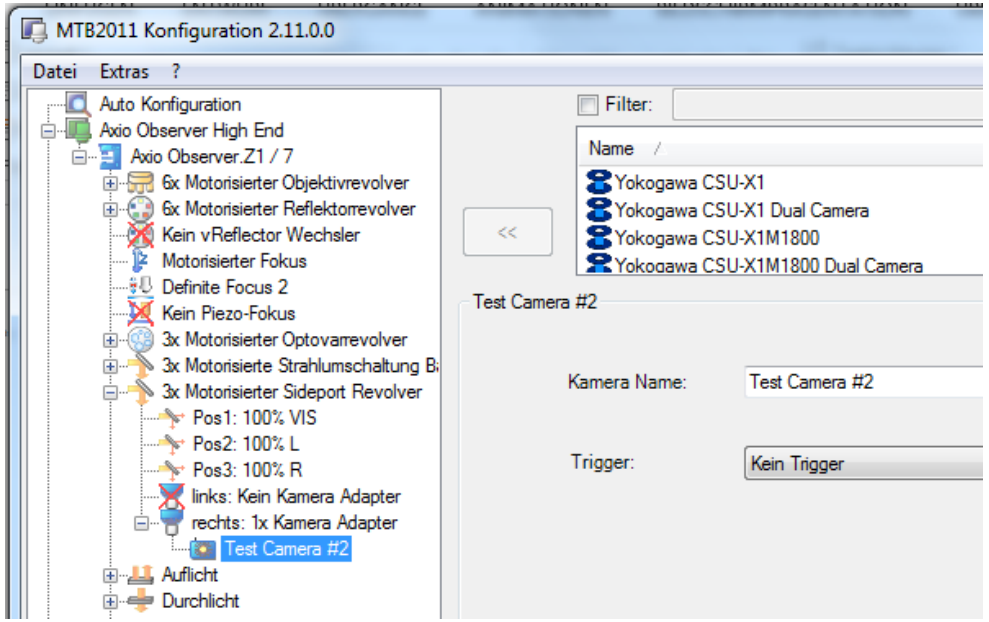
ZEN (blue edition) 3.1
Image Analysis

Dr. Marion Lang
Product Management
2019-11-04

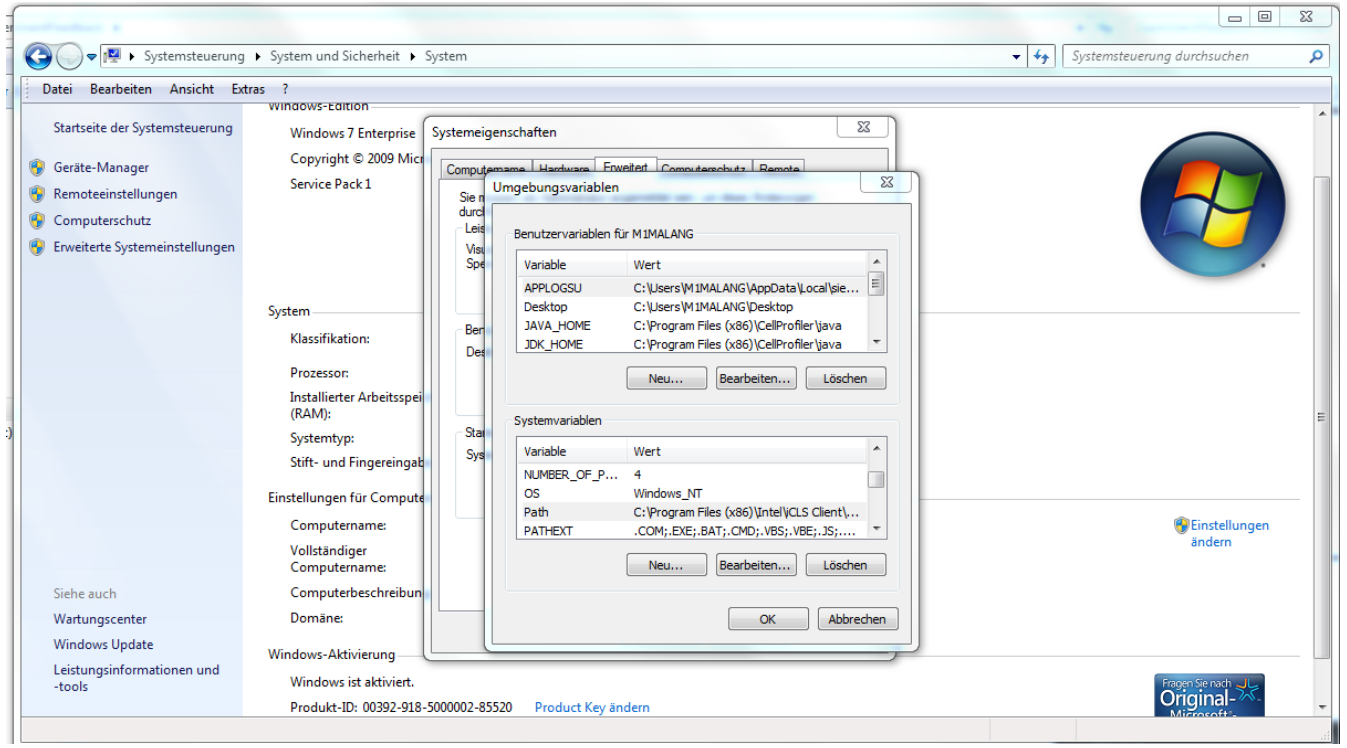
Getting Started
Simulate Hardware

Copy **Active Configuration.xml** and **CZIS_Cameras.xml** to
C:\ProgramData\Carl Zeiss\MTB2011\2.16.0.9

Start MTB → Check "Simulate"
C:\Program Files\Carl Zeiss\MTB 2011 - 2.11.0.0\MTB Configuration\MTBConfig.exe
Check that there is a demo camera

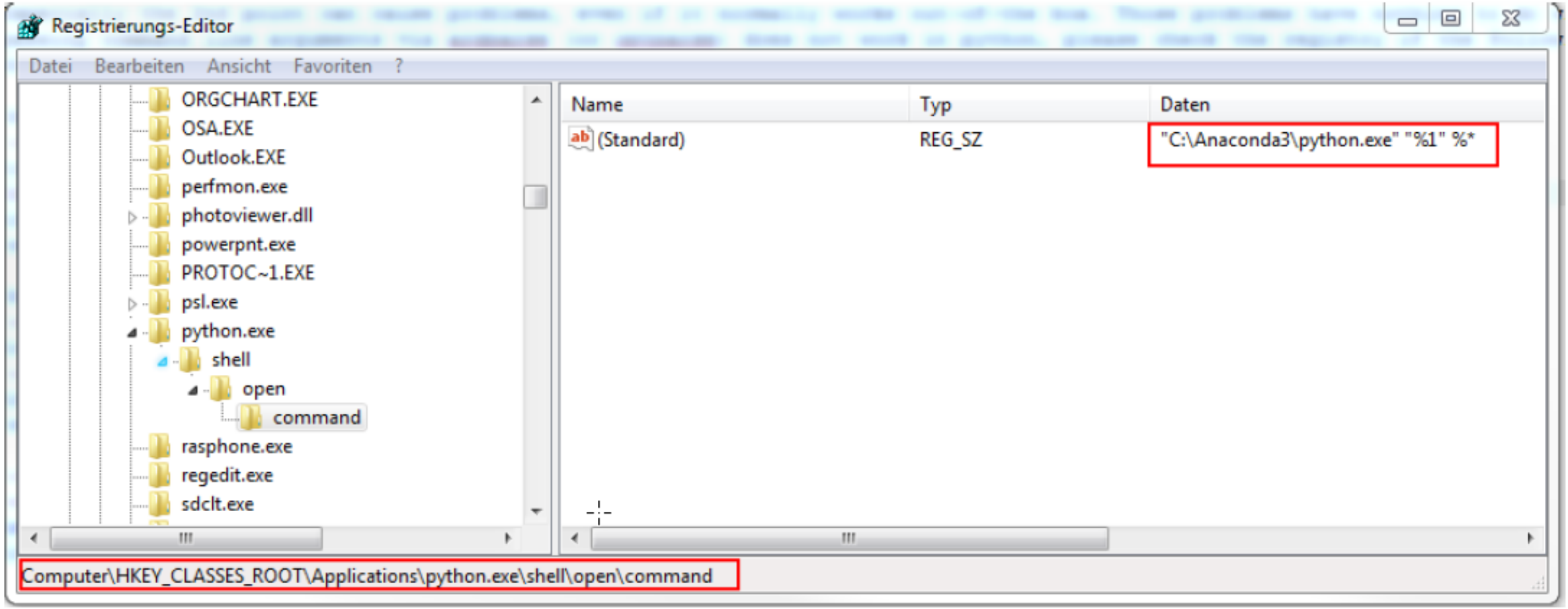


Add Python to the PATH variable

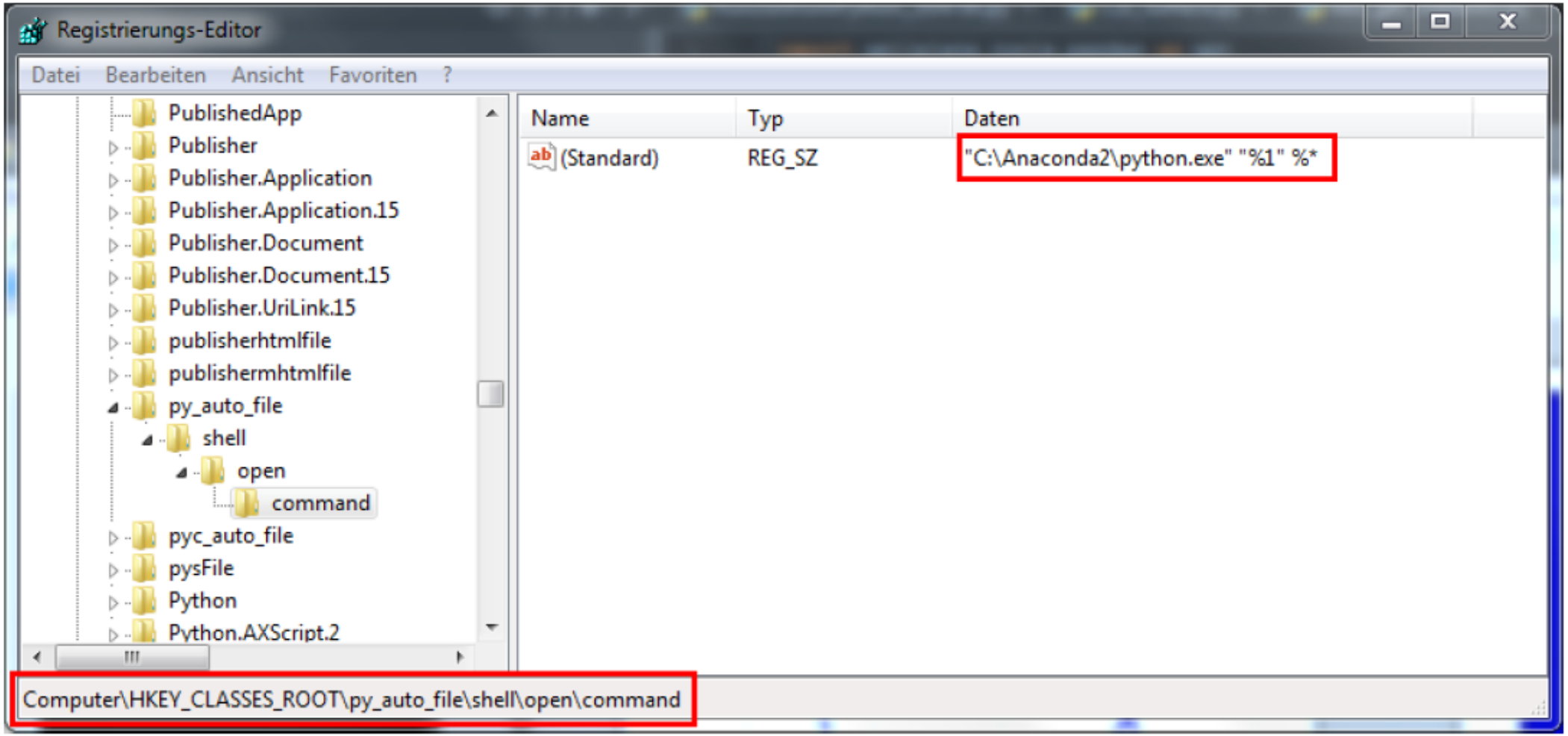


Alternatively: In the script specify the whole path to the python.exe

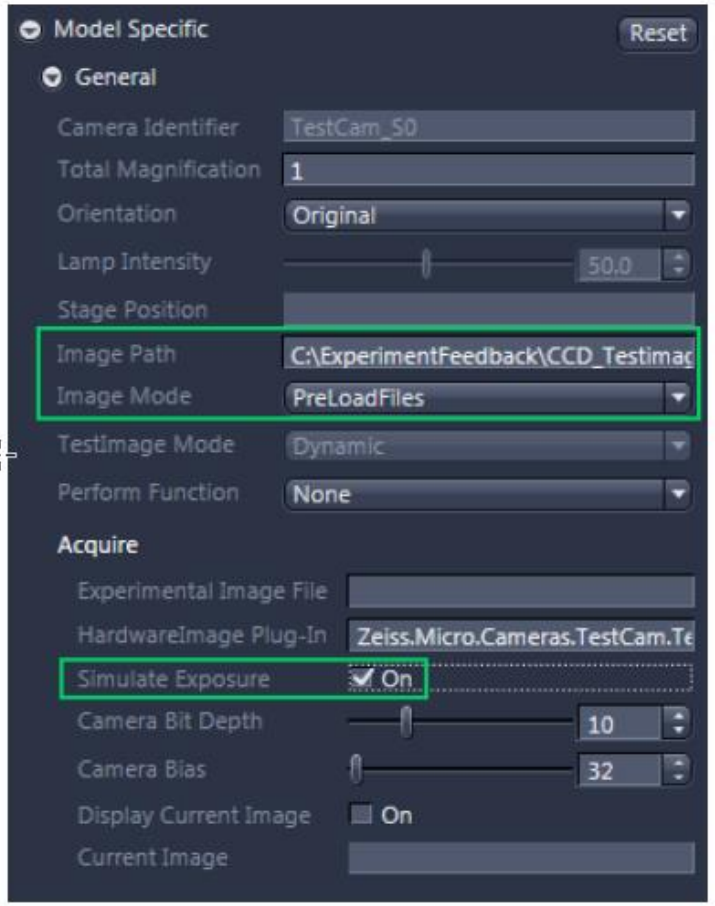
Registry Entry



Registry Entry



Simulate the Acquisition

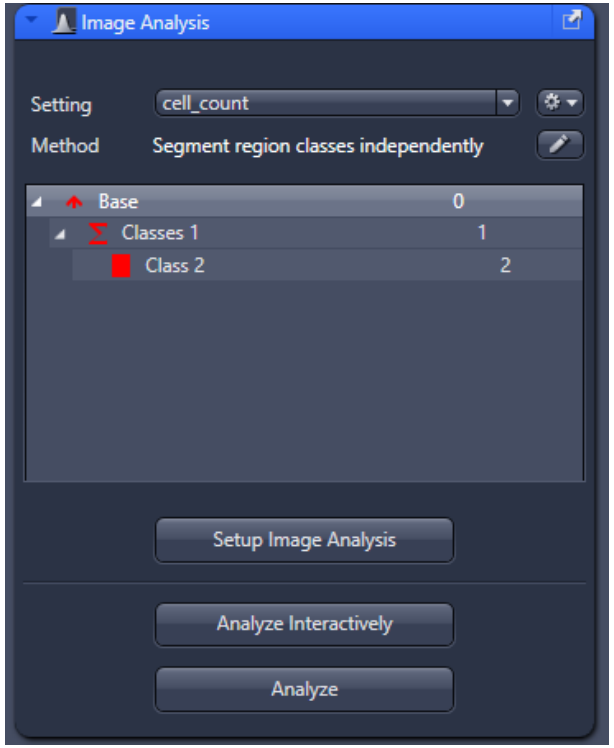


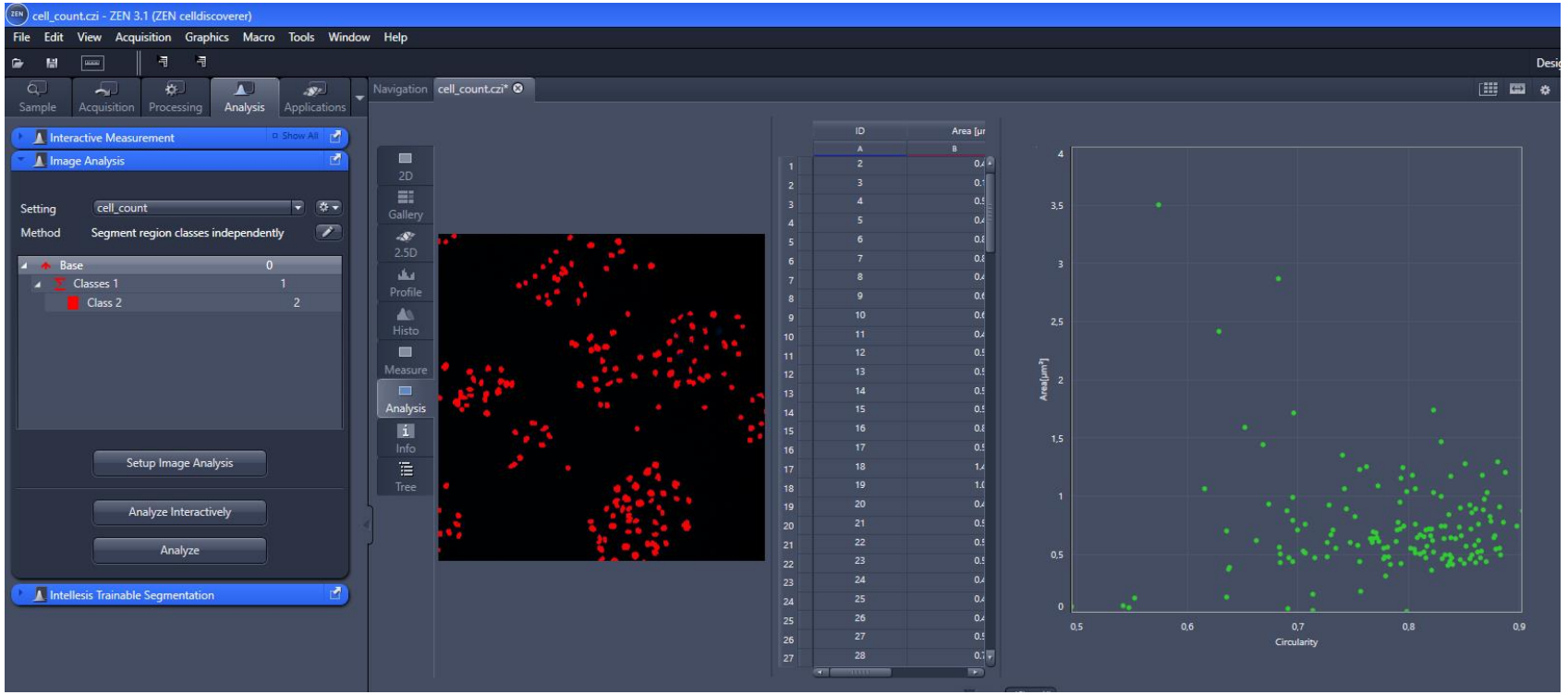
Set path to demo files, e.g.
…\CCD_Testimages_Folder\96_well

Familiarize with ZEN Application and Image Analysis

Task: Acquire some images and set up a simple image analysis with cell counting.

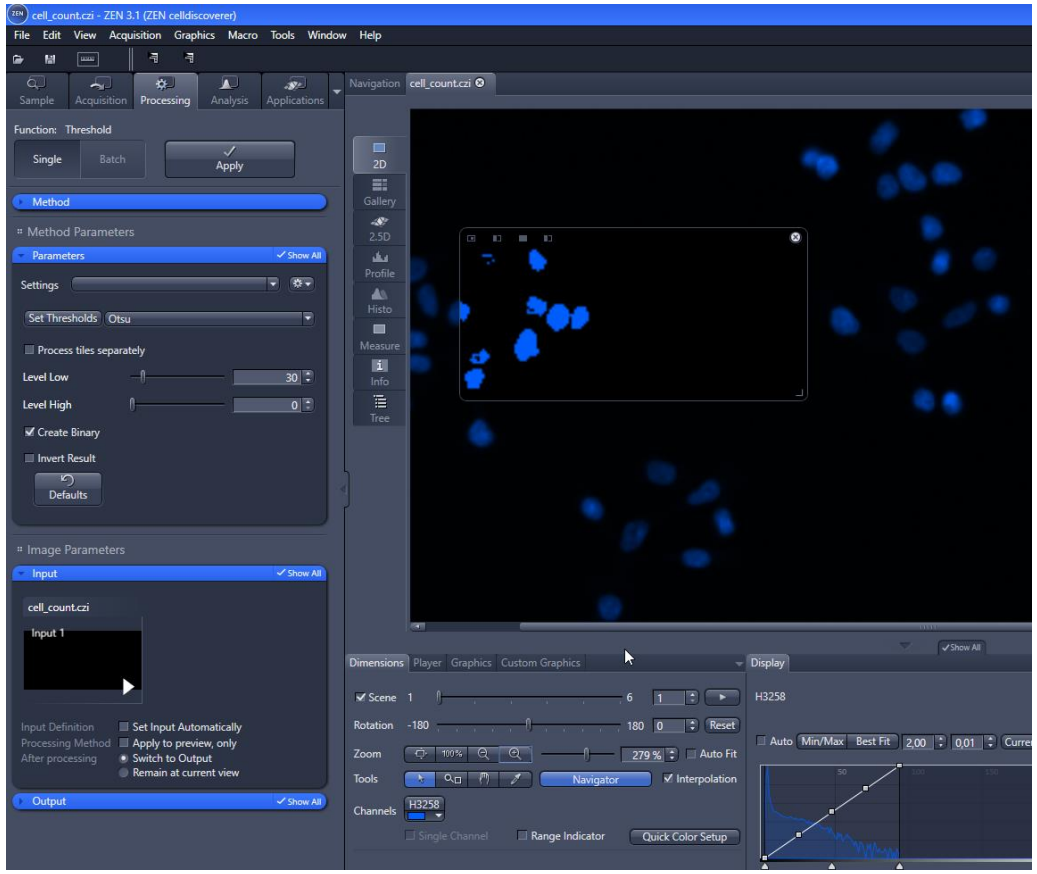
Testdata: …\Simple Cell Count\Cell Count.czi





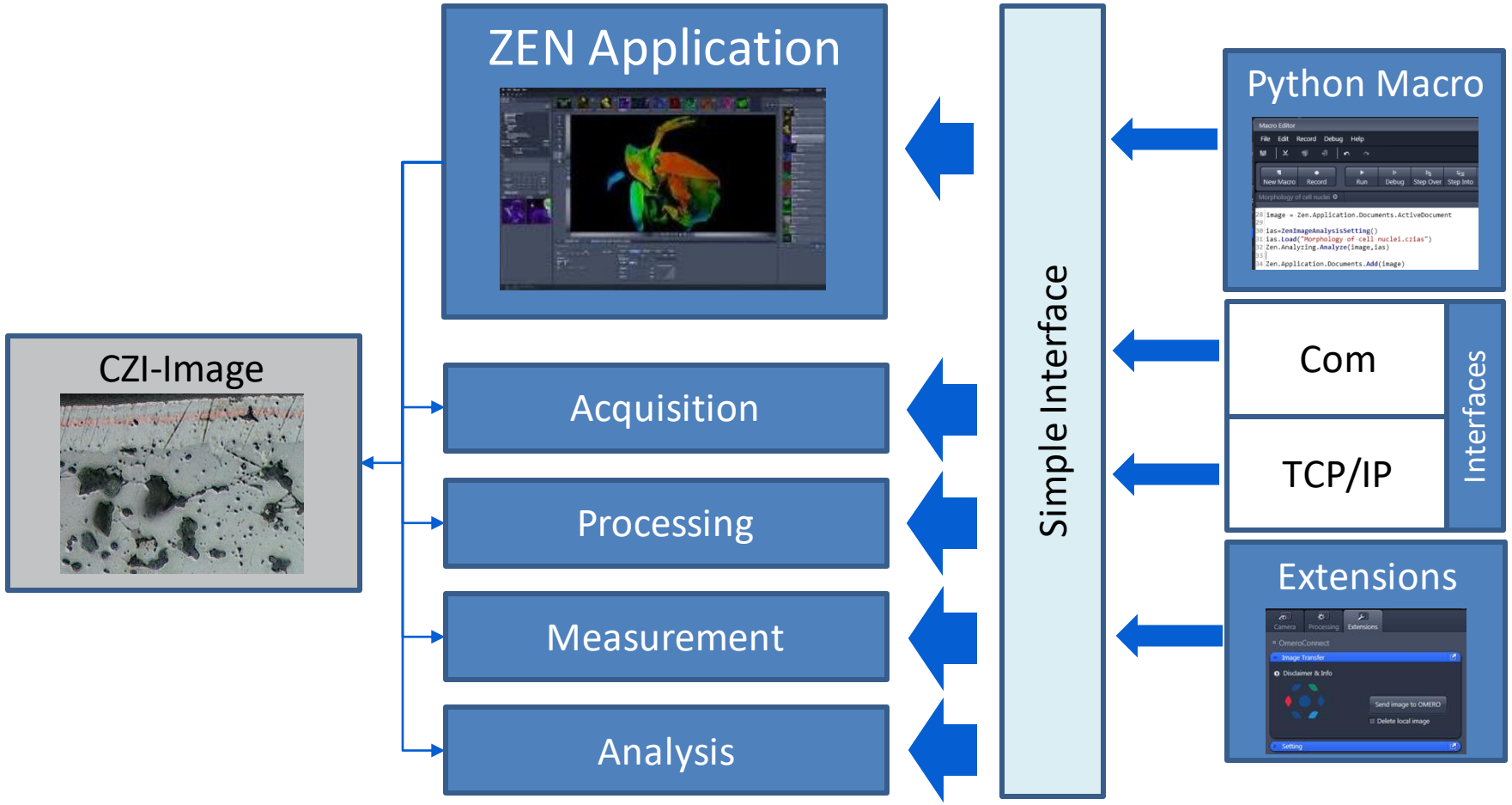
Familiarize with ZEN Application and Image Processing

Task: Apply an Image Processing Function to the data



ZEN blue

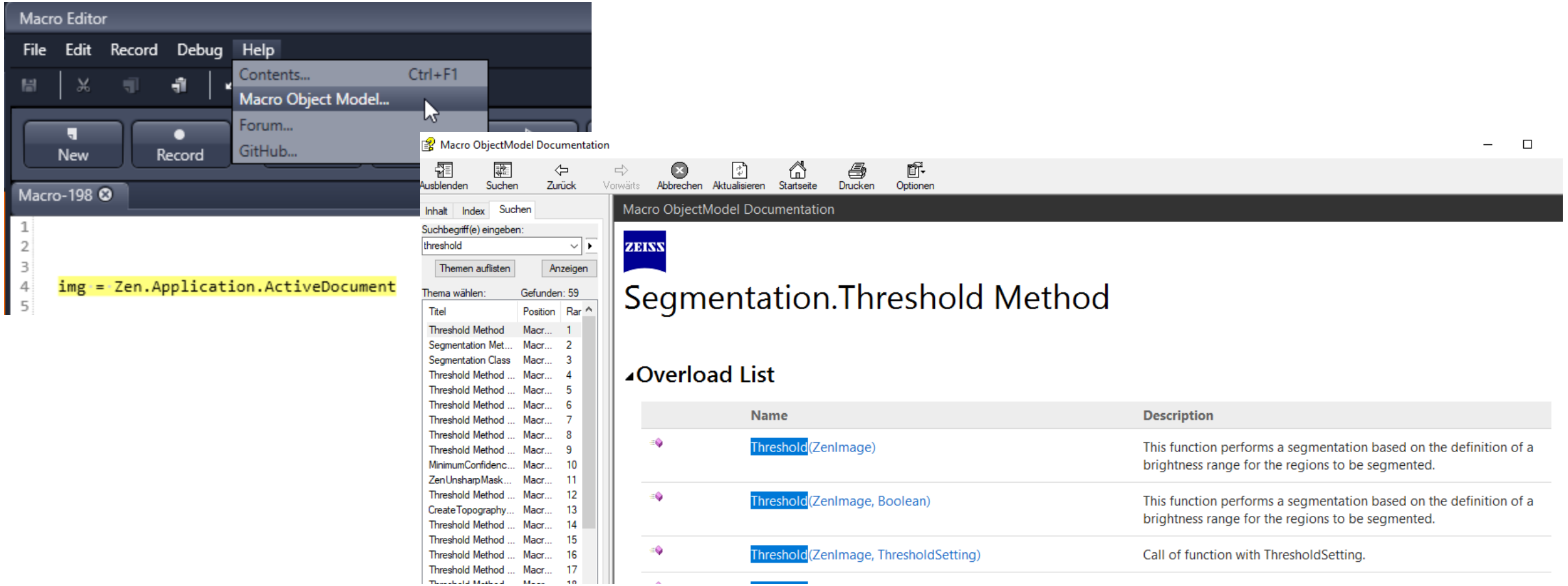
OAD-Concept and Interfaces



OAD Macro Development: Getting Help

Macro object model

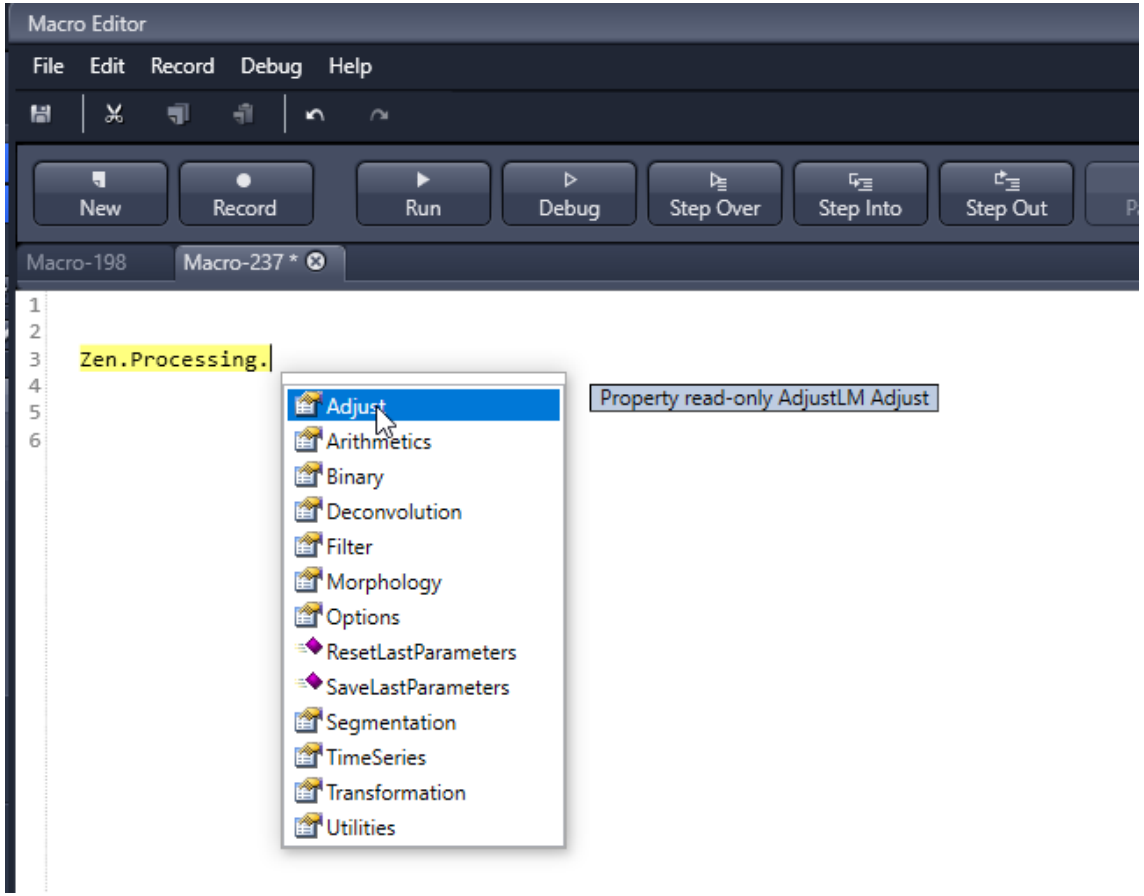
Macro Object Model



OAD Macro Development: Getting Help

Intellisense Auto-Completion

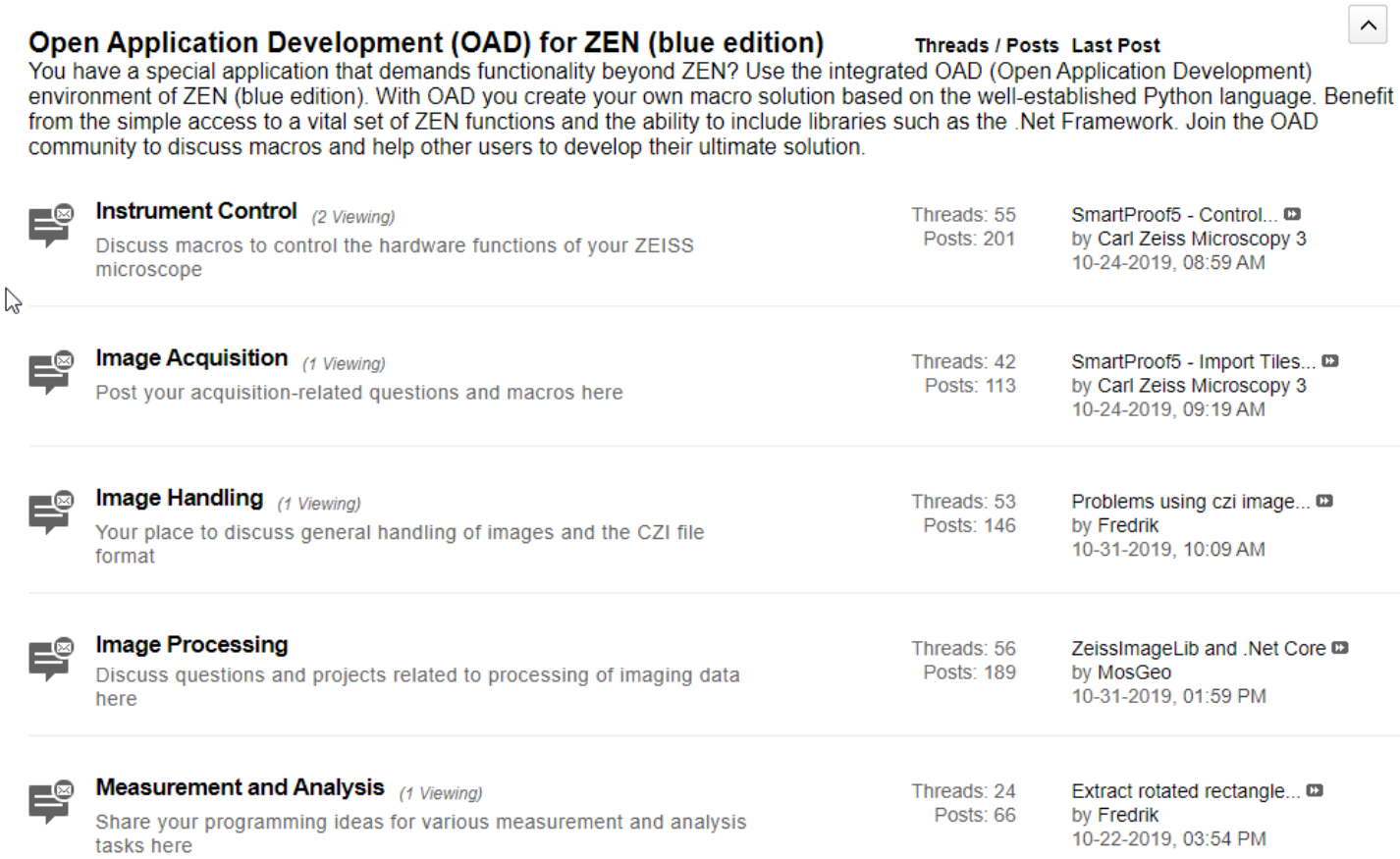
Intellisense Auto-Completion
type **Zen / Zen.**



OAD Macro Development: Getting Help

OAD Forum

http://forums.zeiss.com/microscopy/community/



Open Application Development (OAD) for ZEN (blue edition) Threads / Posts Last Post

You have a special application that demands functionality beyond ZEN? Use the integrated OAD (Open Application Development) environment of ZEN (blue edition). With OAD you create your own macro solution based on the well-established Python language. Benefit from the simple access to a vital set of ZEN functions and the ability to include libraries such as the .Net Framework. Join the OAD community to discuss macros and help other users to develop their ultimate solution.

Instrument Control *(2 Viewing)* Discuss macros to control the hardware functions of your ZEISS microscope	Threads: 55 Posts: 201	SmartProof5 - Control... by Carl Zeiss Microscopy 3 10-24-2019, 08:59 AM
Image Acquisition *(1 Viewing)* Post your acquisition-related questions and macros here	Threads: 42 Posts: 113	SmartProof5 - Import Tiles... by Carl Zeiss Microscopy 3 10-24-2019, 09:19 AM
Image Handling *(1 Viewing)* Your place to discuss general handling of images and the CZI file format	Threads: 53 Posts: 146	Problems using czi image... by Fredrik 10-31-2019, 10:09 AM
Image Processing Discuss questions and projects related to processing of imaging data here	Threads: 56 Posts: 189	ZeissImageLib and .Net Core by MosGeo 10-31-2019, 01:59 PM
Measurement and Analysis *(1 Viewing)* Share your programming ideas for various measurement and analysis tasks here	Threads: 24 Posts: 66	Extract rotated rectangle... by Fredrik 10-22-2019, 03:54 PM

OAD Macro Development: Getting Help

Sample Macros on Github

https://github.com/zeiss-microscopy/OAD

Collection of tools and scripts useful to automate microscopy workflows in ZEN Blue using Python and Open Application Development tools.

python microscopy scripting automation imaging machine-learning open-source image-analysis zen zen-blue acquisition python-script workflow oad image-processing

145 commits | 1 branch | 0 releases | 2 contributors | GPL-3.0

Branch: master | New pull request | Find file | Clone or download

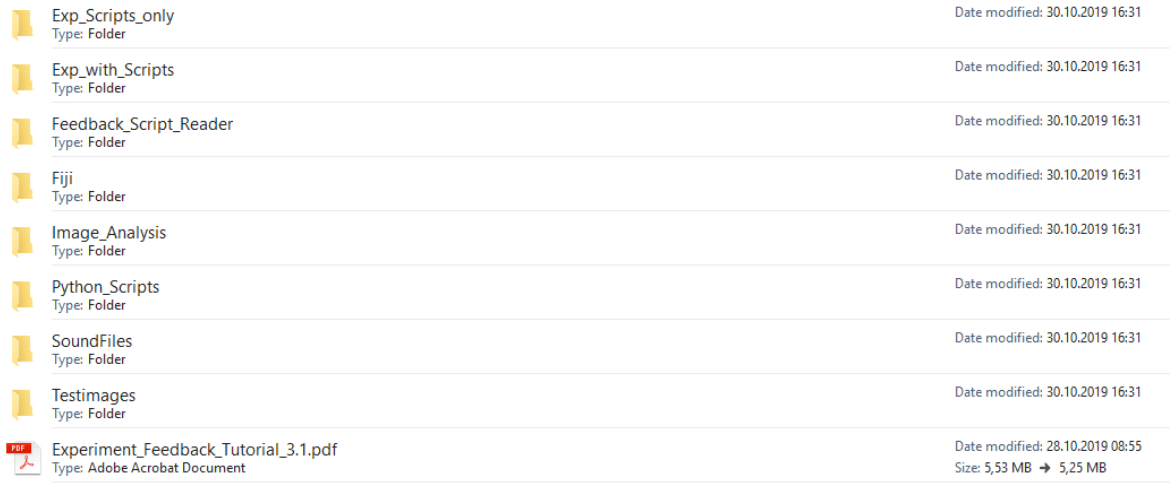
sebi06 updates — Latest commit 405a983 4 days ago

Apeer	updates	9 days ago
BioApps	added APEER fiji module template	5 months ago
Experiment_Feedback	updates	7 months ago
Guided_Acquisition	updates	9 days ago
Images	updates	9 days ago

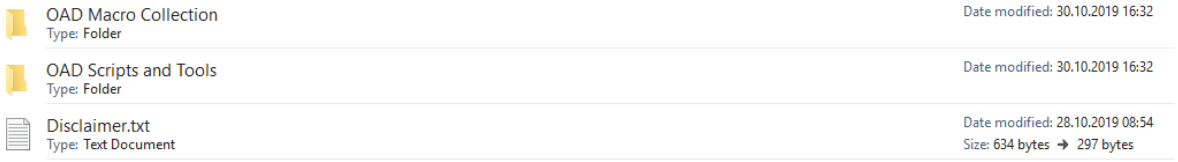
OAD Macro Development: Getting Help

Sample Macros on ZEN DVD

Manuals → ZEN (blue edition) → OAD content → Experiment Feedback

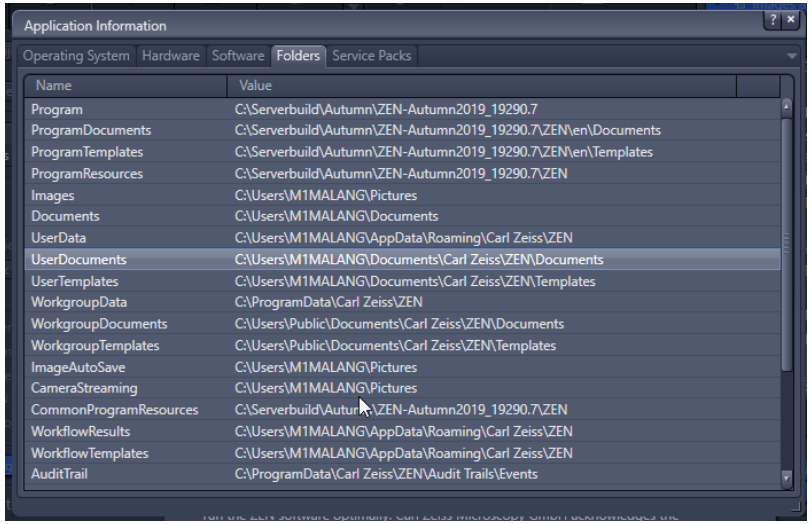


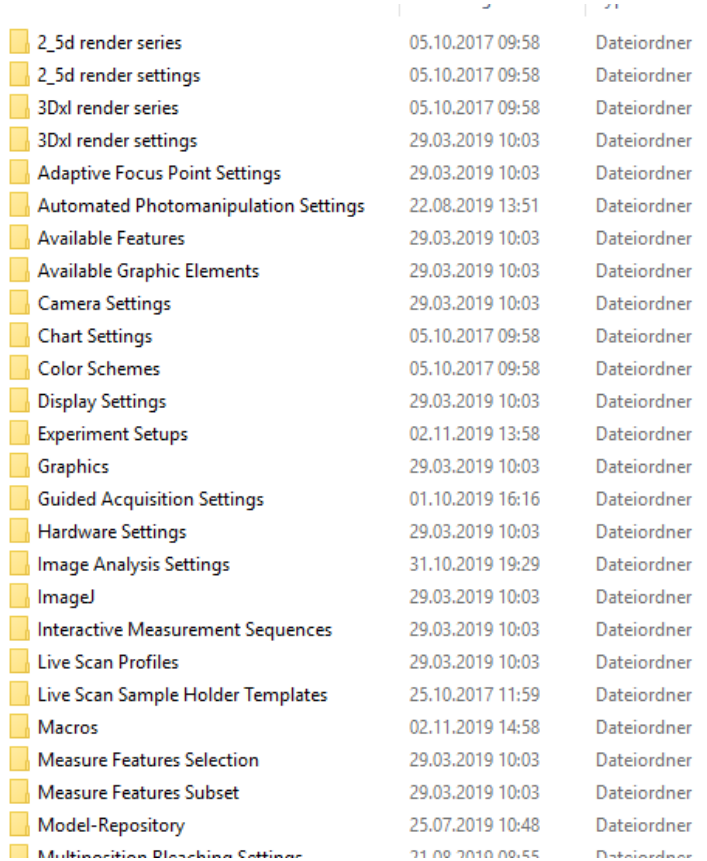
Other → OAD Examples



Finding ZEN Folders

Help → About ZEN → Show ZEN Information → Folders → Doubleclick e.g. on UserDocuments

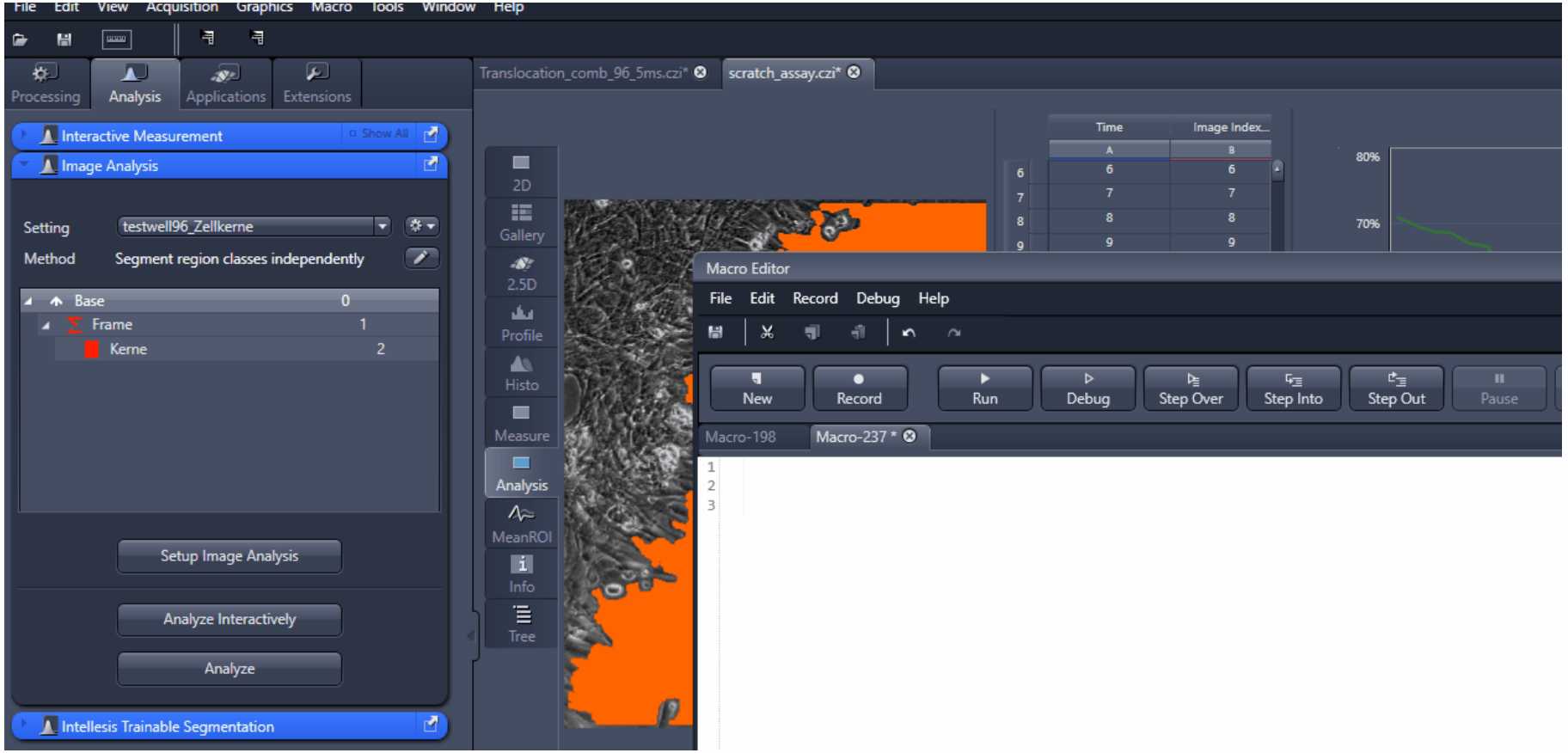




OAD Macro Development: Getting Help

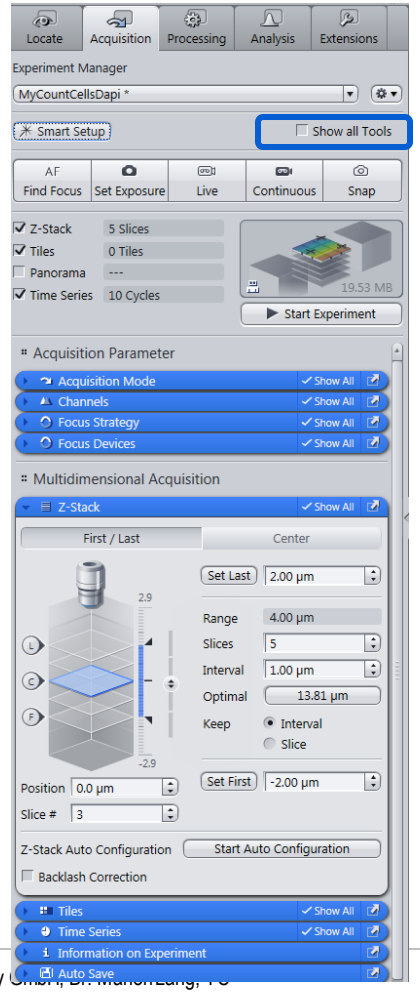
Macro recorder

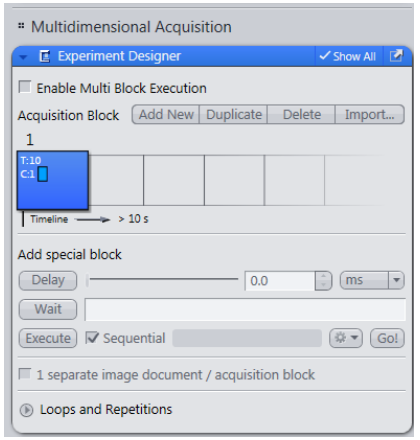
Note: Not everything can be recorded!
But that does not mean, there is not command for a specific functionality

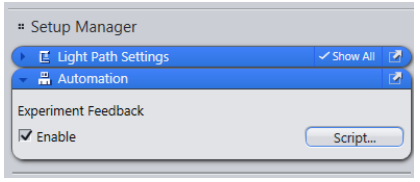


OAD Macro Development
Settings

Almost everything is done via settings. E.g. Experiment

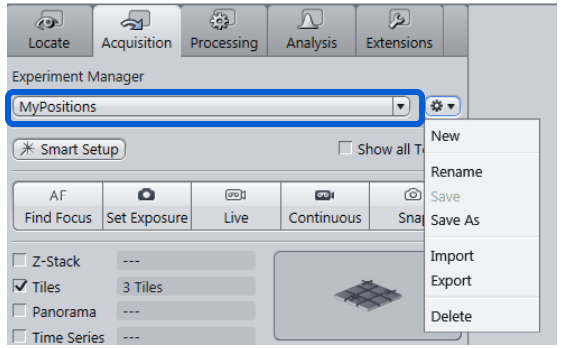






Acquisition
Experiment Manager

Setting

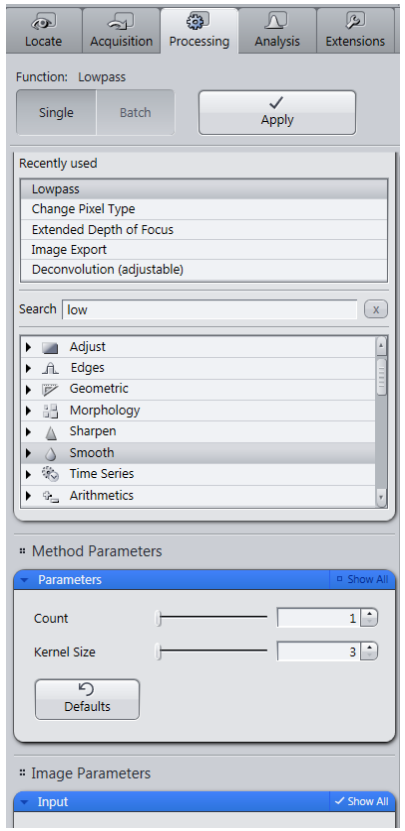


→ XML-File

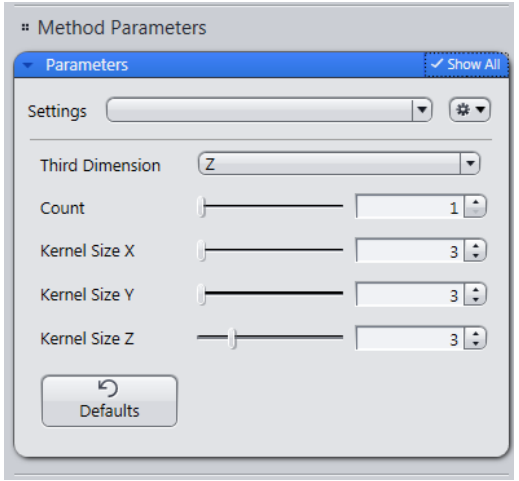
OAD Macro Development
Settings

Almost everything is done via settings. E.g. Processing

Processing



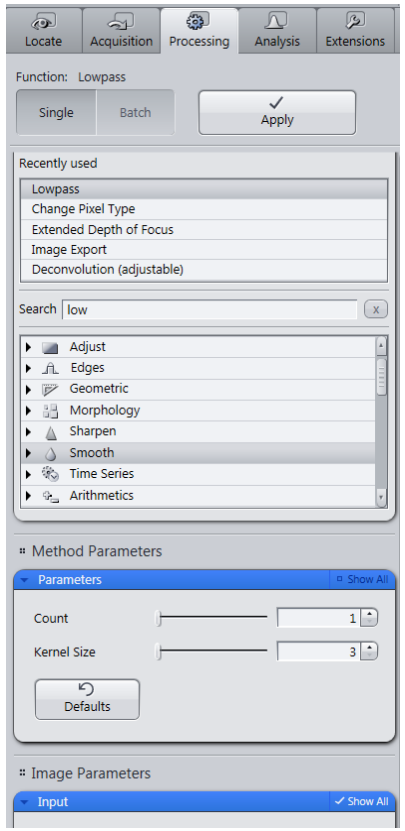
Setting



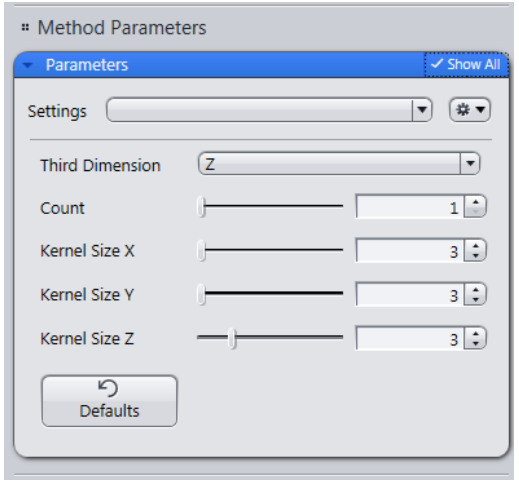
XML-File

OAD Macro Development
Settings

Almost everything is done via settings. E.g. Measurement



Setting



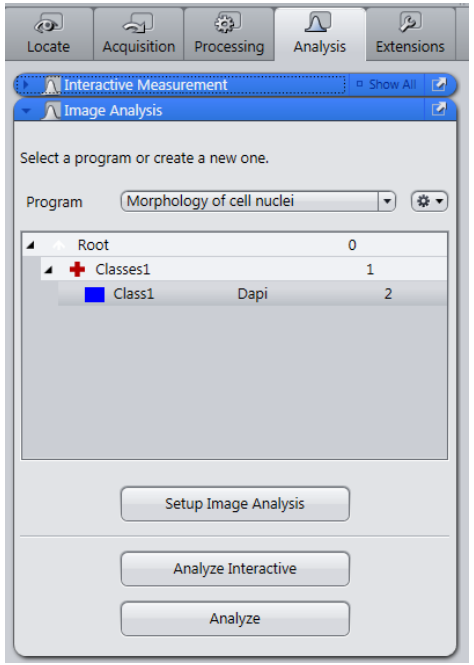
XML-File

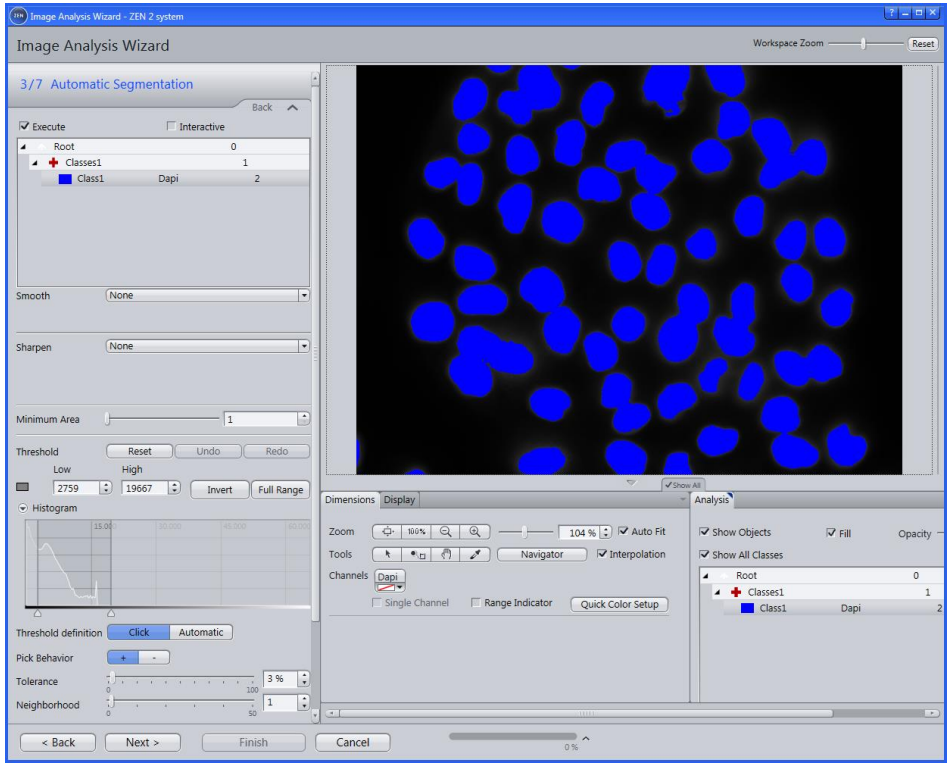
Processing

OAD Macro Development
Settings

Almost everything is done via settings. E.g. Image Analysis

Analysis





Setting → XML-File

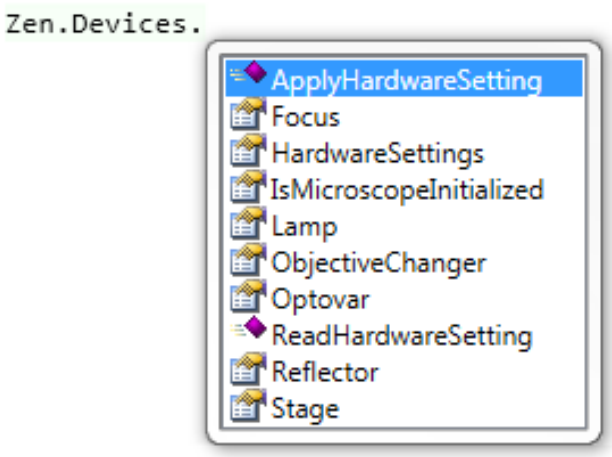
OAD Macro Development
Settings

Hardware:

`Zen.Devices.`

Lists all devices/methods that are directly accessible, e.g.

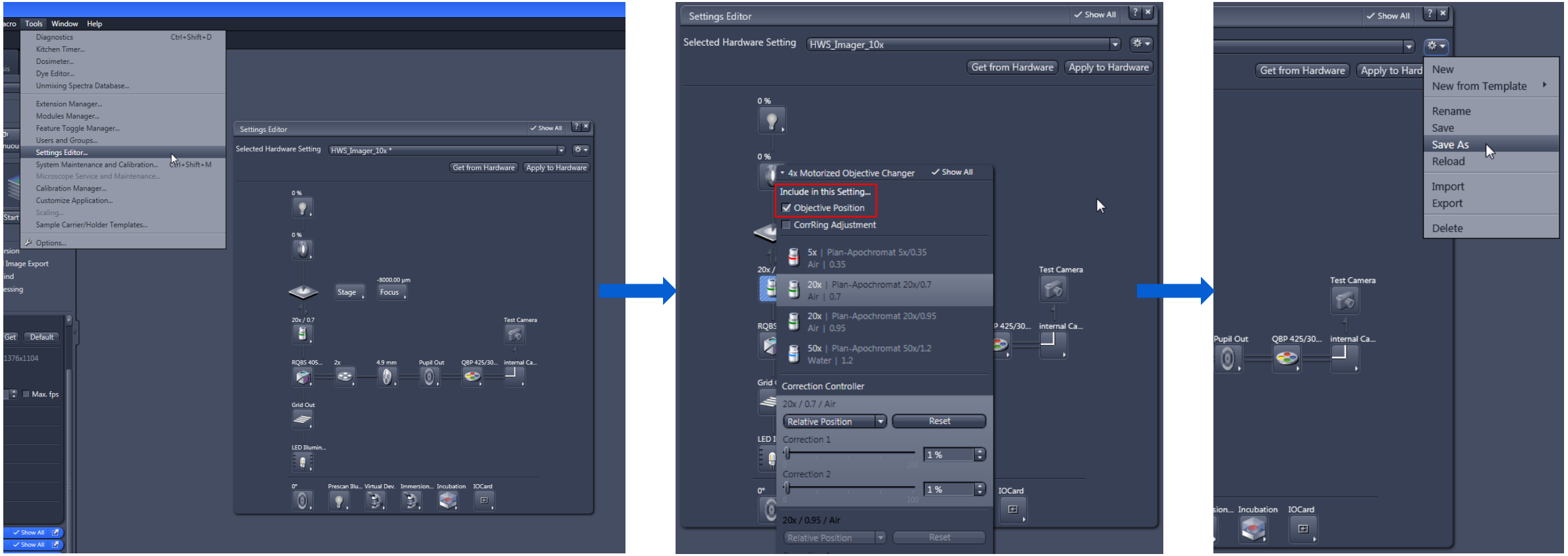
```
posX = Zen.Devices.Stage.ActualPositionX
mag = Zen.Devices.ObjectiveChanger.Magnification
```



OAD Macro Development

Settings

Hardware Setting:



Read & Save Hardware Setting

```
#Read hardware setting:
HWS = Zen.Devices.ReadHardwareSetting()

#Save hardware setting:
HWS.SaveAs(filename)
```

```
#Create an empty container:
HWS = ZenHardwareSetting()

#Load an existing Hardwaresetting:
HWS.Load('filename')
```

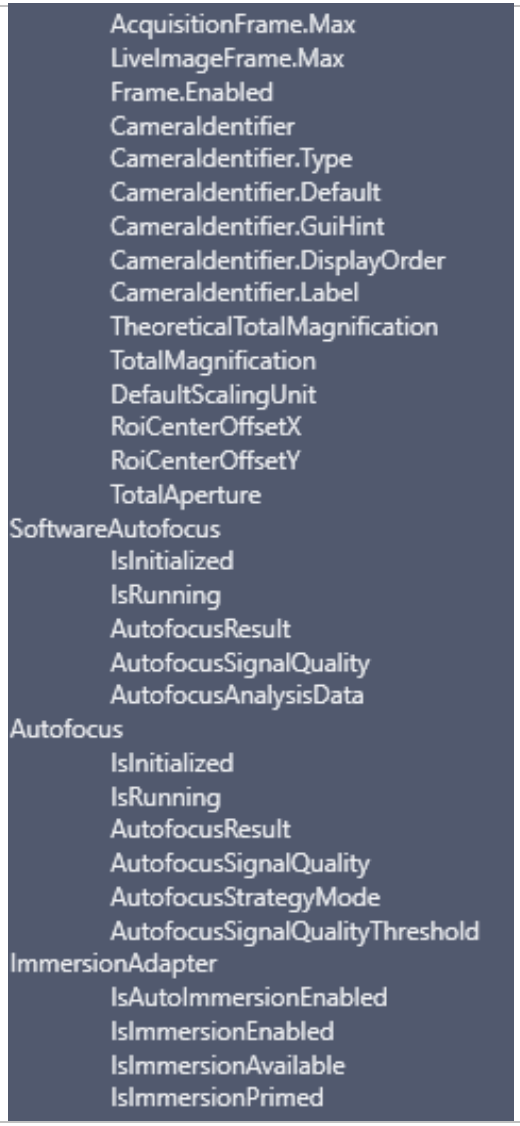
Default file path:

```
C:\Users\...\Documents\Carl
   Zeiss\ZEN\Documents\Hardware Settings
```

→ OAD_Training_Get HWS

Get Components from the Hardware Setting

```
# Get the IDs of every component in the Hardware Setting:
HWSIDs = HWS.GetAllComponentIds()

# go through all the Component IDs
for ID in HWSIDs:
    print(ID)

    # for each Component ID get the Parameter names
    HWSParams = HWS.GetAllParameterNames(ID)

    # go through all Parameters
    for HWSParam in HWSParams:
        print("\t" + HWSParam)
```



Set a parameter and apply

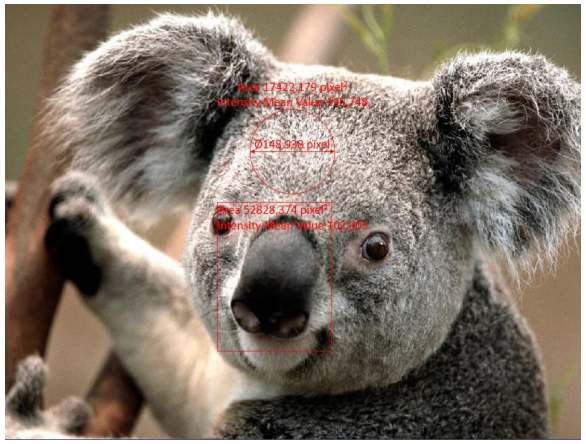
```
#Get a parameter:
stage_speed = HWS.GetParameter('componentId',
    'parameterName')


#Set a parameter:

HWS.SetParameter('componentId', 'parameterName',
    parameterValue)


#Apply the setting:

Zen.Devices.ApplyHardwareSetting(HWS)
```

```
#Apply a single value:
HWS = ZenHardwareSetting()
HWS.SetParameter( 'set parameters')
HWS.SetActive()
HWS.SaveAs('filename')
```

OAD
Image Processing

- Start macro recorder.
- Load an Image
- Use IP „Threshold“

```
# *************** Recorded Code Block ***************
image1 = Zen.Application.LoadImage("C:\\testdata\\Koala_Graphics.czi", True)
Zen.Application.Documents.Add(image1)
image2 = Zen.Processing.Segmentation.Threshold(image1, 53, 181, 198, 127, 0,
199, True, False, False, True)
Zen.Application.Documents.Add(image2)
# *************** End of Code Block ******************
```

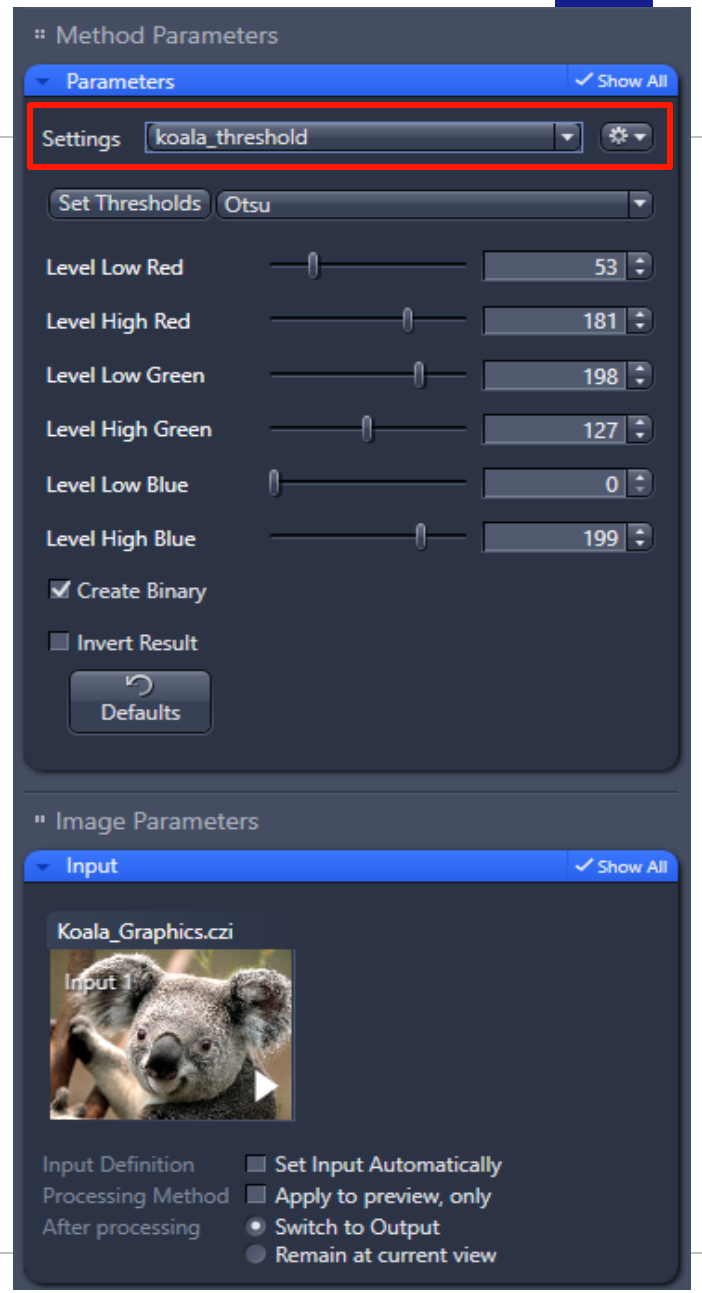


OAD

Image Processing with Setting

- Create a setting for the IP „Threshold“
- Apply the IP with threshold to the image
- Add the image to the document collection

```
# Get the image
image1 = Zen.Application.Documents.GetByName("Koala_Graphics.czi")
# Create a setting object
set = Zen.Processing.Segmentation.Settings.ThresholdSetting()
# Load the setting
set.Load("C:\Users\M1MALANG\Documents\Carl
Zeiss\ZEN\Documents\Processing
Settings\Threshold\koala_threshold.czips")
# Run IP
image2 = Zen.Processing.Segmentation.Threshold(image1, set, True)
# Add result to image documents
Zen.Application.Documents.Add(image2)
```

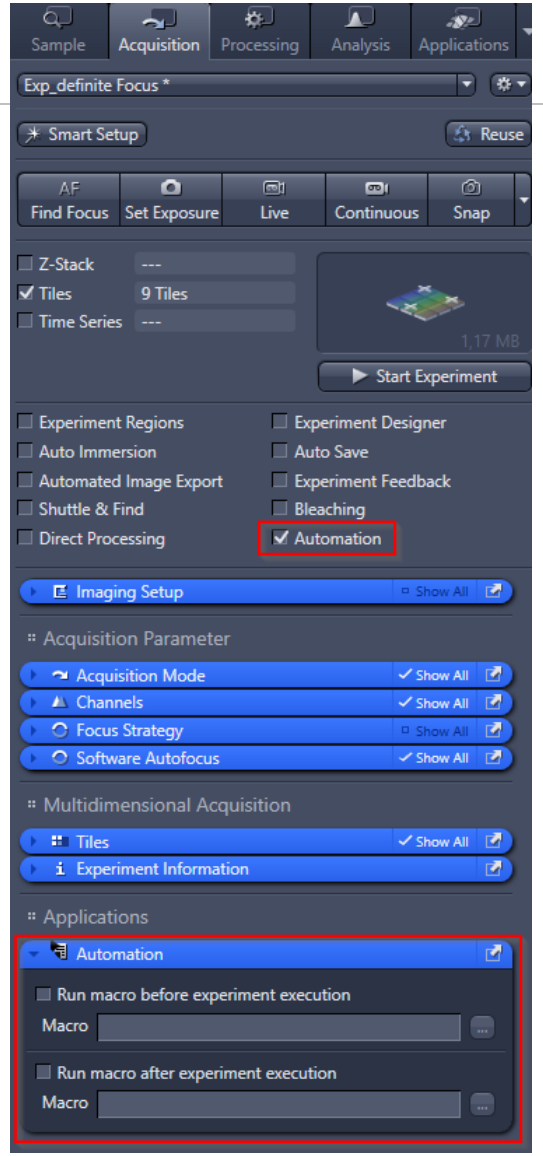


OAD
Automation

Allows to run a macro before or after experiment execution
Will be part of the experiment xml.

- Create a macro, e.g. „invert“
- Run it automatically at the end of the experiment

```
# invert current image
image1 = Zen.Application.ActiveDocument
image2 = Zen.Processing.Binary.Invert(image1)
Zen.Application.Documents.Add(image2)
image1.Close()
```



OAD
Customize Application

Tools → Customize Application

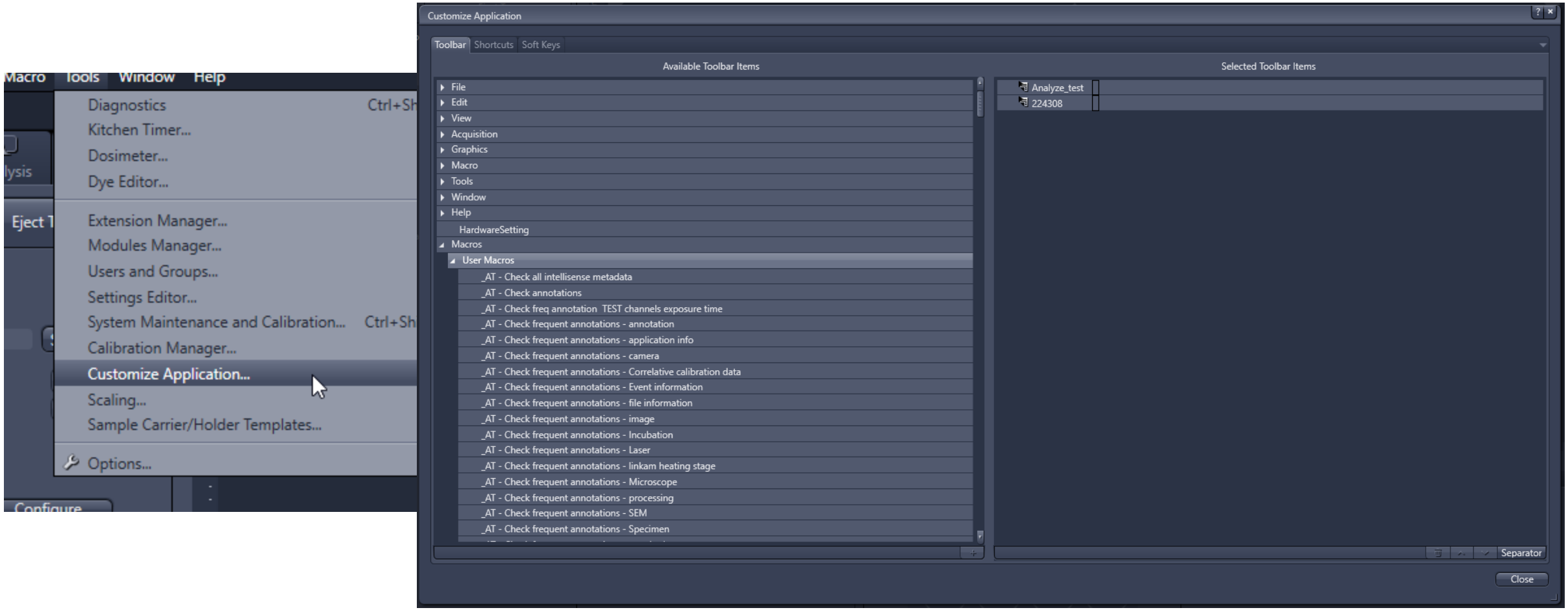


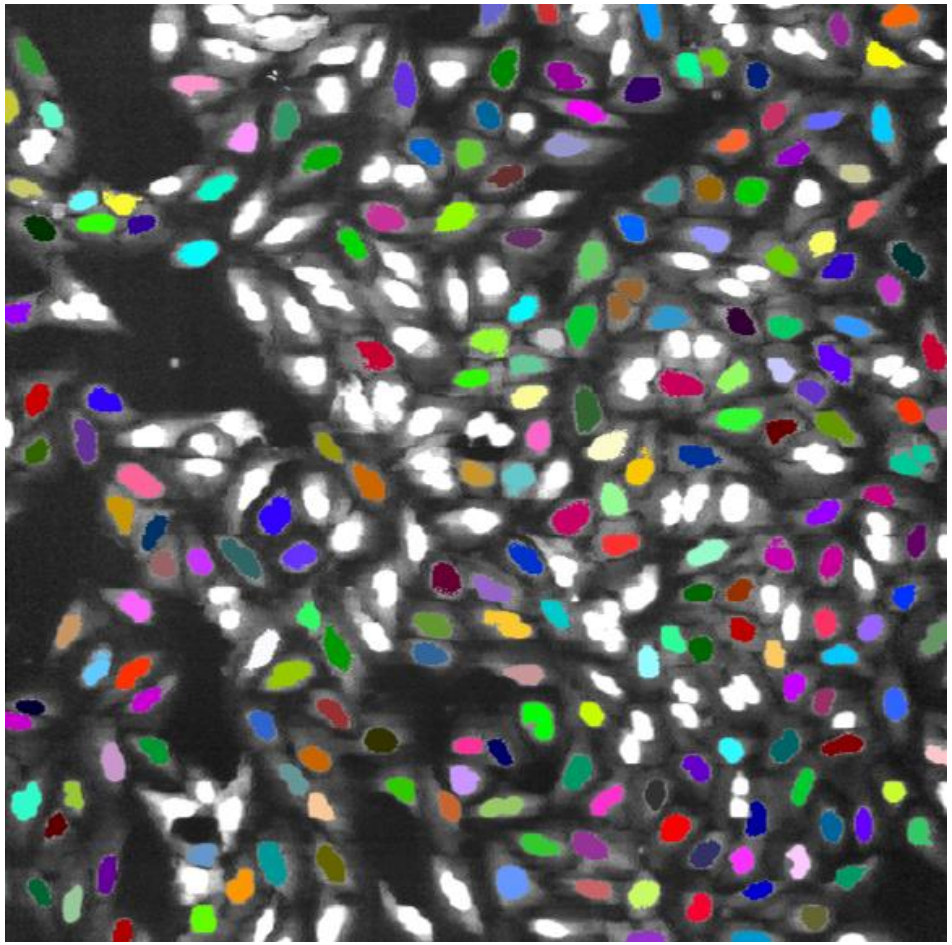
Image Analysis with OAD
Run an image analysis, generate tables

```
img = Zen.Application.ActiveDocument
ias = ZenImageAnalysisSetting()
ias.Load("cell_count")


# run image analysis
img = Zen.Analyzing.Analyze(img, ias)


# create result table
table = Zen.Analyzing.CreateRegionTable(img)
table.Save("C:\\temp\\cell_count_region.csv")
Zen.Application.Documents.Add(table)
```

„Analyze“ stores the results in the czi file

Image Analysis with OAD

```
img = Zen.Application.ActiveDocument
ias = ZenImageAnalysisSetting()
ias.Load("cell_count")

#create labled images
img_out = ZenImage()
Zen.Analyzing.AnalyzeToImage(
img, img_out, ias,
ZenAnalyzerLabelImageMode.RegionUniqueColor,
ZenAnalyzerLabelImageInitMode.CopyAllChannels,
ZenPixelType.Bgr24)
Zen.Application.Documents.Add(img_out)
```



Start External Programs

Start Fiji and load result in ZEN

```
img = ZenImage()
img.Load("C:\\Training\\CZI_DimorderTZC.czi")

filename = img.FileName
exeloc = 'C:\Fiji_Sebi\ImageJ-win64.exe'

# specify the Fiji macro that will be appplied
macro = r'-macro C:\Training\Open_CZI_and_MaxInt_Save.ijm'

# define parameters for the Fiji macro
params =  macro + ' ' + filename

# start Fiji, open the data set and execute the macro
app = Process();
app.StartInfo.FileName = exeloc
app.StartInfo.Arguments = params
app.Start()
app.WaitForExit()
```

→ OAD_Training_Fiji

Start External Programs

Start Fiji and load result in ZEN

ImageJ Macro

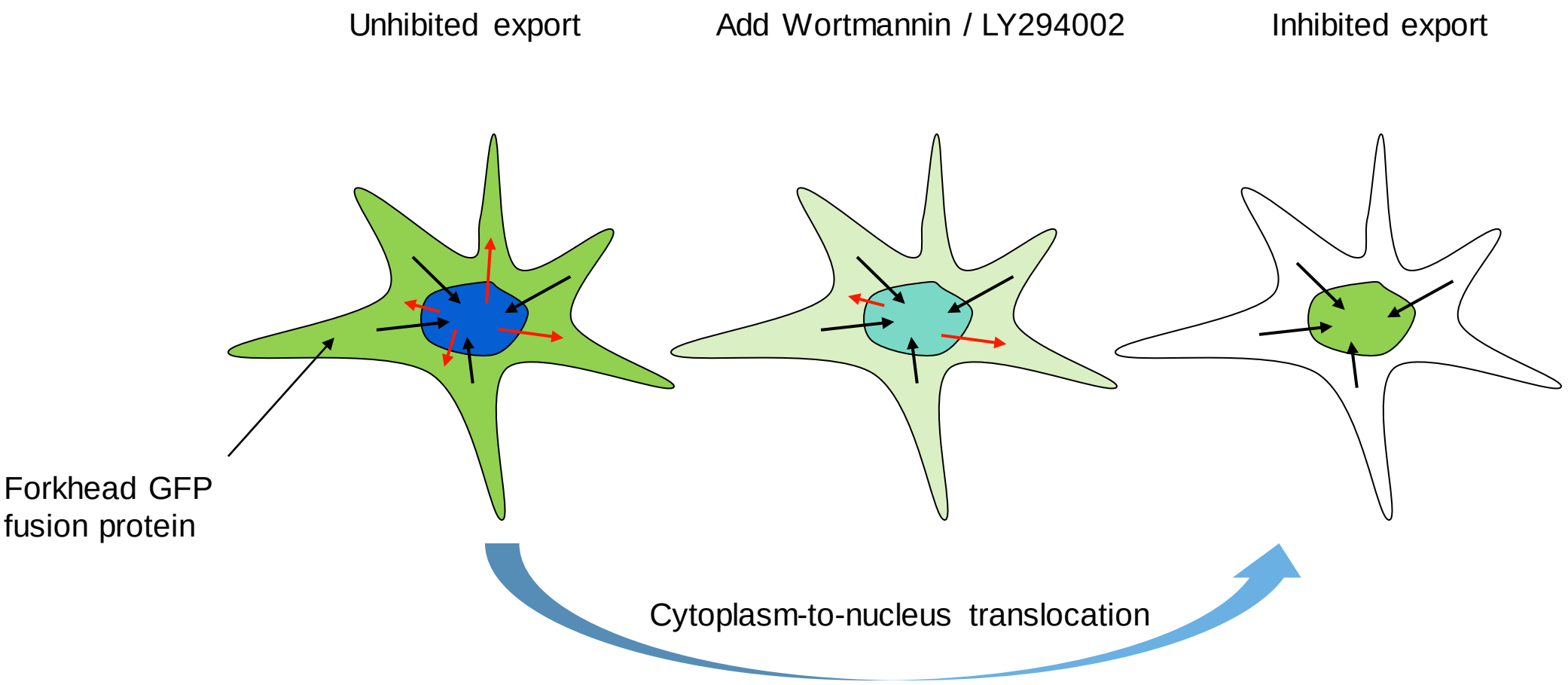
Open_CZI_and_MaxInt_Save.ijm

```
name = getArgument;
if (name=="") exit("No argument!");

run("Bio-Formats Importer", "open=[" + name + "] autoscale color_mode=Default open_all_series view=Hyperstack stack_order=XYCZT");
// get dimension from original image
getDimensions(width,height,channels,slices,frames);
// apply additional operation
run("Z Project...", "start=1 stop=" + slices + " projection=[Max Intensity]"); // do maximum intensity projection
run("Fire"); // apply special LUT
saveAs("PNG",  "C:/Training/fiji.png");
run("Quit");
```

```
savename =  "C:\\Training\\fiji.png"

if File.Exists(savename):
    fiji_result = Zen.Application.LoadImage(savename, False)
    Zen.Application.Documents.Add(fiji_result)
else:
    print 'Saved figure not found.'
```

Application Example: Translocation

Inhibition of nuclear export of Forkhead



OAD Script for automatization

Start Python Script for Calculation and Plotting

Tasks to perform:

1. Load load the image file (*.csv) and image anlaysis setting (*.czias)
2. Run the image analysis
3. Extract the image analysis results as *.csv
4. Start the python script (test_wellplate_from_ZEN.PY)

5. Read in data
6. Calculate the translocation Ratio
7. Generate heatmaps for different features (e.g. Translocation Ratio)
8. Save heatmaps as PNG files

9. Load PNG files in ZEN

Testdata: Translocation_comb_96_5ms.czi
Image Analysis Setting: Translocation_26.czias

OAD Script

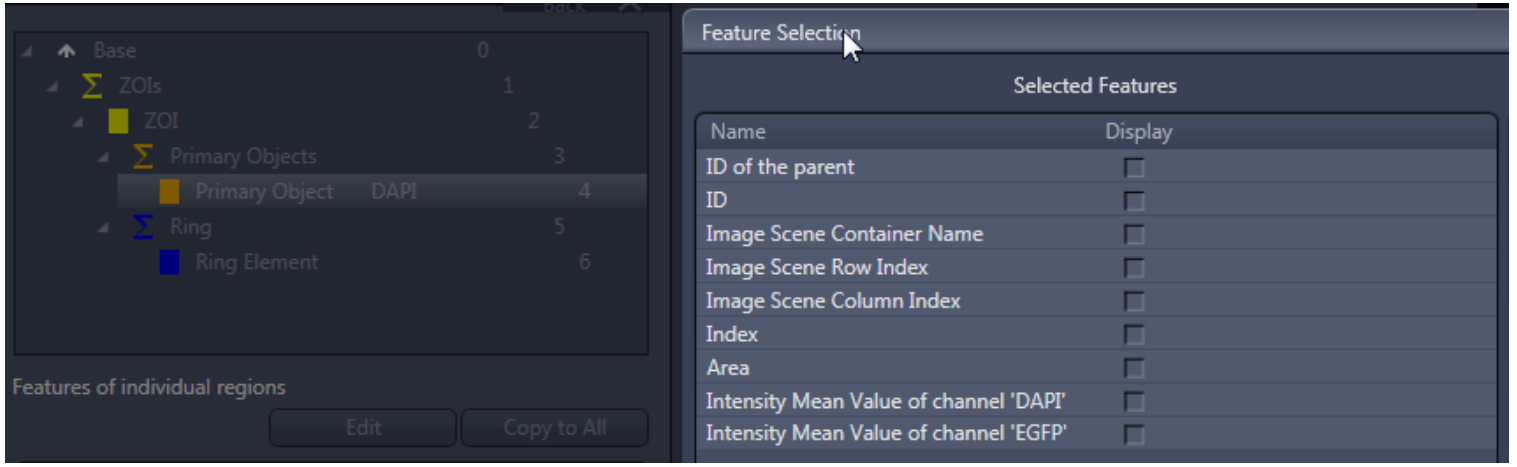
→ Translocation Plot Pandas

```
from System.Diagnostics import Process
from System.IO import File, Path, Directory
import time

# define the external plot script or tool
pythonexe = r'C:\Program Files\Carl Zeiss\ZeissPython\Py20190211\env\python.exe' # Zeisspython
# requires progressbar2 libary: "C:\Program Files\Carl Zeiss\ZeissPython\Py20190211\env\python.exe" -m pip install progressbar2
script = r'C:\...\test_wellplate_from_ZEN.PY'


# load image and add it to ZEN
image_to_analyze = r'C:\...\Translocation_comb_96_5ms.czi'
image = Zen.Application.LoadImage(image_to_analyze)
Zen.Application.Documents.Add(image)

# get the image path
outputpath = Path.GetDirectoryName(image_to_analyze)
resultname = Path.GetFileNameWithoutExtension(image.Name)
```

OAD Script: Run Image Analysis

Features defined in the image analysis setting:
ID parent | ID | Image Scene container Name | Image Scene Row | Image Scene Column | Index # | Area | NucMeanDapi | NucMeanGFP | RingMeanGFP | RingArea

```
# Load image analysis setting and perform image anlaysis
iasfilename = r'C:\...\Image Analysis Settings\Translocation_2.czias'
ias = ZenImageAnalysisSetting()
ias.Load(iasfilename)
Zen.Analyzing.Analyze(image,ias)
```



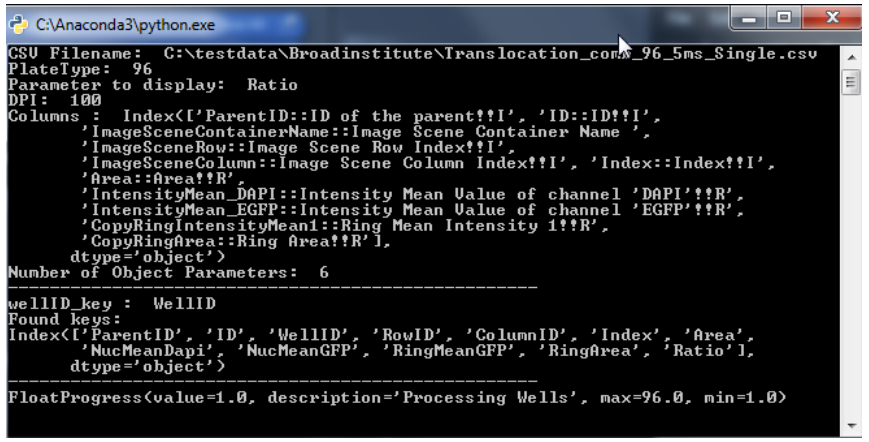
OAD Script: Write CSV File

```
# For ZOI-Image Analysis Settings need to get the results for the Primary Objects
# Create data list with results for each primary object
table_single = Zen.Analyzing.CreateRegionTable(image, "Primary Object")
#Zen.Application.Documents.Add(table_single)

# Save both data list as CSV files
table_single_filename = Path.Combine(outputpath,  resultname + '_Single.csv')
table_single.Save(table_single_filename)

# close the image and image analysis setting
image.Close()
ias.Close()
```

	A	B	C	D	E	F	G	H	I	J	K	L
1	ParentID::ID	ID::ID!!I		ImageScene	ImageScene	ImageScene	Index::Index	Area::Area!!	IntensityMe	IntensityMe	CopyRingInt	CopyRingArea::Ring Area!!R
2								µm²	Gray	Gray	Gray	µm²
3	269	270	A1	1	1	1	2.64	133.337121	88.0530303	141.167832	1.43	
4	271	272	A1	1	1	2	1.84	79.2880435	0	0.02962963	1.35	
5	273	274	A1	1	1	3	3.04	168.177632	9.81907895	9.22058824	1.36	
6	275	276	A1	1	1	4	2.81	106.231317	38.7935943	46.6071429	1.96	
7	277	278	A1	1	1	5	2.47	146.546559	52.7773279	65.0347222	1.44	
8	279	280	A1	1	1	6	2.01	96.6865672	33.4079602	68.3777778	1.35	
9	281	282	A1	1	1	7	1.58	94.3734177	18.7468354	28.1842105	1.14	
10	283	284	A1	1	1	8	1.81	87.6574586	31.0607735	53.5683453	1.39	
11	285	286	A1	1	1	9	2.25	114.826667	0	0	1.36	
12	287	288	A1	1	1	10	2.72	95.4669118	0.11397059	0.625	1.44	
13	289	290	A1	1	1	11	1.95	96.4564103	13.2974359	22.0735294	1.36	
14	291	292	A1	1	1	12	2.23	104.753363	0.07623318	0.10218978	1.37	

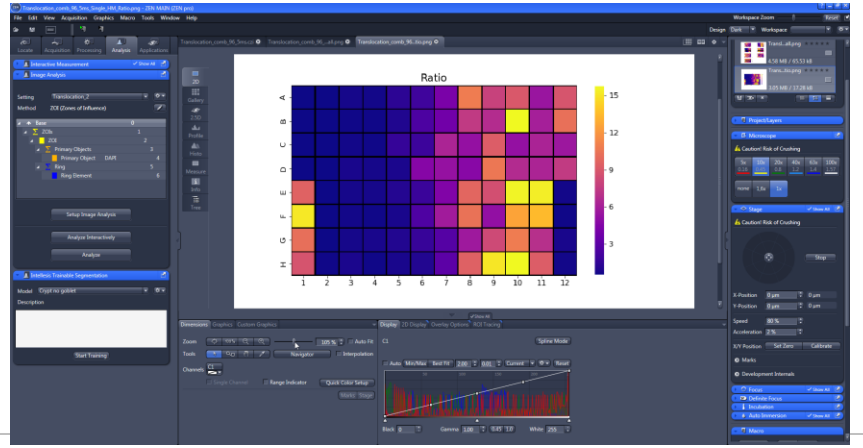
OAD Script: Start Python with Parameters

```
# define the actual CSV file and the parameters
csvfile = Path.Combine(outputpath, table_single_filename)

# this depends on the actual CZIAS and the import of the CSV table in python
#parameter2display = 'CellCount'
parameter2display = 'Ratio'
params = ' -f ' + csvfile + ' -w 96' + ' -p ' + parameter2display + ' -sp False -dpi 100 -xlsx
True'

# start the data display script as an external application
Process.Start(script, params)
```



OAD Script: Load PNG Files

```
# define filenames of PNG files
savename_all =  Path.Combine(Path.GetDirectoryName(image_to_analyze),
Path.GetFileNameWithoutExtension(image_to_analyze) + '_Single_HM_all.png')
savename_single = Path.Combine(Path.GetDirectoryName(image_to_analyze),
Path.GetFileNameWithoutExtension(image_to_analyze) + '_Single_HM_' + parameter2display + '.png')
print 'Showing saved figure in ZEN.'

# Check if fileneame exists, and Load images in ZEN
if File.Exists(savename_all):
    plotfigure1 = Zen.Application.LoadImage(savename_all, False)
    plotfigure2 = Zen.Application.LoadImage(savename_single, False)
    Zen.Application.Documents.Add(plotfigure1)
    Zen.Application.Documents.Add(plotfigure2)
else:
    print 'Saved figure not found.'

print 'Done.'
```



Import from your own libraries

macrolib_2.py

```
def greet_user(username):
    print("Hello, " + username + "!")
```

Macro:

```
import macrolib_2
Zen.Application.MacroEditor.ClearMessages()

macrolib_2.greet_user('Marion')
```


ZEISS