```
#########################################################################
# File        : using_apeer-ometiff-library.ipynb
# Version     : 0.2
# Author      : czsrh
# Date        : 13.01.2019
# Institution : Carl Zeiss Microscopy GmbH
#
# Disclaimer: Just for testing - Use at your own risk.
# Feedback or Improvements are welcome.
#########################################################################
```

Reading OME-TIFF files from Python using the apeer-ometiff-library

The APEER platform allows creating modules and workflows using basically any programming language due to its underlying Docker(TM) technology. Nevertheless Python seems to be the favorite choice for mots of the time for various reasons:

- Simple enough to be used by researchers with scripting experience
- Powerful enough to create amazing computational tools
- A huge and very open community and the whole ecosystem behind it
- Probably the most popular language when it come to topics like Machine Learning

The topic or question what is the "best" image data format for microscopy is a very interesting and also quite difficult question. There are no easy answers and there is no right or wrong here.

Since the APEER platform tries to provide solutions our team decided that we must for sure support the currently most popular image data format for microscopy image data, which clearly is OME-TIFF (despite its known limitations). Therefore we explored "easy and simple" ways to read OME-TIFF for the most common use cases. We just want a simple python-based tool to read and write OME-TIFF without the need to include JAVA etc. into the modules. Therefore we reused parts of the existing python ecosystem, especially python-bioformats and tifffile, added some extra code and created a basic PyPi package.

This package can be easily included in every APEER module but obviously it can be also used inside our python application or within jupyter notebook.

- PyPi - apeer-ometiff-library
- PyPi - python-bioformats.

More information on the source code can be found on the APEER GitHub project page: GitHub - apeer-ometiff-library

In order to make things a bit easier we create a little helper script called imgfileutils.py which can be found here

- ZEISS GitHub OAD - imagefileutils.py

The complete notebook can be found here:

- ZEISS GitHub OAD - using_apeer-ometiff-library.ipynb

Remark: The notebook uses a small subset of the original dataset because of the file size. Therefore the images used for illustration do not exactly correspond to what one will see if using the provided test files.

```
# import the libraries
from apeer_ometiff_library import io, processing, omexmlClass

# import script with some useful functions
import sys
sys.path.append(r'modules')
import imgfileutils as imf
```

```
# define your OME-TIFF file here
filename = r'testdata\CellDivision_T=10_Z=15_CH=2_DCV_small.ome.tiff'

# extract XML and save it to disk
xmlometiff = imf.writexml_ometiff(filename)
```

```
Created OME-XML file for testdata:
testdata\CellDivision_T=10_Z=15_CH=2_DCV_small.ome.tiff
```

Reading the OME-TIFF stack as an NumPy Array

The easily ready and OME-TIFF stack without the need to deal with the JAVA runtime the apeer-ometiff-library used the following code:

```
def read_ometiff(input_path):
    with tifffile.TiffFile(input_path) as tif:
        array = tif.asarray()
        omexml_string = tif[0].image_description.decode('utf-8')

    # Turn Ome XML String to an Bioformats object for parsing
    metadata = omexmlClass.OMEXML(omexml_string)

    # Parse pixel sizes
    pixels = metadata.image(0).Pixels
    size_c = pixels.SizeC
    size_t = pixels.SizeT
    size_z = pixels.SizeZ
    size_x = pixels.SizeX
    size_y = pixels.SizeY

    # Expand image array to 5D of order (T, Z, C, X, Y)
    if size_c == 1:
        array = np.expand_dims(array, axis=-3)
    if size_z == 1:
```

```
        array = np.expand_dims(array, axis=-4)
    if size_t == 1:
        array = np.expand_dims(array, axis=-5)

    return array, omexml_string
```

```
# Read metadata and array differently for OME-TIFF by using the io function of the
apeer-ometiff library

# Return value is an array of order (T, Z, C, X, Y)
array, omexml = io.read_ometiff(filename)

# get the metadata for the OME-TIFF file
metadata, add_metadata = imf.get_metadata(filename)
```

```
Image Type:  ometiff
```

```
# check the shape of Numpy array containing the pixel data
print('Array Shape: ', array.shape)

# get dimension order from metadata
print('Dimension Order (BioFormats) : ', metadata['DimOrder BF Array'])

# show dimensions and scaling
print('SizeT : ', metadata['SizeT'])
print('SizeZ : ', metadata['SizeZ'])
print('SizeC : ', metadata['SizeC'])
print('SizeX : ', metadata['SizeX'])
print('SizeY : ', metadata['SizeY'])
print('XScale: ', metadata['XScale'])
print('YScale: ', metadata['YScale'])
print('ZScale: ', metadata['ZScale'])
```

```
Array Shape:  (10, 15, 2, 256, 256)
Dimension Order (BioFormats) :  TZCYX
SizeT :  10
SizeZ :  15
SizeC :  2
SizeX :  256
SizeY :  256
XScale:  0.09057667415221031
YScale:  0.09057667415221031
ZScale:  0.32
```

```
# show the complete metadata dictionary
for key, value in metadata.items():
    # print all key-value pairs for the dictionary
    print(key, ' : ', value)
```

```
Directory  :  testdata
Filename  :  CellDivision_T=10_Z=15_CH=2_DCV_small.ome.tiff
Extension  :  ome.tiff
ImageType  :  ometiff
Name  :  CellDivision_T=10_Z=15_CH=2_DCV_small.czi #1
AcqDate  :  2016-02-12T09:41:02.491
TotalSeries  :  1
SizeX  :  256
SizeY  :  256
SizeZ  :  15
SizeC  :  2
SizeT  :  10
Sizes BF  :  [1, 10, 15, 2, 256, 256]
DimOrder BF  :  XYCZT
DimOrder BF Array  :  TZCYX
DimOrder CZI  :  None
Axes  :  None
Shape  :  None
isRGB  :  None
ObjNA  :  1.2
ObjMag  :  50.0
ObjID  :  Objective:1
ObjName  :  None
ObjImmersion  :  None
XScale  :  0.09057667415221031
YScale  :  0.09057667415221031
ZScale  :  0.32
XScaleUnit  :  µm
YScaleUnit  :  µm
ZScaleUnit  :  µm
DetectorModel  :  None
DetectorName  :  []
DetectorID  :  Detector:506
InstrumentID  :  Instrument:0
Channels  :  ['LED555', 'LED470']
ImageIDs  :  [0]
NumPy.dtype  :  None
```

```
# Here we use https://ipywidgets.readthedocs.io/en/latest/ to create some simple
and interactive controls to navigate
# through the planes of an multi-dimensional NumPy Array.
```

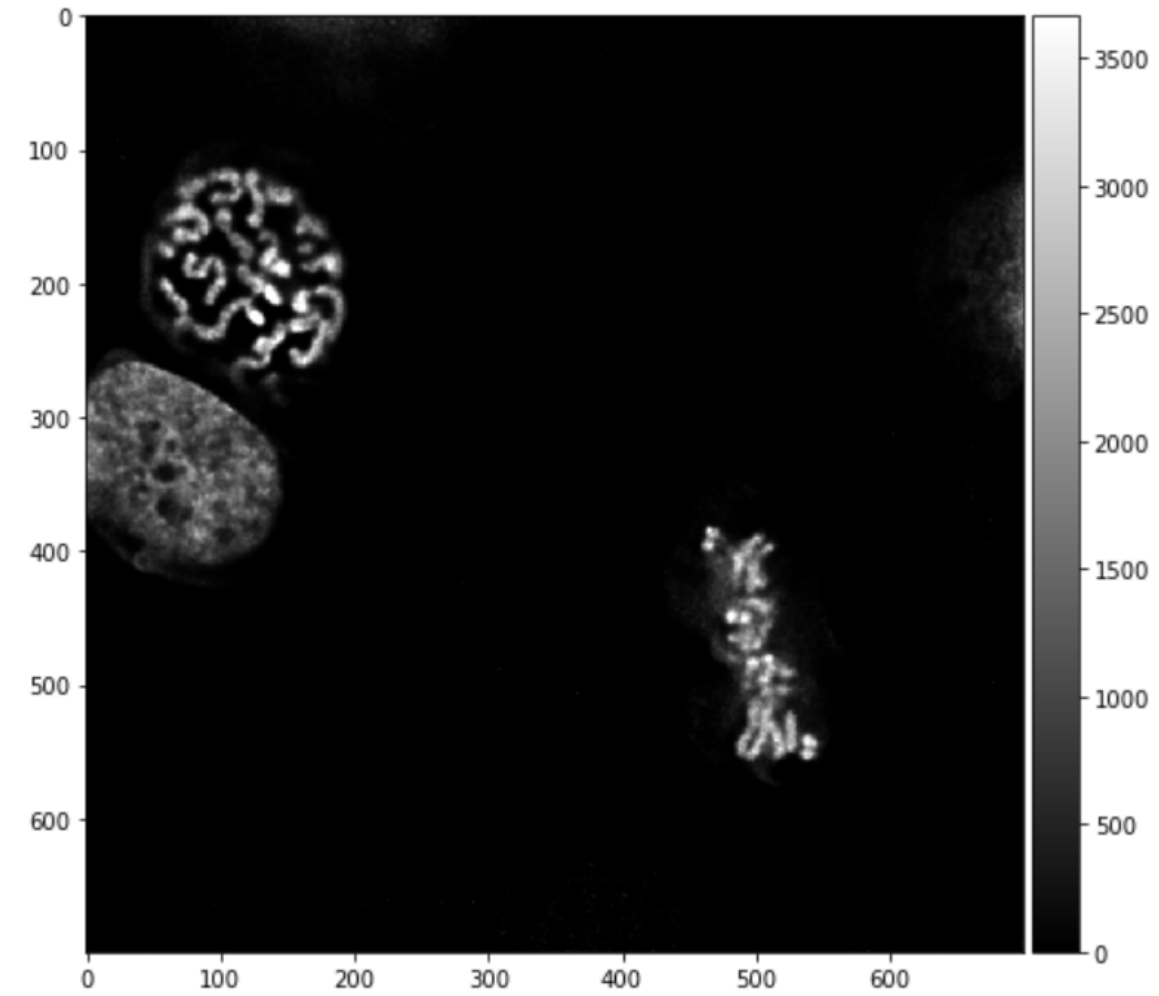
```
# display data using ipywidgets
ui, out = imf.create_ipyviewer_ome_tiff(array, metadata)

# show the interactive widget
display(ui, out)
```

```
Output()
```

```
VBox(children=(IntSlider(value=1, continuous_update=False, description='Time:',
max=10, min=1), IntSlider(valu…
```

```
Min-Max (Current Plane): 0 - 5920
```

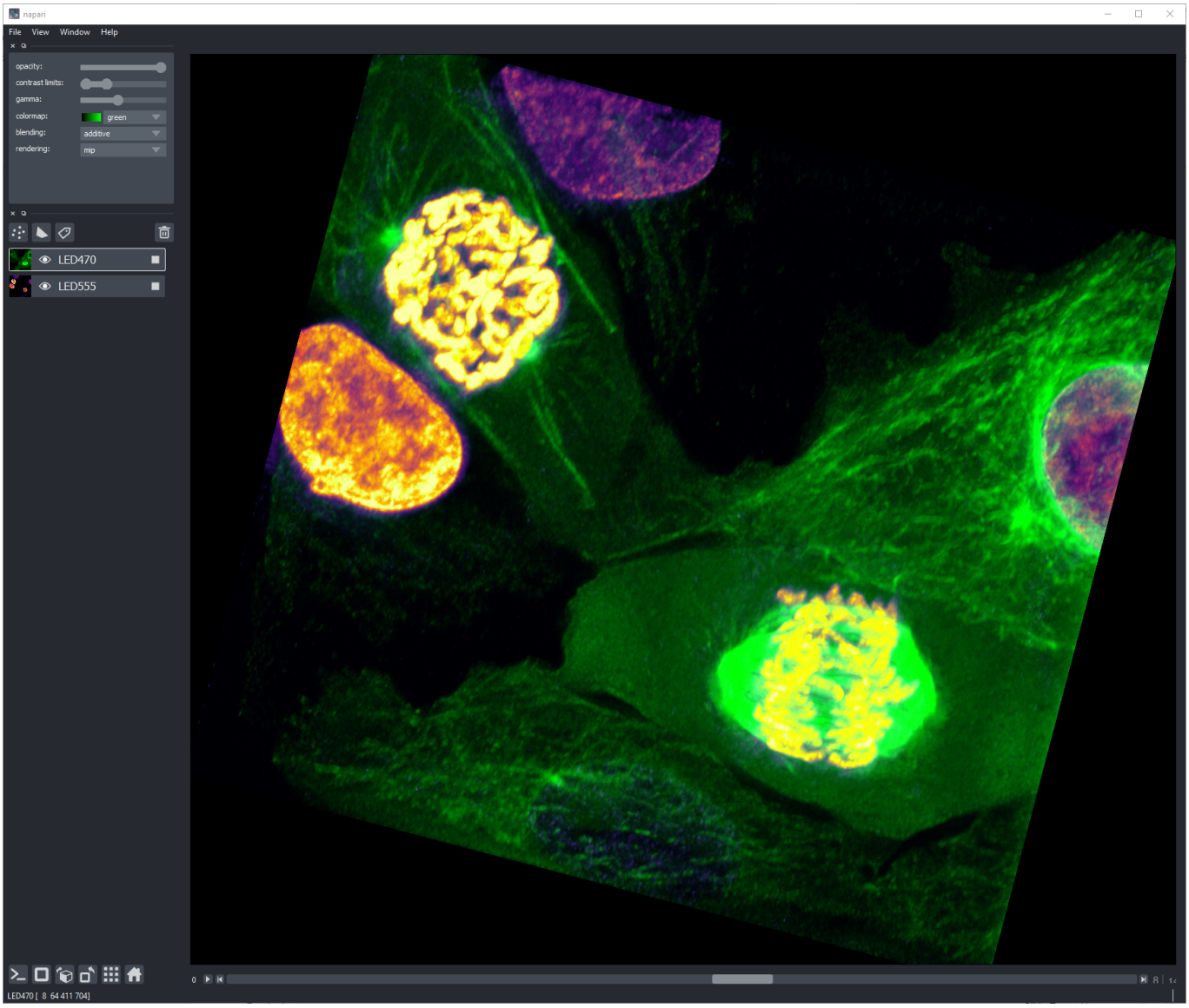



```
# Here we use the Napari viewer (https://github.com/napari/napari) to visualize the complete OME-TIFF stack,
# which is represented by a multi-dimensional NumPy Array.

# configure napari automatiaclly based on metadata and show the OME-TIFF stack
imf.show_napari(array, metadata)
```

```
Initializing Napari Viewer ...
Dim PosT :  0
Dim PosC :  2
Dim PosZ :  1
```

```
Scale Factors :  [1.0, 3.533, 1.0, 1.0, 1.0]
Shape Channel :  0 (10, 15, 256, 256)
Scaling Factors:  [1.0, 3.533, 1.0, 1.0, 1.0]
Scaling:  [0, 4927.0]
Shape Channel :  1 (10, 15, 256, 256)
Scaling Factors:  [1.0, 3.533, 1.0, 1.0, 1.0]
Scaling:  [0, 25331.0]
```



In order to create a slideshow using this notebook run the following lines from a command line:

```
cd
c:\Users\...\jupyter_notebooks\Read_CZI_and_OMETIFF_and_display_widgets_and_napari

jupyter nbconvert using_apeer-ometiff-library.ipynb --to slides --post serve
```