead.

Example

```
import xlwings as xw

rng = xw.Range('A1:C4')
```

(continues on next page)

( )

```
assert len(rng.columns) == 3  # or rng.columns.count

rng.columns[0].value = 'a'

assert rng.columns[2] == xw.Range('C1:C4')
assert rng.columns(2) == xw.Range('B1:B4')

for c in rng.columns:
    print(c.address)
```

autofit()
Autofits the width of the columns.

count
Returns the number of columns.

0.9.0 .

24.2.10 Shapes

class xlwings.main.**Shapes**(*impl*)
A collection of all `shape` objects on the specified sheet:

```
>>> import xlwings as xw
>>> xw.books['Book1'].sheets[0].shapes
Shapes([<Shape 'Oval 1' in <Sheet [Book1]Sheet1>>, <Shape 'Rectangle 1' in
↪<Sheet [Book1]Sheet1>>])
```

0.9.0 .

api
Returns the native object (`pywin32` or `appscript` obj) of the engine being used.

count
Returns the number of objects in the collection.

24.2.11 Shape

class xlwings.**Shape**(**args*, ***options*)
The shape object is a member of the *shapes* collection:

```
>>> import xlwings as xw
>>> sht = xw.books['Book1'].sheets[0]
>>> sht.shapes[0]  # or sht.shapes['ShapeName']
<Shape 'Rectangle 1' in <Sheet [Book1]Sheet1>>
```

0.9.0 .

activate()
Activates the shape.

0.5.0 .

delete()
Deletes the shape.

0.5.0 .

height
Returns or sets the number of points that represent the height of the shape.

0.5.0 .

left
Returns or sets the number of points that represent the horizontal position of the shape.

0.5.0 .

name
Returns or sets the name of the shape.

0.5.0 .

parent
Returns the parent of the shape.

0.9.0 .

top
Returns or sets the number of points that represent the vertical position of the shape.

0.5.0 .

type
Returns the type of the shape.

0.9.0 .

width
Returns or sets the number of points that represent the width of the shape.

0.5.0 .

24.2.12 Charts

class xlwings.main.Charts(*impl*)
A collection of all `chart` objects on the specified sheet:

```
>>> import xlwings as xw
>>> xw.books['Book1'].sheets[0].charts
Charts([<Chart 'Chart 1' in <Sheet [Book1]Sheet1>>, <Chart 'Chart 1' in
↪<Sheet [Book1]Sheet1>>])
```

0.9.0 .

add(*left=0*, *top=0*, *width=355*, *height=211*)

Creates a new chart on the specified sheet.

- **left** (*float, default 0*) – left position in points
- **top** (*float, default 0*) – top position in points
- **width** (*float, default 355*) – width in points
- **height** (*float, default 211*) – height in points

Chart

Examples

```
>>> import xlwings as xw
>>> sht = xw.Book().sheets[0]
>>> sht.range('A1').value = [['Foo1', 'Foo2'], [1, 2]]
>>> chart = sht.charts.add()
>>> chart.set_source_data(sht.range('A1').expand())
>>> chart.chart_type = 'line'
>>> chart.name
'Chart1'
```

api

Returns the native object (`pywin32` or `appscript` obj) of the engine being used.

count

Returns the number of objects in the collection.

24.2.13 Chart

class xlwings.Chart(*name_or_index=None*, *impl=None*)

The chart object is a member of the `charts` collection:

```
>>> import xlwings as xw
>>> sht = xw.books['Book1'].sheets[0]
>>> sht.charts[0]  # or sht.charts['ChartName']
<Chart 'Chart 1' in <Sheet [Book1]Sheet1>>
```

api

Returns the native object (`pywin32` or `appscript` obj) of the engine being used.

0.9.0 .

chart_type

Returns and sets the chart type of the chart.

0.1.1 .

delete()
Deletes the chart.

height
Returns or sets the number of points that represent the height of the chart.

left
Returns or sets the number of points that represent the horizontal position of the chart.

name
Returns or sets the name of the chart.

parent
Returns the parent of the chart.

0.9.0 .

set_source_data(*source*)
Sets the source data range for the chart.

source (Range) – Range object, e.g. `xw.books['Book1'].sheets[0].range('A1')`

top
Returns or sets the number of points that represent the vertical position of the chart.

width
Returns or sets the number of points that represent the width of the chart.

24.2.14 Pictures

class xlwings.main.Pictures(*impl*)
A collection of all `picture` objects on the specified sheet:

```
>>> import xlwings as xw
>>> xw.books['Book1'].sheets[0].pictures
Pictures([<Picture 'Picture 1' in <Sheet [Book1]Sheet1>>, <Picture 'Picture
↪2' in <Sheet [Book1]Sheet1>>])
```

0.9.0 .

add(*image, link_to_file=False, save_with_document=True, left=0, top=0, width=None, height=None, name=None, update=False*)
Adds a picture to the specified sheet.

- **image** (*str or path-like object or matplotlib.figure.Figure*) – Either a filepath or a Matplotlib figure object.
- **left** (*float, default 0*) – Left position in points.
- **top** (*float, default 0*) – Top position in points.

- **width** (*float, default None*) – Width in points. If PIL/Pillow is installed, it defaults to the width of the picture. Otherwise it defaults to 100 points.
- **height** (*float, default None*) – Height in points. If PIL/Pillow is installed, it defaults to the height of the picture. Otherwise it defaults to 100 points.
- **name** (*str, default None*) – Excel picture name. Defaults to Excel standard name if not provided, e.g. Picture 1 .
- **update** (*bool, default False*) – Replace an existing picture with the same name. Requires `name` to be set.

Picture

Examples

1. Picture

```
>>> import xlwings as xw
>>> sht = xw.Book().sheets[0]
>>> sht.pictures.add(r'C:\path\to\file.jpg')
<Picture 'Picture 1' in <Sheet [Book1]Sheet1>>
```

2. Matplotlib

```
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure()
>>> plt.plot([1, 2, 3, 4, 5])
>>> sht.pictures.add(fig, name='MyPlot', update=True)
<Picture 'MyPlot' in <Sheet [Book1]Sheet1>>
```

api
Returns the native object (`pywin32` or `appscript` obj) of the engine being used.

count
Returns the number of objects in the collection.

24.2.15 Picture

`class xlwings.Picture(`*impl=None*`)`
The picture object is a member of the *pictures* collection:

```
>>> import xlwings as xw
>>> sht = xw.books['Book1'].sheets[0]
>>> sht.pictures[0]  # or sht.charts['PictureName']
<Picture 'Picture 1' in <Sheet [Book1]Sheet1>>
```

0.9.0 .

api
Returns the native object (**pywin32** or **appscript** obj) of the engine being used.

0.9.0 .

delete()
Deletes the picture.

0.5.0 .

height
Returns or sets the number of points that represent the height of the picture.

0.5.0 .

left
Returns or sets the number of points that represent the horizontal position of the picture.

0.5.0 .

name
Returns or sets the name of the picture.

0.5.0 .

parent
Returns the parent of the picture.

0.9.0 .

top
Returns or sets the number of points that represent the vertical position of the picture.

0.5.0 .

update(*image*)
Replaces an existing picture with a new one, taking over the attributes of the existing picture.

image (*str or path-like object or matplotlib.figure.Figure*) – Either a filepath or a Matplotlib figure object.

0.5.0 .

width
Returns or sets the number of points that represent the width of the picture.

0.5.0 .

24.2.16 Names

`class xlwings.main.Names(impl)`

A collection of all `name` objects in the workbook:

```
>>> import xlwings as xw
>>> sht = xw.books['Book1'].sheets[0]
>>> sht.names
[<Name 'MyName': =Sheet1!$A$3>]
```

0.9.0 .

`add(name, refers_to)`

Defines a new name for a range of cells.

- **name** (*str*) – Specifies the text to use as the name. Names cannot include spaces and cannot be formatted as cell references.
- **refers_to** (*str*) – Describes what the name refers to, in English, using A1-style notation.

Name

0.9.0 .

api

Returns the native object (`pywin32` or `appscript` obj) of the engine being used.

0.9.0 .

count

Returns the number of objects in the collection.

24.2.17 Name

`class xlwings.Name(impl)`

The name object is a member of the *names* collection:

```
>>> import xlwings as xw
>>> sht = xw.books['Book1'].sheets[0]
>>> sht.names[0]   # or sht.names['MyName']
<Name 'MyName': =Sheet1!$A$3>
```

0.9.0 .

api

Returns the native object (`pywin32` or `appscript` obj) of the engine being used.

0.9.0 .

delete()
Deletes the name.

0.9.0 .

name
Returns or sets the name of the name object.

0.9.0 .

refers_to
Returns or sets the formula that the name is defined to refer to, in A1-style notation, beginning with an equal sign.

0.9.0 .

refers_to_range
Returns the Range object referred to by a Name object.

0.9.0 .

24.3 UDF decorators

xlwings.**func**(*category="xlwings"*, *volatile=False*, *call_in_wizard=True*)
Functions decorated with `xlwings.func` will be imported as `Function` to Excel when running Import Python UDFs .

category [int or str, default xlwings] 1-14 represent built-in categories, for user-defined categories use strings

0.10.3 .

volatile [bool, default False] Marks a user-defined function as volatile. A volatile function must be recalculated whenever calculation occurs in any cells on the worksheet. A nonvolatile function is recalculated only when the input variables change. This method has no effect if it's not inside a user-defined function used to calculate a worksheet cell.

0.10.3 .

call_in_wizard [bool, default True] Set to False to suppress the function call in the function wizard.

0.10.3 .

xlwings.**sub**()
Functions decorated with `xlwings.sub` will be imported as `Sub` (i.e. macro) to Excel when running Import Python UDFs .

xlwings.**arg**(*arg*, *convert=None*, ***options*)
Apply converters and options to arguments, see also *Range.options()*.

Examples:

Convert `x` into a 2-dimensional numpy array:

```
import xlwings as xw
import numpy as np

@xw.func
@xw.arg('x', np.array, ndim=2)
def add_one(x):
    return x + 1
```

`xlwings.ret`(*convert=None*, ***options*)

Apply converters and options to return values, see also *Range.options()*.

Examples

1) Suppress the index and header of a returned DataFrame:

```
import pandas as pd

@xw.func
@xw.ret(index=False, header=False)
def get_dataframe(n, m):
    return pd.DataFrame(np.arange(n * m).reshape((n, m)))
```

2) Dynamic array:

`expand='table'` turns the UDF into a dynamic array. Currently you must not use volatile functions as arguments of a dynamic array, e.g. you cannot use `=TODAY()` as part of a dynamic array. Also note that a dynamic array needs an empty row and column at the bottom and to the right and will overwrite existing data without warning.

Unlike standard Excel arrays, dynamic arrays are being used from a single cell like a standard function and auto-expand depending on the dimensions of the returned array:

```
import xlwings as xw
import numpy as np

@xw.func
@xw.ret(expand='table')
def dynamic_array(n, m):
    return np.arange(n * m).reshape((n, m))
```

0.10.0 .

CHAPTER 25

License

25.1 xlwings

xlwings is distributed under a BSD 3-clause license.

Copyright (C) 2014 - present, Zoomer Analytics GmbH. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS

SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

25.2 Dependencies

xlwings is built on top of a few open-source dependencies. Their licenses are listed here:

25.2.1 pywin32 (Windows only)

com subpackage

Unless stated in the specfic source file, this work is Copyright (c) 1996-2008, Greg Stein and Mark Hammond. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither names of Greg Stein, Mark Hammond nor the name of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

win32 subpackage

Unless stated in the specfic source file, this work is Copyright (c) 1994-2008, Mark Hammond All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither name of Mark Hammond nor the name of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Pythonwin subpackage

Unless stated in the specfic source file, this work is Copyright (c) 1994-2008, Mark Hammond All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither name of Mark Hammond nor the name of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

25.2.2 comtypes (Windows only)

This software is OSI Certified Open Source Software. OSI Certified is a certification mark of the Open Source Initiative.

Copyright (c) 2006-2013, Thomas Heller. Copyright (c) 2014, Comtypes Developers. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the Software), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED AS IS , WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

25.2.3 psutil (macOS only)

BSD 3-Clause License

Copyright (c) 2009, Jay Loden, Dave Daeschler, Giampaolo Rodola All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the psutil authors nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

25.2.4 Appscript (macOS only)

Appscript is released into the public domain, except for the following code:

- portions of ae.c, which are Copyright (C) the original authors:
  Original code taken from _AEmodule.c, _CFmodule.c, _Launchmodule.c
  Copyright (C) 2001-2008 Python Software Foundation.
  License: https://docs.python.org/3/license.html.
- SendThreadSafe.h/SendThreadSafe.m, which are modified versions of Apple code (https://developer.apple.com/library/archive/samplecode/AESendThreadSafe):
  Written by: DTS
  Copyright: Copyright (c) 2007 Apple Inc. All Rights Reserved.
  Disclaimer: IMPORTANT: This Apple software is supplied to you by Apple Inc.

  (Apple) in consideration of your agreement to the following terms, and your use, installation, modification or redistribution of this Apple software constitutes acceptance of these terms. If you do not agree with these terms, please do not use, install, modify or redistribute this Apple software. In consideration of your agreement to abide by the following terms, and subject to these terms, Apple grants you a personal, non-exclusive license, under Apple's copyrights in this original Apple software (the Apple Software), to use, reproduce, modify and redistribute the Apple Software, with or without modifications, in source and/or binary forms; provided that if you redistribute the Apple Software in its entirety and without modifications, you must retain this notice and the following text and disclaimers in all such redistributions of the Apple Software. Neither the name, trademarks, service marks or logos of Apple Inc. may be used to endorse or promote products derived from the Apple Software without specific prior written permission from Apple. Except as expressly stated in this notice, no other rights or licenses, express or implied, are granted by Apple herein, including but not limited to any patent rights that may be infringed by your derivative works or by other works in which the Apple Software may be incorporated. The Apple Software is provided by Apple on an AS IS basis.

  APPLE MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE APPLE SOFTWARE OR ITS USE AND OPERATION ALONE OR IN COMBINATION WITH YOUR PRODUCTS. IN NO EVENT SHALL APPLE BE LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) ARISING IN ANY WAY OUT OF THE USE, REPRODUCTION, MODIFICATION AND/OR DISTRIBUTION OF THE APPLE SOFTWARE, HOWEVER CAUSED AND WHETHER UNDER THEORY OF CONTRACT, TORT (INCLUDING NEGLIGENCE), STRICT LIABILITY OR OTHERWISE, EVEN IF APPLE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

A

activate() (*xlwings.App*), 113
activate() (*xlwings.Book*), 117
activate() (*xlwings.Shape*), 133
activate() (*xlwings.Sheet*), 120
active (*xlwings.main.Apps*), 112
active (*xlwings.main.Books*), 115
active (*xlwings.main.Sheets*), 120
add() (*xlwings.main.Apps*), 112
add() (*xlwings.main.Books*), 116
add() (*xlwings.main.Charts*), 134
add() (*xlwings.main.Names*), 139
add() (*xlwings.main.Pictures*), 136
add() (*xlwings.main.Sheets*), 120
add_hyperlink() (*xlwings.Range*), 123
address (*xlwings.Range*), 123
api (*xlwings.App*), 113
api (*xlwings.Book*), 117
api (*xlwings.Chart*), 135
api (*xlwings.main.Charts*), 135
api (*xlwings.main.Names*), 139
api (*xlwings.main.Pictures*), 137
api (*xlwings.main.Shapes*), 133
api (*xlwings.Name*), 139
api (*xlwings.Picture*), 138
api (*xlwings.Range*), 123
api (*xlwings.Sheet*), 120
App (*xlwings*), 112
app (*xlwings.Book*), 117
Apps (*xlwings.main*), 112
autofit() (*xlwings.Range*), 123
autofit() (*xlwings.RangeColumns*), 133
autofit() (*xlwings.RangeRows*), 132
autofit() (*xlwings.Sheet*), 121

B

Book (*xlwings*), 116
book (*xlwings.Sheet*), 121
books (*xlwings.App*), 113
Books (*xlwings.main*), 115

C

calculate() (*xlwings.App*), 113
calculation (*xlwings.App*), 113
caller() (*xlwings.Book*), 118
cells (*xlwings.Sheet*), 121
Chart (*xlwings*), 135
chart_type (*xlwings.Chart*), 135
Charts (*xlwings.main*), 134
charts (*xlwings.Sheet*), 121
clear() (*xlwings.Range*), 123
clear() (*xlwings.Sheet*), 121
clear_contents() (*xlwings.Range*), 123
clear_contents() (*xlwings.Sheet*), 121
close() (*xlwings.Book*), 118
color (*xlwings.Range*), 123
column (*xlwings.Range*), 124
column_width (*xlwings.Range*), 124
columns (*xlwings.Range*), 124
copy() (*xlwings.Range*), 124
count (*xlwings.main.Apps*), 112
count (*xlwings.main.Charts*), 135
count (*xlwings.main.Names*), 139
count (*xlwings.main.Pictures*), 137
count (*xlwings.main.Shapes*), 133
count (*xlwings.Range*), 125
count (*xlwings.RangeColumns*), 133
count (*xlwings.RangeRows*), 132
current_region (*xlwings.Range*), 125

D

delete() (xlwings.Chart), 136
delete() (xlwings.Name), 139
delete() (xlwings.Picture), 138
delete() (xlwings.Range), 125
delete() (xlwings.Shape), 134
delete() (xlwings.Sheet), 121
display_alerts (xlwings.App), 114

E

end() (xlwings.Range), 125
expand() (xlwings.Range), 125

F

formula (xlwings.Range), 126
formula_array (xlwings.Range), 126
fullname (xlwings.Book), 118

G

get_address() (xlwings.Range), 126

H

height (xlwings.Chart), 136
height (xlwings.Picture), 138
height (xlwings.Range), 127
height (xlwings.Shape), 134
hwnd (xlwings.App), 114
hyperlink (xlwings.Range), 127

I

index (xlwings.Sheet), 121
insert() (xlwings.Range), 127

K

keys() (xlwings.main.Apps), 112
kill() (xlwings.App), 114

L

last_cell (xlwings.Range), 128
left (xlwings.Chart), 136
left (xlwings.Picture), 138
left (xlwings.Range), 128
left (xlwings.Shape), 134

M

macro() (xlwings.App), 114
macro() (xlwings.Book), 118
merge() (xlwings.Range), 128
merge_area (xlwings.Range), 128
merge_cells (xlwings.Range), 128

N

Name (xlwings), 139
name (xlwings.Book), 118
name (xlwings.Chart), 136
name (xlwings.Name), 140
name (xlwings.Picture), 138
name (xlwings.Range), 128
name (xlwings.Shape), 134
name (xlwings.Sheet), 121
names (xlwings.Book), 118
Names (xlwings.main), 139
names (xlwings.Sheet), 121
number_format (xlwings.Range), 128

O

offset() (xlwings.Range), 129
open() (xlwings.main.Books), 116
options() (xlwings.Range), 129

P

parent (xlwings.Chart), 136
parent (xlwings.Picture), 138
parent (xlwings.Shape), 134
paste() (xlwings.Range), 129
Picture (xlwings), 137
Pictures (xlwings.main), 136
pictures (xlwings.Sheet), 122
pid (xlwings.App), 114

Q

quit() (xlwings.App), 114

R

Range (xlwings), 122
range() (xlwings.App), 115
range() (xlwings.Sheet), 122
RangeColumns (xlwings), 132
RangeRows (xlwings), 132
raw_value (xlwings.Range), 130
refers_to (xlwings.Name), 140
refers_to_range (xlwings.Name), 140
resize() (xlwings.Range), 130
row (xlwings.Range), 130
row_height (xlwings.Range), 130
rows (xlwings.Range), 131

S

save() (*xlwings.Book*), 119
screen_updating (*xlwings.App*), 115
select() (*xlwings.Range*), 131
select() (*xlwings.Sheet*), 122
selection (*xlwings.App*), 115
selection (*xlwings.Book*), 119
set_mock_caller() (*xlwings.Book*), 119
set_source_data() (*xlwings.Chart*), 136
Shape (*xlwings*), 133
shape (*xlwings.Range*), 131
Shapes (*xlwings.main*), 133
shapes (*xlwings.Sheet*), 122
Sheet (*xlwings*), 120
sheet (*xlwings.Range*), 131
sheets (*xlwings.Book*), 119
Sheets (*xlwings.main*), 120
size (*xlwings.Range*), 131

T

top (*xlwings.Chart*), 136
top (*xlwings.Picture*), 138
top (*xlwings.Range*), 131
top (*xlwings.Shape*), 134
type (*xlwings.Shape*), 134

U

unmerge() (*xlwings.Range*), 131
update() (*xlwings.Picture*), 138
used_range (*xlwings.Sheet*), 122

V

value (*xlwings.Range*), 131
version (*xlwings.App*), 115
view() (*xlwings*), 111
visible (*xlwings.App*), 115

W

width (*xlwings.Chart*), 136
width (*xlwings.Picture*), 138
width (*xlwings.Range*), 131
width (*xlwings.Shape*), 134

X

xlwings ( ), 111
xlwings.arg() (*xlwings*), 140
xlwings.func() (*xlwings*), 140
xlwings.ret() (*xlwings*), 141
xlwings.sub() (*xlwings*), 140