Contents

1 Introduction

This application note will explain how to create a workflow using the ZEN-Python connection. The basic idea is to control everything from within Python (Master). While ZEN internally is using IronPython, this notes describes how to control ZEN from within a CPython implementation

The ZEN Image Acquisition (Slave) software is "only" doing the image acquisition. The signal to start the experiment is send from Python to ZEN.

When the experiment is finished, the CZI data are imported into Python using BioFormats and "some" simple image analysis is carried out to underline the workflow concept.

2 Prerequisites

To be able to use the functionality described herein, you need

- **ZEN Blue 2012 or later with the Macro Environment module -** `http://www.zeiss.com/microscopy/en_us/products/microscope-software/zen-2012.html`
- **Python 3.5 64-bit Installation with some extensions -** `https://www.python.org/`

To profit fully from the following example, you should have working experience with both applications. You should be familiar with the user interface of your favorite python IDE and the creation and use of scripts.

2.1 Required Python Modules

In order to try out the complete example including the image analysis part some additional python modules are required. Most of the packages are included into typical python distribution like Anaconda (https://www.continuum.io/downloads). Some of the additional other modules, that might not be part of the python distribution can be typically found on PyPI (https://pypi.python.org/pypi).

- **python-bioformats**
- **Javabridge**
- **tifffile**
- **mahotas**
- **czifile.py - http://www.lfd.uci.edu/~gohlke**

2.2 Register ZEN Functionality

To be able to use ZEN services in a COM environment, the ZEN functionality must be registered as follows using the following BAT-File, which **must be executed as administrator**.

```
:: regScripting_Release.bat

echo off

pushd "C:\Windows\Microsoft.NET\Framework64\v4.0.30319"

SET dll-1="C:\Program Files\Carl Zeiss\ZEN 2\ZEN 2 (blue edition)\Zeiss.Micro.Scripting.dll"
regasm /u /codebase /tlb %dll-1%
regasm /codebase /tlb %dll-1%

SET dll-2="C:\Program Files\Carl Zeiss\ZEN 2\ZEN 2 (blue edition)\Zeiss.Micro.LM.Scripting.dll"
regasm /u /codebase /tlb %dll-2%
regasm /codebase /tlb %dll-2%
popd
pause
```

The final result of this set of commands is that the .NET classes and structures within **Zeiss.Micro.Scripting.dll** and **Zeiss.Micro.LM.Scripting.dll** are made known and accessible within the COM environment as well.

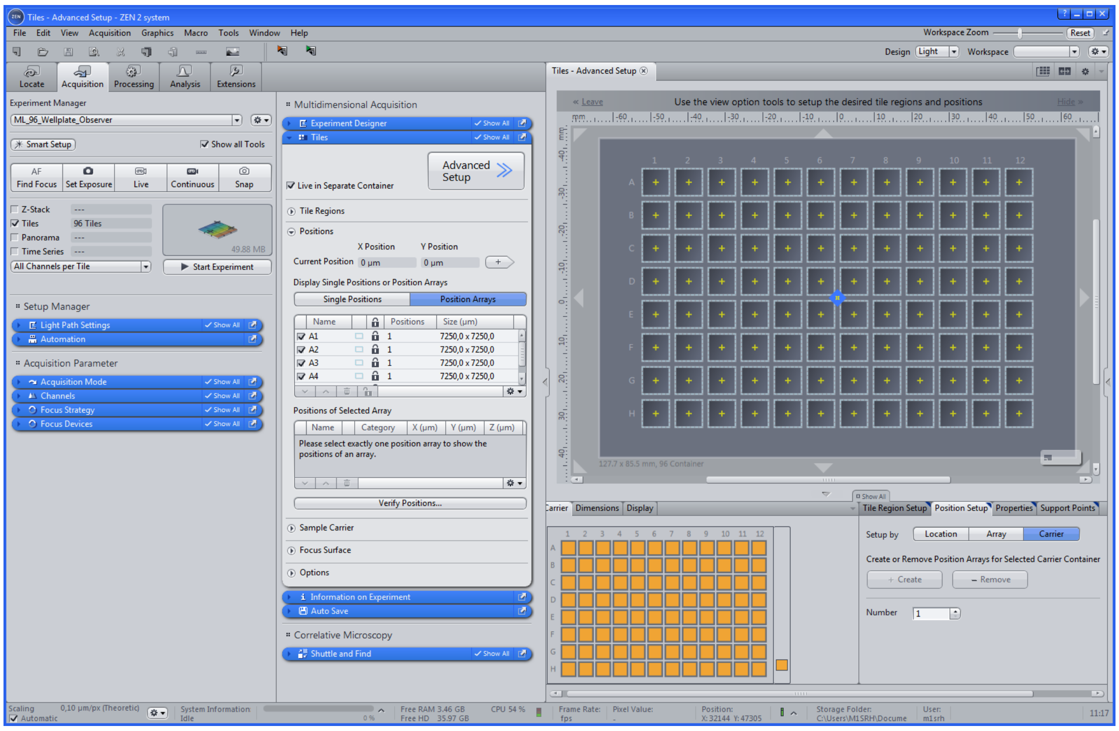
Remark: Please note that you must edit the BAT-file and adapt the path location, where the DLLs can be found to your needs.

3 ZEN Blue - Setting up the Experiment

The easiest way to setup a wellplate experiment is to use the Tile & Position Module. It is assumed that one is familiar with setting up ZEN Blue experiments in general since this is not explained in detail here.

- Specify the correct folder for your sample camera in case of a simulated experiment.
- Use Smart Setup to create a (DAPI) channel.
- Activate **Tiles**, change to **Position Setup** and select **Carrier** mode.
- Add **one** position per well.

The ZEN Blue software should look similar to this:


Tiles - Advanced Setup - ZEN 2 system
File Edit View Acquisition Graphics Macro Tools Window Help
Locate Acquisition Processing Analysis Extensions
Experiment Manager
ML_96_Wellplate_Observer
Smart Setup
Show all Tools
Find Focus Set Exposure Live Continuous Snap
Z-Stack
Tiles 96 Tiles
Panorama
Time Series
All Channels per Tile
Start Experiment
Setup Manager
Light Path Settings
Automation
Acquisition Parameter
Acquisition Mode
Channels
Focus Strategy
Focus Devices
Multidimensional Acquisition
Experiment Designer
Tiles
Advanced Setup
Live in Separate Container
Tile Regions
Positions
Current Position
Single Positions
Position Arrays
Verify Positions...
Sample Carrier
Focus Surface
Options
Information on Experiment
Auto Save
Correlative Microscopy
Shuttle and Find
Tiles - Advanced Setup
Use the view option tools to setup the desired tile regions and positions
127,7 x 85,5 mm, 96 Container
Carrier Dimensions Display
Tile Region Setup Position Setup Properties Support Points
Setup by Location Array Carrier
Create or Remove Position Arrays for Selected Carrier Container
Create Remove
Number 1

Name	Positions	Size (µm)
A1	1	7250,0 x 7250,0
A2	1	7250,0 x 7250,0
A3	1	7250,0 x 7250,0
A4	1	7250,0 x 7250,0

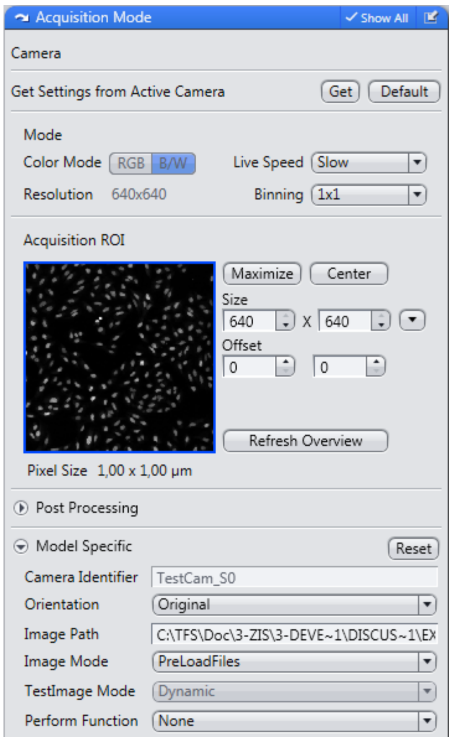
Figure 1: ZEN Blue - Wellplate Experiment Setup

3.1 ZEN - Configure a Sample Camera

The idea is to place some "real sample" images in a specific folder in order to use them for a simulated experiment including a real image analysis. This is an extremely useful feature to test and demonstrate most of the capabilities of ZEN Blue Experiments without the need to have real hardware in place.

This requires a special XML-file which is usually located at C:/ProgramData/Carl Zeiss/MTB2011/2.7.0.5/CZIS_Cameras.xml (Windows 7, depending on the current ZEN Blue and MTB version).

If configured correctly, one can specify the image folder inside ZEN on the Camera Conrol on the Locate tab or the Acquisition Mode ToolWindow on the Acquisition Tab:



Mode
Color Mode: RGB [✓ B/W]
Live Speed: Slow
Resolution: 640x640
Binning: 1x1

Acquisition ROI
Size: 640 x 640
Offset: 0 0
Pixel Size 1,00 x 1,00 µm

Camera Identifier: TestCam_S0
Orientation: Original
Image Path: C:\TFS\Doc\3-ZIS\3-DEVE~1\DISCUS~1\EX
Image Mode: PreLoadFiles
TestImage Mode: Dynamic
Perform Function: None

Figure 2: ZEN Blue - Sample Camera Configuration

4 Python - Control ZEN from a python script

The main idea behind controlling ZEN from within Python is to make Python aware of the ZEN Scripting DLLs via the .COM interface. This way you can use all the commands available inside the OAD python scripts inside ZEN directly within a normal python script inside your favorite IDE.

The main ToDo´s in Python inside this application note are:

- **Import the ZEN objects into Python and run the actual experiment.**
- **Read the CZI file using BioFormats into a numpy array.**
- **Cycle trough all the images and do some "meaningful" analysis.**
- **Display the image analysis results.**

Of course the most important part of this example is to understand the implications of the 1st point. Once the ZEN-Python bridge is established, the ZEN OAD Simple-API Interface becomes part of the running python script an can be used in conjunction with the rest of python functionality.

4.1 Control ZEN from a Python script

This is the main python script controlling the complete workflow. It establishes the connection to ZEN by making the OAD Simple-API available within python.

```
 1  # -*- coding: utf-8 -*-
 2  """
 3  File: RunZenfromPython.py
 4  Date: 03.05.2017
 5  Author: Sebastian Rhode
 6  Version: 1.1
 7
 8  This Python script demonstrates the capabilities of the .COM interface
 9  to establish a connection between ZEN Blue and Python.
10  This connection allows to use ZEN Blue OAD Simple-API objects from
11  within a python script.
12
13  Requirements:
14
15  - bftools.py, czitools.py, misctooly.py
16  - tifffile
17  - czifile.py
18  - mahotas
19  """
20
21  import win32com.client
22  import os
23  import ReadAnalyzeImageData as rad
24
25  # Define place to store the CZI file
26  savefolder = 'C:\\Python_ZEN_Output\\'
27  # check if the folder already exists
28  try:
29      os.makedirs(savefolder)
30  except OSError:
31      if not os.path.isdir(savefolder):
32          raise
33
34  # Import the ZEN OAD Scripting into Python
35  Zen = win32com.client.GetActiveObject("Zeiss.Micro.Scripting.ZenWrapperLM")
```

The most important line is number 35. Here you tell Python to load the ZEN Scripting API. Once this is done, you can use all available ZEN objects (from the Simple API) the same way one would do it inside the ZEN software.

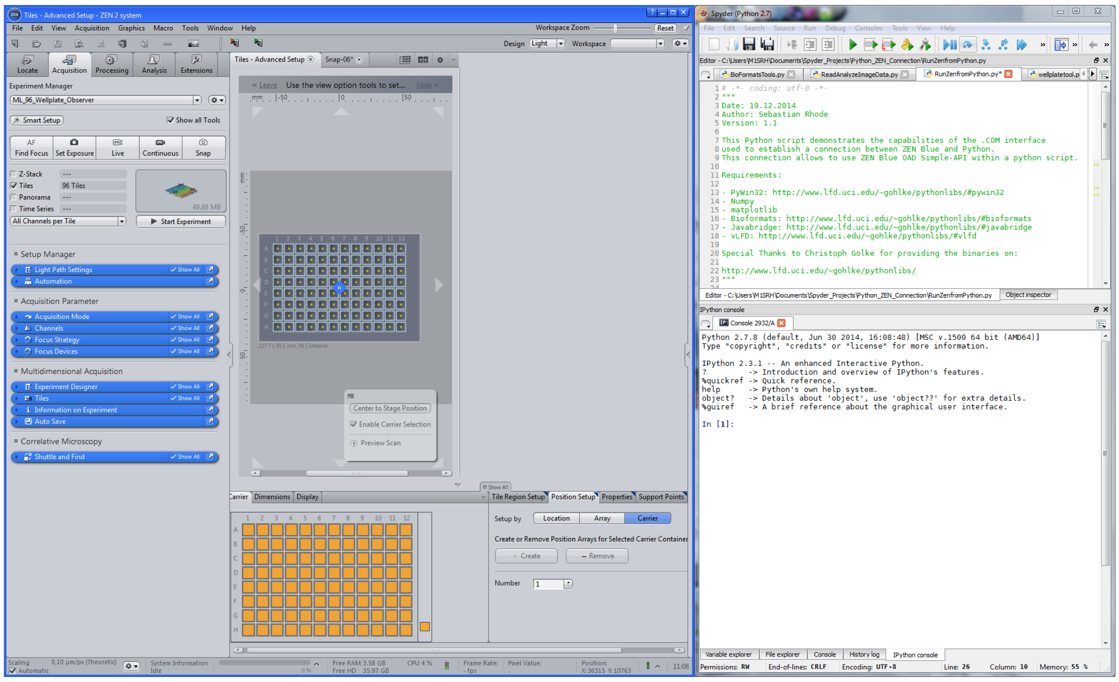
- **Define save folder and ZEN Experiment.**
- **Get the ZEN experiment using the correct name and run it.**
- **Read the CZI file using python-bioformats.**
- **Analyze the image from every well and display the results.**

Of course there is little bit more behind the scenes than shown inside this script, but all the functionality to control ZEN from within Python is located inside this script.

This opens up a whole range of exciting uses cases. You can use all the functionality of Python for image and data analysis or even hardware control while having the option to use the powerful ZEN Blue acquisition engine for all your imaging experiments and controlling the microscope itself.

4.2 ZEN-Python at work

Now we are ready to start the workflow by running **RunZenfromPython.py** from within the python IDE and watch the output.



```
# -*- coding: utf-8 -*-
"""
Date: 19.12.2014
Author: Sebastian Rhode
Version: 1.1

This Python script demonstrates the capabilities of the .COM interface
used to establish a connection between ZEN Blue and Python.
This connection allows to use ZEN Blue OAD Simple-API within a python script.

Requirements:

- PyWin32: http://www.lfd.uci.edu/~gohlke/pythonlibs/#pywin32
- Numpy
- matplotlib
- Bioformats: http://www.lfd.uci.edu/~gohlke/pythonlibs/#bioformats
- Javabridge: http://www.lfd.uci.edu/~gohlke/pythonlibs/#javabridge
- vLFD: http://www.lfd.uci.edu/~gohlke/pythonlibs/#vlfd

Special Thanks to Christoph Golke for providing the binaries on:

http://www.lfd.uci.edu/~gohlke/pythonlibs/
"""
```

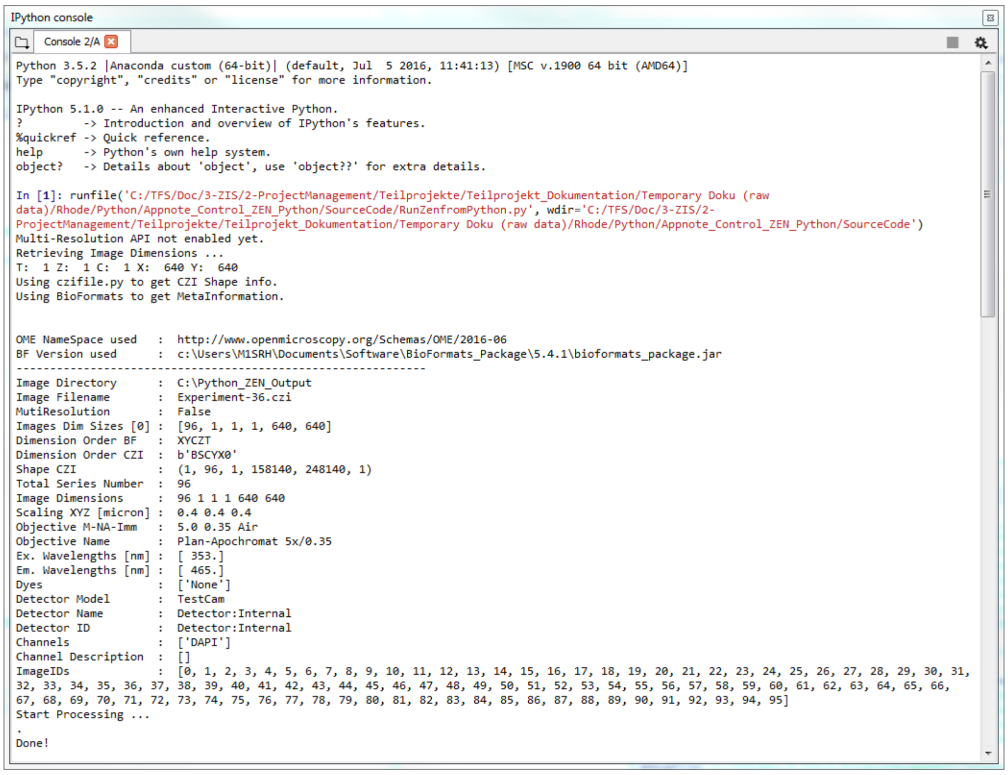
```
Python 2.7.8 (default, Jun 30 2014, 16:08:48) [MSC v.1500 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 2.3.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.
%guiref   -> A brief reference about the graphical user interface.

In [1]:
```

Figure 3: ZEN and Python running

Now one just has to start the python script inside the IDE. This will launch the actual ZEN Experiment and finally read and analyze the resulting CZI file in order to display some plots.



```
Python 3.5.2 |Anaconda custom (64-bit)| (default, Jul  5 2016, 11:41:13) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: runfile('C:/TFS/Doc/3-ZIS/2-ProjectManagement/Teilprojekte/Teilprojekt_Dokumentation/Temporary Doku (raw
data)/Rhode/Python/Appnote_Control_ZEN_Python/SourceCode/RunZenfromPython.py', wdir='C:/TFS/Doc/3-ZIS/2-
ProjectManagement/Teilprojekte/Teilprojekt_Dokumentation/Temporary Doku (raw data)/Rhode/Python/Appnote_Control_ZEN_Python/SourceCode')
Multi-Resolution API not enabled yet.
Retrieving Image Dimensions ...
T:  1 Z:  1 C:  1 X:  640 Y:  640
Using czifile.py to get CZI Shape info.
Using BioFormats to get MetaInformation.


OME NameSpace used    :  http://www.openmicroscopy.org/Schemas/OME/2016-06
BF Version used       :  c:\Users\M1SRH\Documents\Software\BioFormats_Package\5.4.1\bioformats_package.jar
-------------------------------------------------------------
Image Directory       :  C:\Python_ZEN_Output
Image Filename        :  Experiment-36.czi
MutiResolution        :  False
Images Dim Sizes [0] :  [96, 1, 1, 1, 640, 640]
Dimension Order BF    :  XYCZT
Dimension Order CZI   :  b'BSCYX0'
Shape CZI             :  (1, 96, 1, 158140, 248140, 1)
Total Series Number   :  96
Image Dimensions      :  96 1 1 1 640 640
Scaling XYZ [micron] :  0.4 0.4 0.4
Objective M-NA-Imm    :  5.0 0.35 Air
Objective Name        :  Plan-Apochromat 5x/0.35
Ex. Wavelengths [nm] :  [ 353.]
Em. Wavelengths [nm] :  [ 465.]
Dyes                  :  ['None']
Detector Model        :  TestCam
Detector Name         :  Detector:Internal
Detector ID           :  Detector:Internal
Channels              :  ['DAPI']
Channel Description   :  []
ImageIDs              :  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66,
67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95]
Start Processing ...
.
Done!
```

Figure 4: Python - Output of the running script inside console

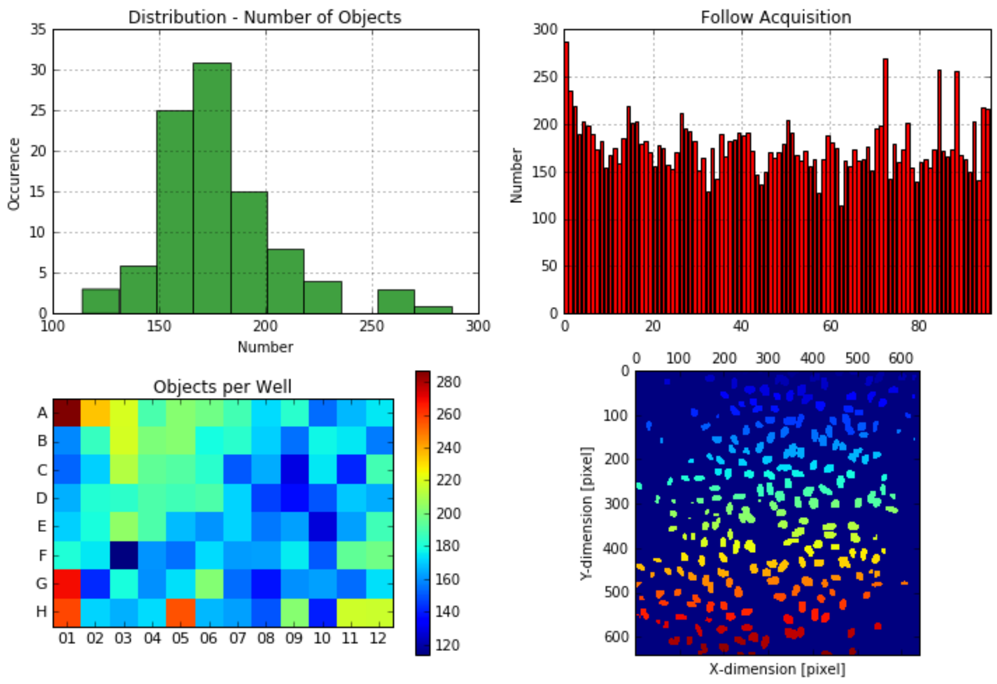


Figure 5: Python - Visulization of Image Analysis Results

4.3 Importing CZI files into Python

ZEN Blue uses the CZI data format to store the data. Due to its open concept it is very well supported by the BioFormats library. Further info can be found at:

- www.zeiss.com/czi

Instructions on the usage of the Python wrapper for BioFormats can be found here:

- http://pythonhosted.org//python-bioformats
- https://github.com/CellProfiler/python-bioformats/

There is more than one way to import a CZI file, so the choice depends on the nature of the data. For this example we use a pretty generic approach to read the CZI dataset.

4.4 Importing the CZI image data into a NumPy array

A very important topic are the CZI Metadata. Again we can rely on the BioFormats library and just use the already existing functionality to get all the information we need. For the example we use the **get_image6d** function provided by the **bftools.py** script.

4.5 Displaying the analysis results

To make this example more descriptive, a few additional scripts will be used to display the data and demonstrate the idea to automate a complete workflow within Python. ZEN is a very powerful image acquisition software, but especially when it comes data analysis and crunching numbers, one may needs to leave the ZEN world. For this example the python package **mahotas** was used to do the image analysis and **matplotlib** was used to create the plots. The python script to create the simple heatmap can be found in **5.2.3**.

5 Appendix: Python Scripts

5.1 RunZenfromPython.py

This is the main Python script controlling the complete workflow.

```python
# -*- coding: utf-8 -*-
"""
File: RunZenfromPython.py
Date: 03.05.2017
Author: Sebastian Rhode
Version: 1.1

This Python script demonstrates the capabilities of the .COM interface
to establish a connection between ZEN Blue and Python.
This connection allows to use ZEN Blue OAD Simple-API objects from
within a python script.

Requirements:

- bftools.py, czitools.py, misctooly.py
- tifffile
- czifile.py
- mahotas
"""

import win32com.client
import os
import ReadAnalyzeImageData as rad

# Define place to store the CZI file
savefolder = 'C:\\Python_ZEN_Output\\'
# check if the folder already exists
try:
    os.makedirs(savefolder)
except OSError:
    if not os.path.isdir(savefolder):
        raise

# Import the ZEN OAD Scripting into Python
Zen = win32com.client.GetActiveObject("Zeiss.Micro.Scripting.ZenWrapperLM")

# Define the experiment to be executed
ZEN_Experiment = "ML_96_Wellplate_Castor.czexp"

# run the experiment in ZEN and save the data to the specified folder
exp = Zen.Acquisition.Experiments.GetByName(ZEN_Experiment)
img = Zen.Acquisition.Execute(exp)

# Show the image in ZEN
Zen.Application.Documents.Add(img)

# Use the correct save method - it is polymorphic ... :)
filename = savefolder + img.Name_2
img.Save_2(filename)

# get the actual CZI image data using Python wrapper for BioFormats
img6d = rad.ReadImage(filename)

# Analyze the images - Example: Count Cells
obj, labeled = rad.CountObjects(img6d)

# Display some data
rad.DisplayData_2(obj, labeled)
```

5.2 Additional Python Scripts

This main script uses the following additional tools:

- **bftools.py**
- **czitools.py**
- **bftools.py**
- **ReadAnalyszeImageData.py**
- **wellplatetool.py**

5.2.1 Package BioFormatsRead

This is a little collection of python tools used to actually read the CZI image data via BioFormats. It used the python wrapper of the BioFormats library and czifile.py to read CZI image data into a NumPy array.

The scripts and additional tools can be also found here:

https://github.com/sebi06/BioFormatsRead

For more detailed information on the usage please check the readme on the project page

5.2.2 ReadAnalyzeImageData.py

```python
# -*- coding: utf-8 -*-
"""
File: ReadAnalyzeImageData.py
Date: 03.05.2017
Author: Sebastian Rhode
Version: 0.3
"""

from matplotlib import pyplot as plt, cm
import numpy as np
import mahotas
import wellplatetool as wp
import sys
import bftools as bf


def ReadImage(filename):

    urlnamespace = 'http://www.openmicroscopy.org/Schemas/OME/2016-06'
    # specify bioformats_package.jar to use if required
    bfpackage = r'c:\Users\M1SRH\Documents\Software\BioFormats_Package\5.4.1\bioformats_package.jar'
    bf.set_bfpath(bfpackage)

    # get image meta-information
    MetaInfo = bf.get_relevant_metainfo_wrapper(filename, namespace=urlnamespace, bfpath=bfpackage, showinfo=False)
    img6d, readstate = bf.get_image6d(filename, MetaInfo['Sizes'])

    # show relevant image Meta-Information
    bf.showtypicalmetadata(MetaInfo, namespace=urlnamespace, bfpath=bfpackage)

    return img6d


def CountObjects(img6d):

    obj = np.zeros(img6d.shape[0])

    print('Start Processing ...',)
    steps = img6d.shape[0]/10

    # count cells with individual thresholds per frame
    for i in range(0, img6d.shape[0], 1):

        img = img6d[i, 0, 0, 0, :, :]
        T = mahotas.otsu(img)
        img = (img > T)
        img = mahotas.gaussian_filter(img, 0.5)
        labeled, numobjects = mahotas.label(img)
        obj[i] = numobjects
        if i%steps == 0:
            print('\b.',)
            sys.stdout.flush()

    print('Done!',)

    return obj, labeled


def DisplayData_1(obj, labeled):

    fig1 = plt.figure()
    ax1 = fig1.add_subplot(111)
    cax = ax1.matshow(labeled, interpolation='nearest', cmap=cm.jet)
    cbar = fig1.colorbar(cax)
    ax1.set_title('Example - Labeled Objects')
    ax1.set_xlabel('X-dimension [pixel]')
    ax1.set_ylabel('Y-dimension [pixel]')

    fig2 = plt.figure()
    ax2 = fig2.add_subplot(111)
    ax2.hist(obj, normed=0, facecolor='green', alpha=0.75)
    ax2.set_title('Distribution - Number of Objects')
    ax2.set_xlabel('Number')
    ax2.set_ylabel('Occurence')
    ax2.grid(True)

    wp.ShowPlateData(obj, 8, 12, 'Object Number')

    # show plots
    plt.show()


def DisplayData_2(obj, labeled):

    fig = plt.figure(figsize=(12, 8), dpi=100)
    ax1 = fig.add_subplot(221)
    ax1.hist(obj, normed=0, facecolor='green', alpha=0.75)
    ax1.set_title('Distribution - Number of Objects')
    ax1.set_xlabel('Number')
    ax1.set_ylabel('Occurence')
    ax1.grid(True)

    ax2 = fig.add_subplot(222)
    ind = np.arange(obj.shape[0])
    ax2.bar(ind, obj, width=0.8, color='r')
    ax2.set_title('Follow Acquisition')
    ax2.set_ylabel('Number')
    ax2.set_xlim(0, obj.shape[0])
    ax2.grid(True)

    ax3 = fig.add_subplot(223)
    Nc = 12
    Nr = 8
    WellData, labelx, labely, fs =  wp.ReturnPlateHeatmap(obj, Nr, Nc)
    cax = ax3.imshow(WellData, interpolation='nearest', cmap=cm.jet)
    cbar = fig.colorbar(cax)
    ax3.set_title('Objects per Well')
    ax3.set_xticks(np.arange(0, Nc, 1))
    ax3.set_xticklabels(labelx, fontsize=fs)
    ax3.set_yticks(np.arange(0, Nr, 1))
    ax3.set_yticklabels(labely, fontsize=fs)

    ax4 = fig.add_subplot(224)
    cax = ax4.matshow(labeled, interpolation='nearest', cmap=cm.jet)
    ax4.set_xlabel('X-dimension [pixel]')
    ax4.set_ylabel('Y-dimension [pixel]')

    # show plots
    plt.show()
```

5.2.3 wellplatetools.py

```python
# -*- coding: utf-8 -*-
"""
File: wellplatetool.py
Date: 03.05.2017
Author: Sebastian Rhode
Version: 1.2
"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm


def ExtractLabels(Nr, Nc):

    # labeling schemes
    LabelX = ['01','02','03','04','05','06','07','08','09','10','11','12',
              '13','14','15','16','17','18','19','20','21','22','23','24',
              '25','26','27','28','29','30','31','32','33','34','35','36',
              '37','38','39','40','41','42','43','44','45','46','47','48',]

    LabelY = ['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P',
              'Q','R','S','T','U','V','W','X','Y','Z','AA','AB','AC','AD','AE','AF']

    Lx = LabelX[0:Nc]
    Ly = LabelY[0:Nr]

    return Lx, Ly


def ShowPlateData(data, Nr, Nc, parameter):

    # reshape data to create 2D matrix
    WellData = data.reshape(Nr, Nc)
    [labelx, labely] = ExtractLabels(Nr, Nc)

    # create figure
    fig = plt.figure(figsize=(6, 4), dpi=100)
    ax1 = fig.add_subplot(111)
    # set colormap
    cmap = cm.jet
    # show the well plate as an image
    cax = ax1.imshow(WellData, interpolation='nearest', cmap=cmap)
    # determine an appropriate font size
    if Nr <= 32 and Nr > 16:
        fs = 7
    elif Nr <= 16 and Nr > 8:
        fs = 9
    elif Nr <= 8:
        fs = 11
    # format the display
    ax1.set_xticks(np.arange(0, Nc, 1))
    ax1.set_xticklabels(labelx, fontsize=fs)
    ax1.set_yticks(np.arange(0, Nr, 1))
    ax1.set_yticklabels(labely, fontsize=fs)
    ax1.set_title(parameter)
    cbar = fig.colorbar(cax)


def ReturnPlateHeatmap(data, Nr, Nc):

    # reshape data to create 2D matrix
    WellData = data.reshape(Nr, Nc)
    [labelx, labely] = ExtractLabels(Nr, Nc)

    ## determine an appropriate font size
    if Nr <= 32 and Nr > 16:
        fontsize = 7
    elif Nr <= 16 and Nr > 8:
        fontsize = 9
    elif Nr <= 8:
        fontsize = 11

    return WellData, labelx, labely, fontsize
```

6 Acknowledgements and Links

Special thanks to Christoph Gohlke for providing the Windows binaries for the providing the python extension packages at:

- `http://www.lfd.uci.edu/~gohlke/pythonlibs/`

and the CellProfiler and the BioFormats Team for providing the BioFormats library and the respective python wrapper.

- `https://github.com/CellProfiler/python-bioformats/`
- `www.http://loci.wisc.edu/software/bio-formats`

Thanks to Luis Pedro Coelho for the mahotas package.

Parts of the **bftools.py** script are based on the project lesion by Juan Nunez-Iglesias:

- `https://github.com/jni/lesion`

7 Disclaimer

This is an application note free to use for everybody. Use it on your own risk.

Carl Zeiss Microscopy GmbH's ZEN software allows to connect to a the third party software Python. Therefore Carl Zeiss Microscopy GmbH undertakes no warranty concerning Python, makes no representation that Python will work on your hardware and will not be liable for any damages caused by the use of this extension. By running this example you agree to this disclaimer.