Open Application Development (OAD)
Tools for Smart Microscopy

ZEN is only part of the workflow

ZEN Connect

OAD

Python

Excel

Scripting

MATLAB

ZEN blue

KNIME

Analysis

Fiji

Automation

Machine Learning

Extensions

Simplify

BioFormats

OAD – General Concept and Key Features

- **Open Application Development** (OAD) uses powerful **Python Scripts** to **simplify**, **customize** and **automate** your workflows.

- **Analyze** and **Exchange** data with applications like **Fiji, Python, Knime, CellProfiler, Icy, MATLAB, Excel** and …

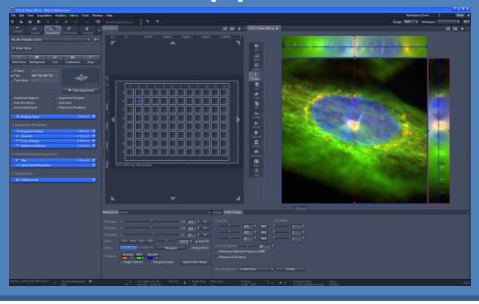
- The **CZI-API for .NET** (ZeissImgLib) and for **C++** (libCZI) and **BioFormats** (CZIReader) allow easy access to CZI files from many external applications. (OME-TIFF Export and Import in ZEN Blue is possible)

- **BioFormats Import** as a module inside ZEN Blue

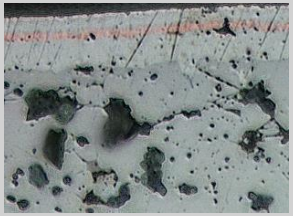
- Create "**smart**" experiments with **Experiment. Feedback** and modify the acquisition **On-the-fly** based on **Online Image Analysis** and **External Inputs**

OAD – ZEN Interfaces

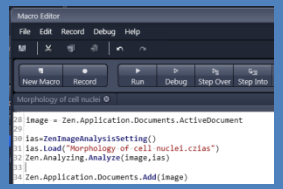
ZEN Application



CZI-Image



IronPython



Simple Interface

COM

TCP/IP

Interfaces

Acquisition

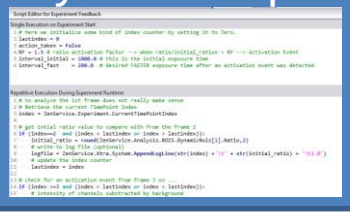
Processing

Measurement

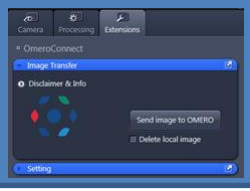
Analysis

ZEN
Acq. Server

Exp. Feedback
Python Scripts



Extensions



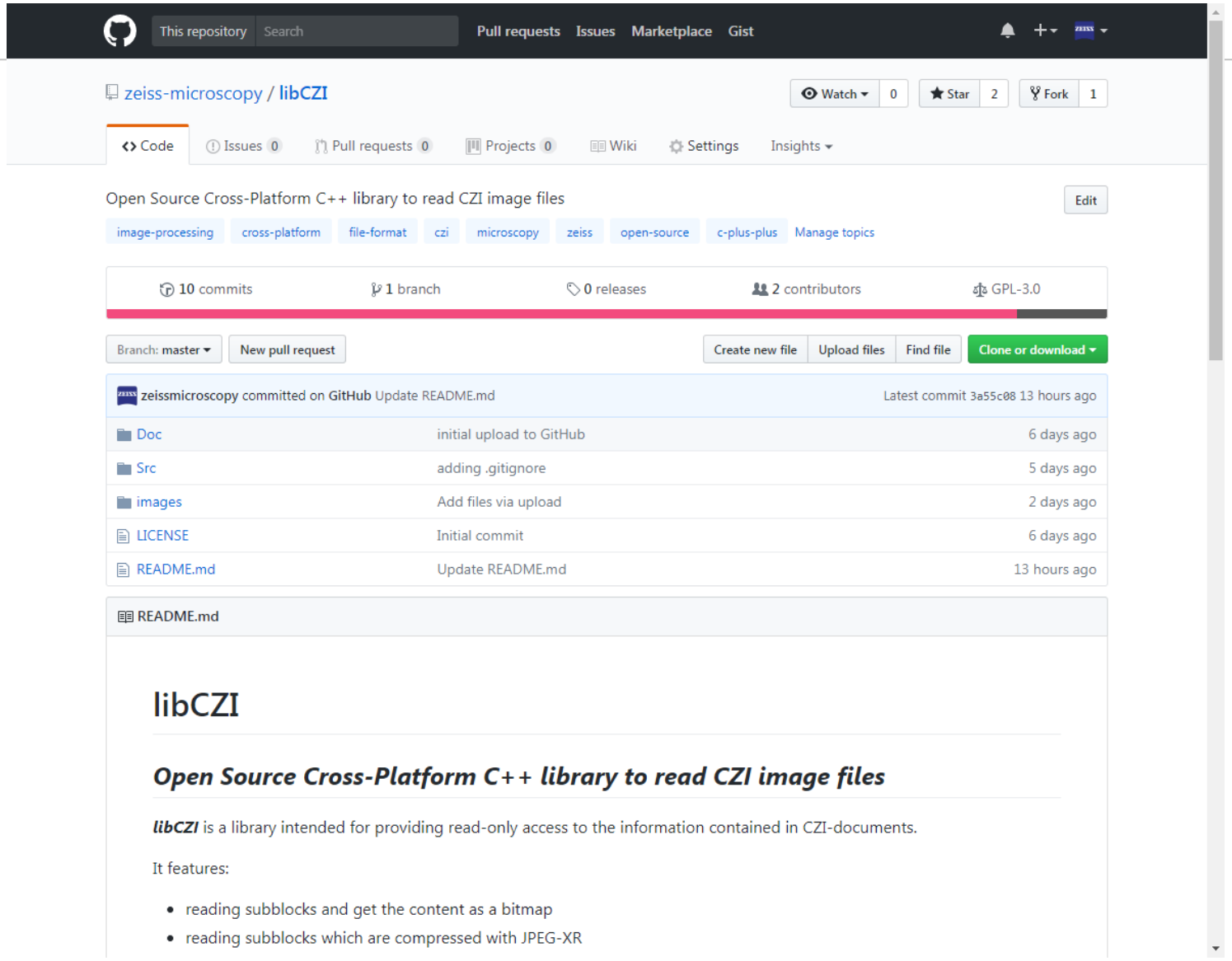
Carl Zeiss Image (CZI) File Format

- **CZI = Unified File Format for all ZEISS Light Microscopy Systems**
- Allows to JPG-XR compression **with lossless option**
- **XML structure of Metadata** developed with “an eye” on OME Meta data scheme (to facilitate compatibility)
- **Open up access** to your imaging **data** acquired with ZEISS Microscopes
- **CZIReader as part of OME-BioFormats-Reader** → ImageJ/Fiji, OMERO, KNIME, Python, CellProfiler, MATLAB, Icy etc.
- API: **ZeissImageLib** (.NET) and **libCZI** (C++) for commercial & non-commercial software

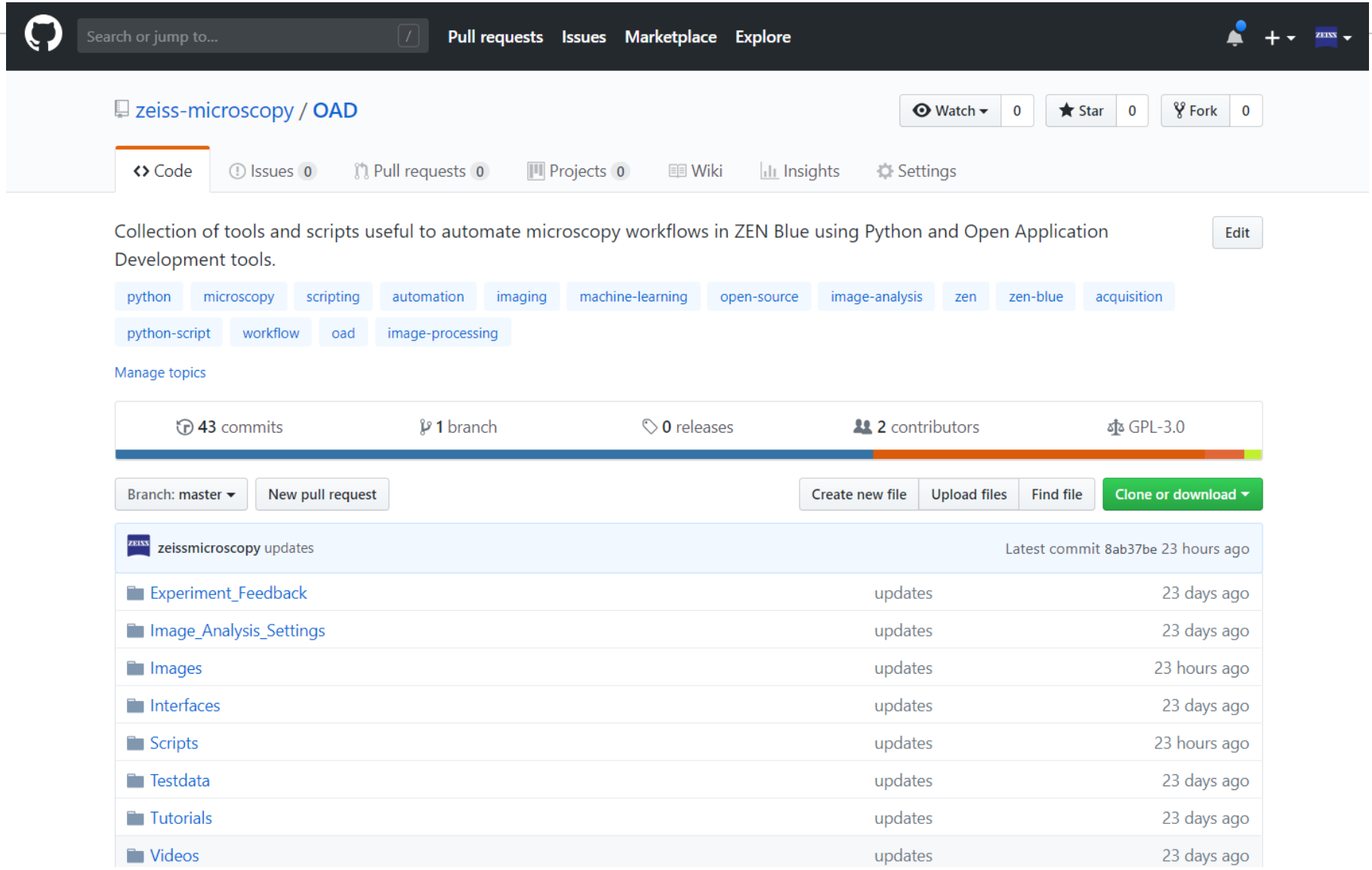
http://www.zeiss.com/czi

http://github.com/zeiss-microscopy/libCZI

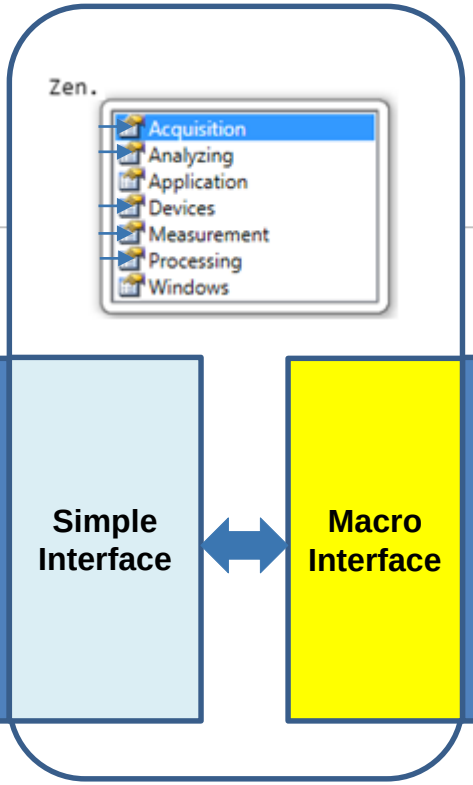
libCZI – open-source crossplatform C++ library



OAD – Examples and Script are available on GitHub

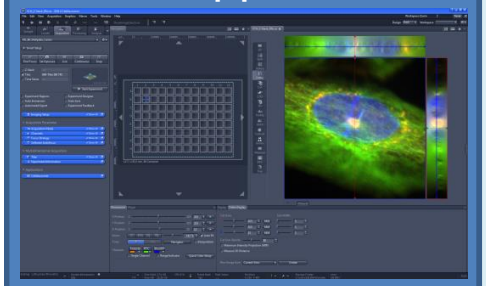


Macro Interface



IronPython

ZEN Application



Simple Interface

Macro Interface

Python IDE

- Recorder
- Editor
- Debugger

Start External Apps or Scripts

Exchange Data

.Net based (managed code)

import

Basic Libraries

IronPython Standard Library

Your Choice

.Net Framework

Third Party Customized Solution

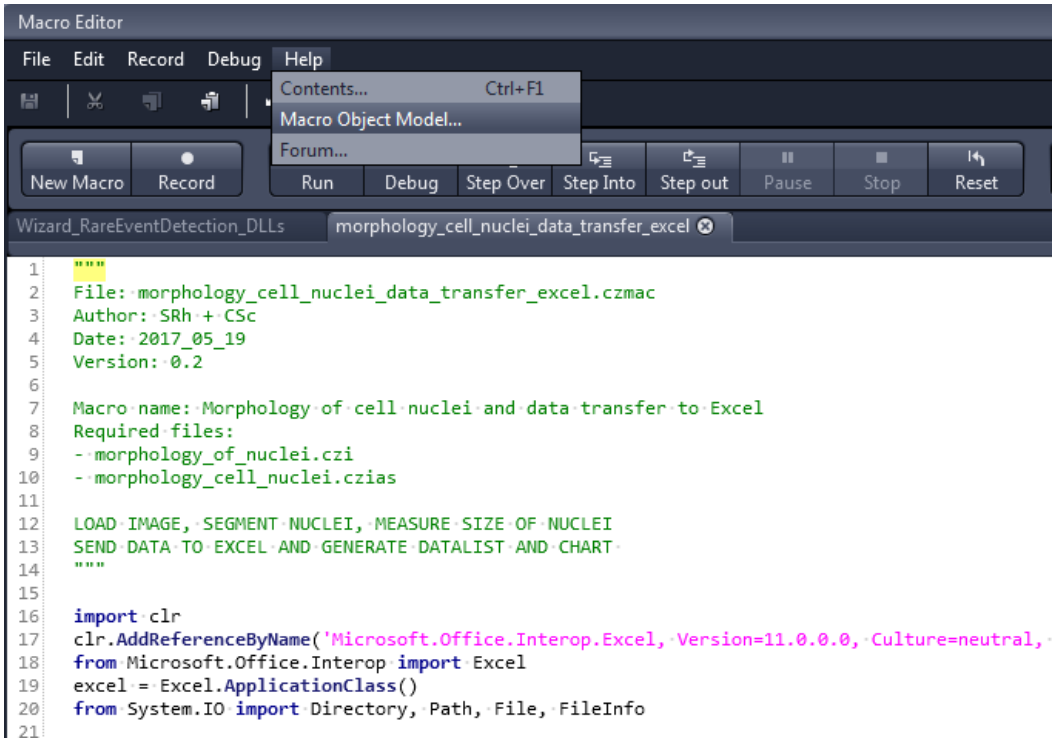
Other Python Libraries*

Other .Net DLL

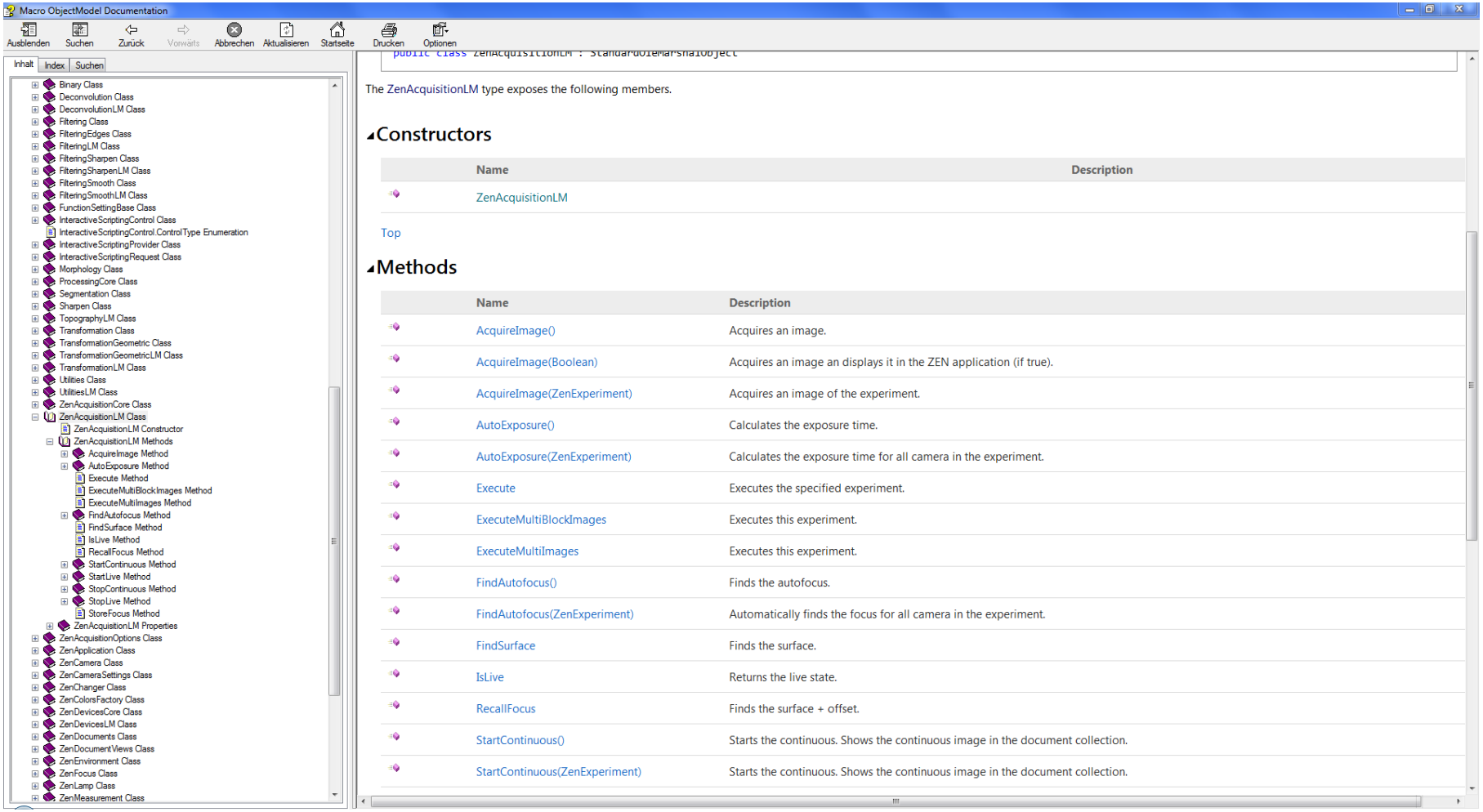
* With support of IronPython

Macro Interface – General Remarks

- ZEN uses IronPython as its internal scripting language → this allows using ZEN and .NET functionality directly, but not SciPy, NumPy etc.
- Not everything can be recorded, but that does not mean, there is not command for a specific functionality
- script editor has an built-in help file
- there is a OAD forum
- Example and Scripts can be found on the DVD

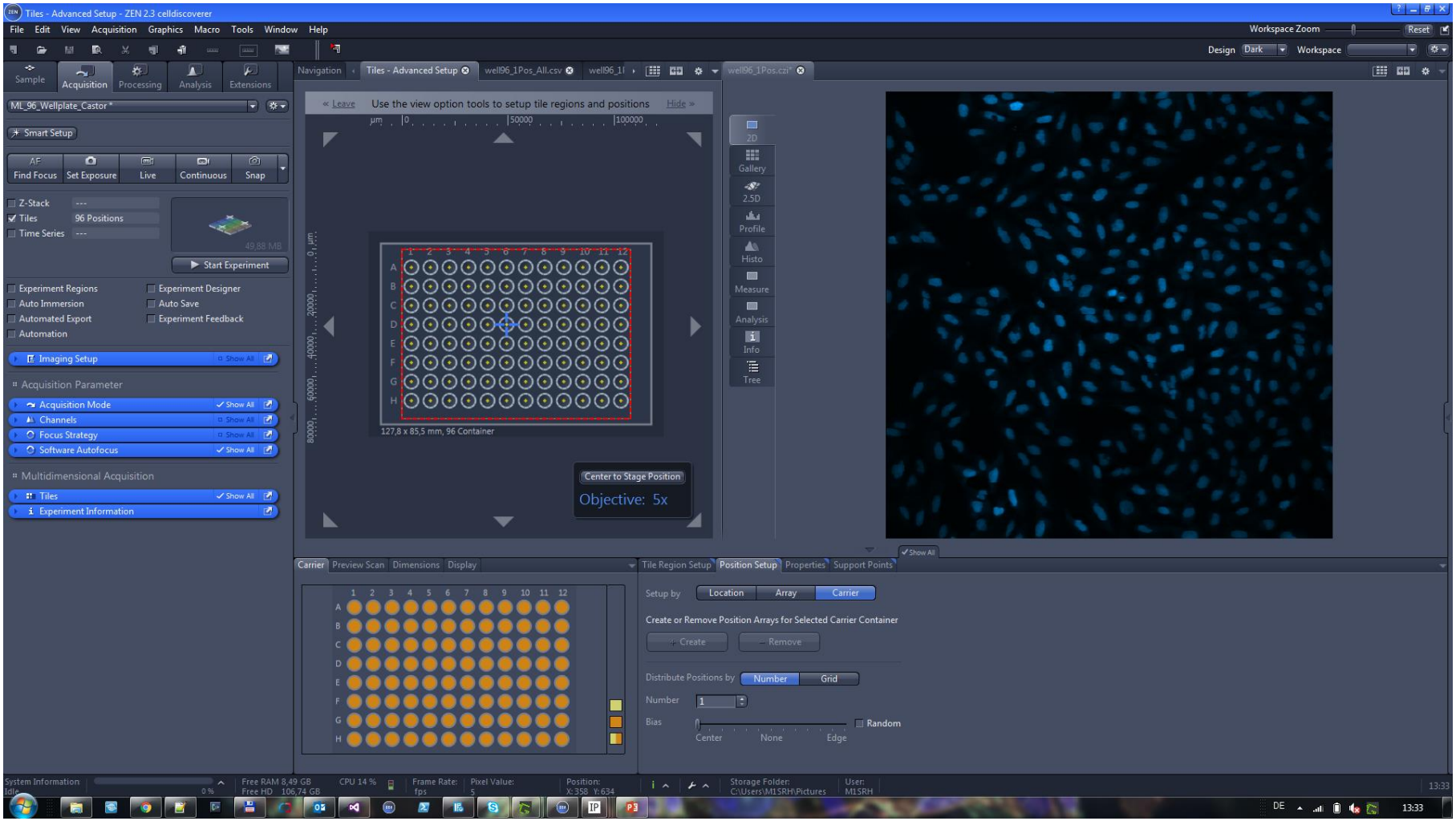


Macro Interface – Object Model Documentation

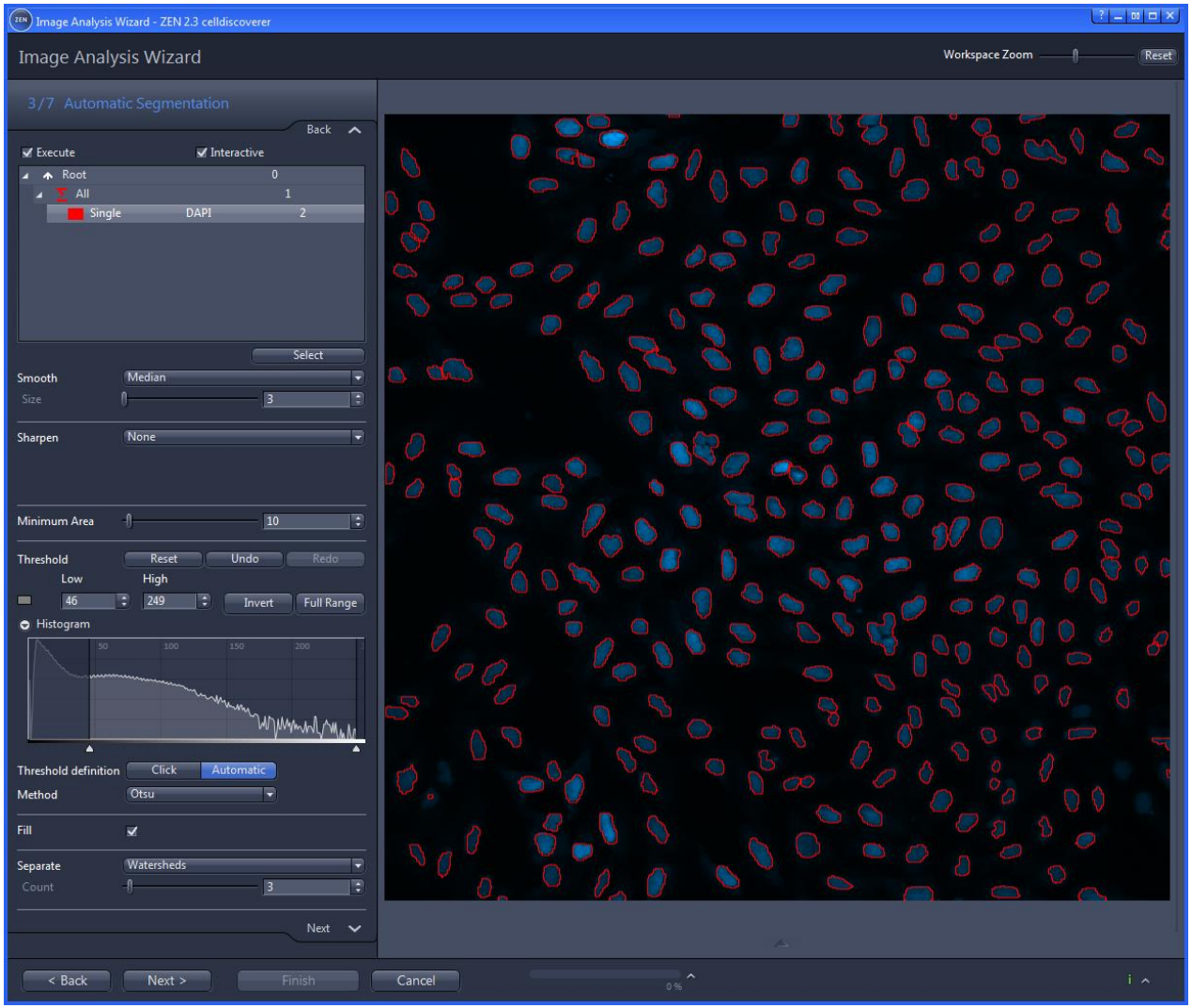


… yes, it is still an “old-school” CHM-file

Scripting - Automated Image Acquisition



Scripting - Automated Image Analysis



- Automated image analysis is built-in.
- Online Image Analysis is using the same pipelines.
- ZEN can do “hierarchical” measurements.
- **But where required OAD allows integration of (your own) external image analysis.**

Scripting - Access to Image Analysis Parameters

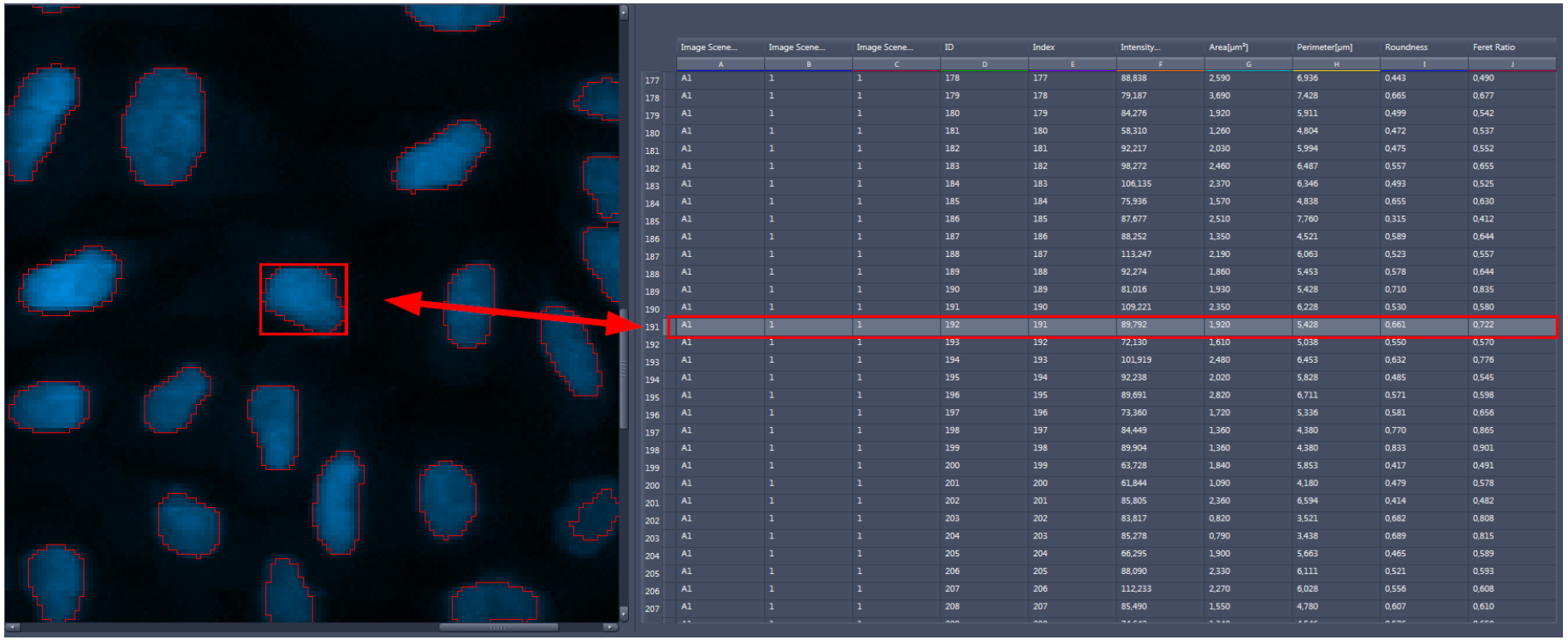
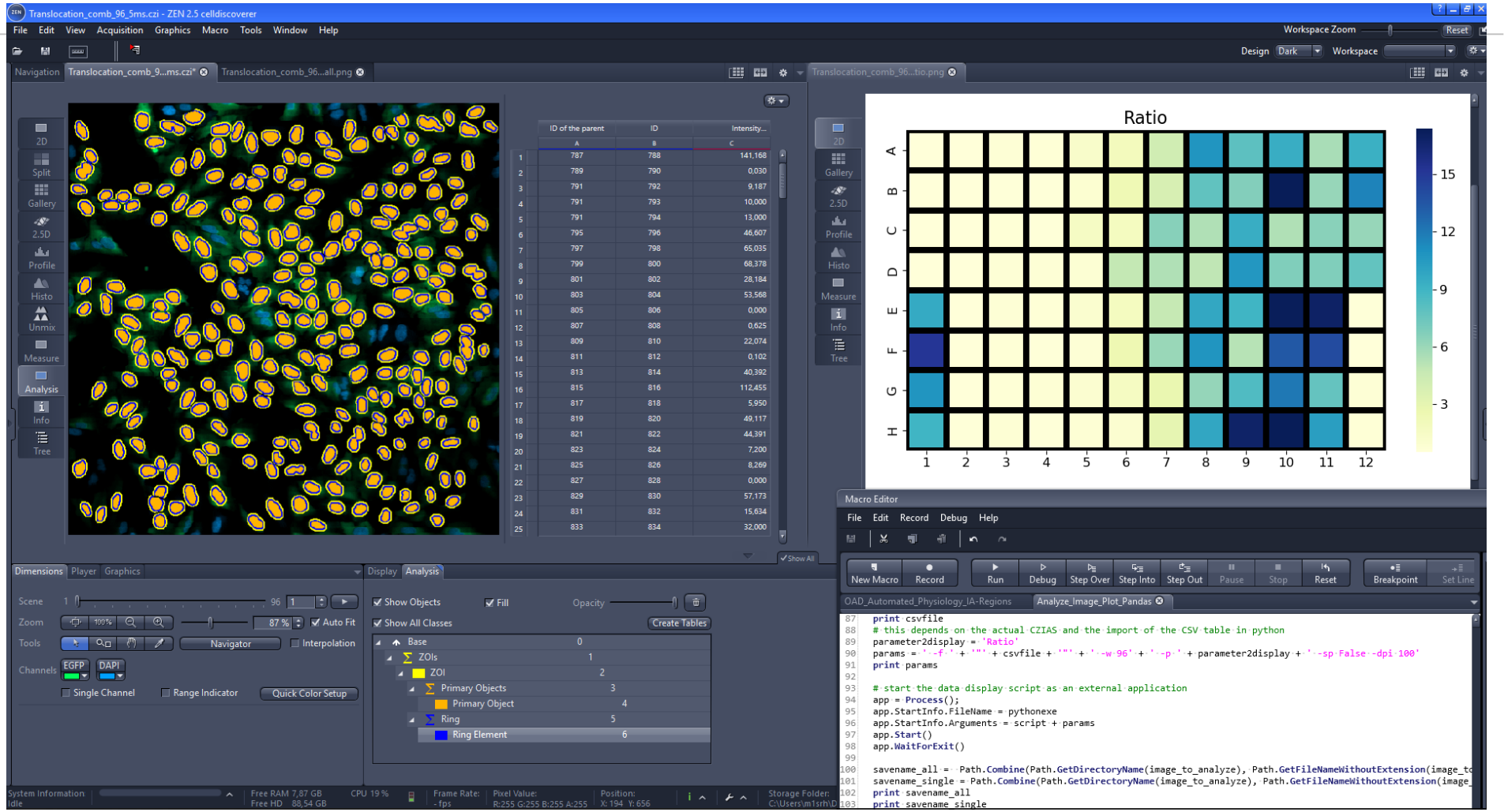


	Image Scene...	Image Scene...	Image Scene...	ID	Index	Intensity...	Area[µm²]	Perimeter[µm]	Roundness	Feret Ratio
	A	B	C	D	E	F	G	H	I	J
177	A1	1	1	178	177	88,838	2,590	6,936	0,443	0,490
178	A1	1	1	179	178	79,187	3,690	7,428	0,665	0,677
179	A1	1	1	180	179	84,276	1,920	5,911	0,499	0,542
180	A1	1	1	181	180	58,310	1,260	4,804	0,472	0,537
181	A1	1	1	182	181	92,217	2,030	5,994	0,475	0,552
182	A1	1	1	183	182	98,272	2,460	6,487	0,557	0,655
183	A1	1	1	184	183	106,135	2,370	6,346	0,493	0,525
184	A1	1	1	185	184	75,936	1,570	4,838	0,655	0,630
185	A1	1	1	186	185	87,677	2,510	7,760	0,315	0,412
186	A1	1	1	187	186	88,252	1,350	4,521	0,589	0,644
187	A1	1	1	188	187	113,247	2,190	6,063	0,523	0,557
188	A1	1	1	189	188	92,274	1,860	5,453	0,578	0,644
189	A1	1	1	190	189	81,016	1,930	5,428	0,710	0,835
190	A1	1	1	191	190	109,221	2,350	6,228	0,530	0,580
191	A1	1	1	192	191	89,792	1,920	5,428	0,661	0,722
192	A1	1	1	193	192	72,130	1,610	5,038	0,550	0,570
193	A1	1	1	194	193	101,919	2,480	6,453	0,632	0,776
194	A1	1	1	195	194	92,238	2,020	5,828	0,485	0,545
195	A1	1	1	196	195	89,691	2,820	6,711	0,571	0,598
196	A1	1	1	197	196	73,360	1,720	5,336	0,581	0,656
197	A1	1	1	198	197	84,449	1,360	4,380	0,770	0,865
198	A1	1	1	199	198	89,904	1,360	4,380	0,833	0,901
199	A1	1	1	200	199	63,728	1,840	5,853	0,417	0,491
200	A1	1	1	201	200	61,844	1,090	4,180	0,479	0,578
201	A1	1	1	202	201	85,805	2,360	6,594	0,414	0,482
202	A1	1	1	203	202	83,817	0,820	3,521	0,682	0,808
203	A1	1	1	204	203	85,278	0,790	3,438	0,689	0,815
204	A1	1	1	205	204	66,295	1,900	5,663	0,465	0,589
205	A1	1	1	206	205	88,090	2,330	6,111	0,521	0,593
206	A1	1	1	207	206	112,233	2,270	6,028	0,556	0,608
207	A1	1	1	208	207	85,490	1,550	4,780	0,607	0,610

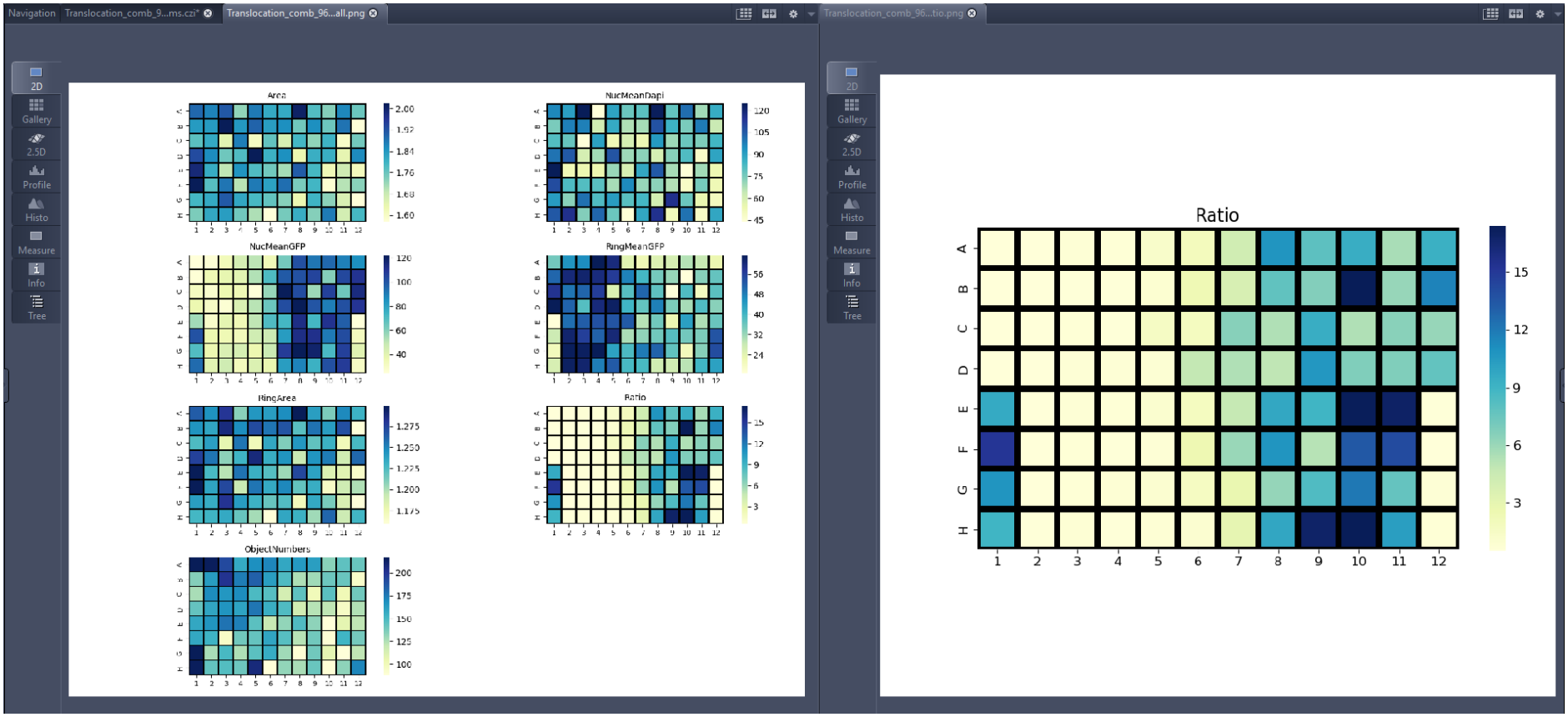
- Parameters for single objects are measured.
- Data Tables can be exported as CSV easily for usage in external applications.

OAD – Automated Image Analysis



- Scripting is used to automate the image analysis and to trigger the external visualization

OAD – Use external tools to visualize your results



- Data Tables can be exported easily for usage in external applications
- Those images are imported back into ZEN on demand

External SW is our friend – Python, MATLAB, R …

In [6]:

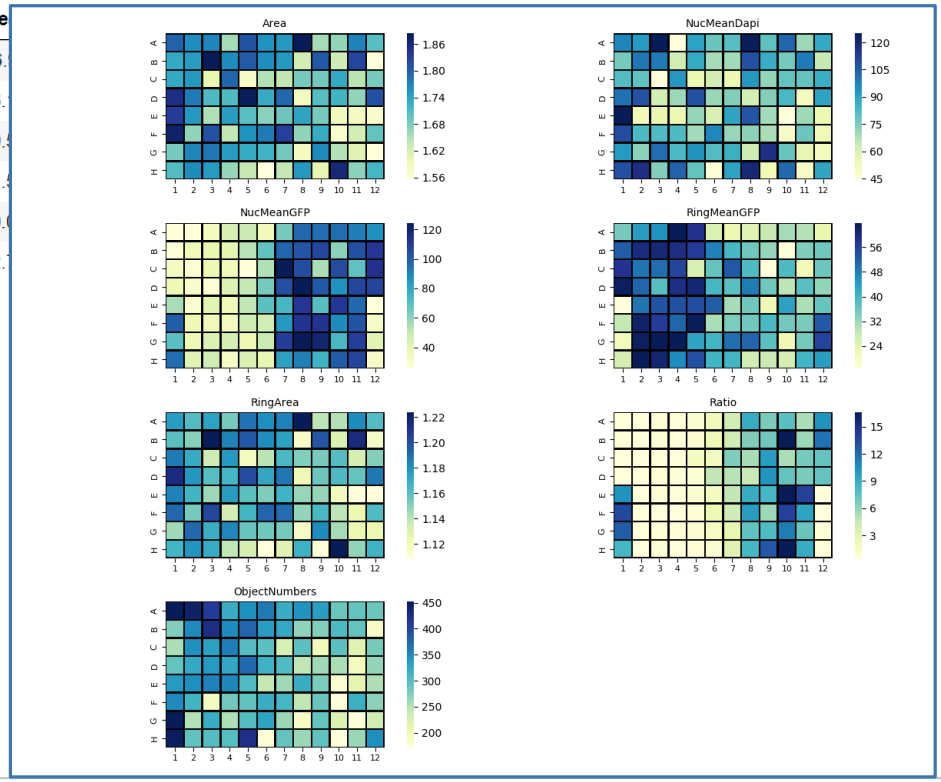
Slide Type: Slide

```
# calculate the number of actual measurment parameters for single objects
num_param = len(df_single.columns) - num_nonmp
print('Number of Object Parameters: ', num_param)

# show part of dataframe
df_single[:6]
```

```
Number of Object Parameters:  6
```

Out[6]:

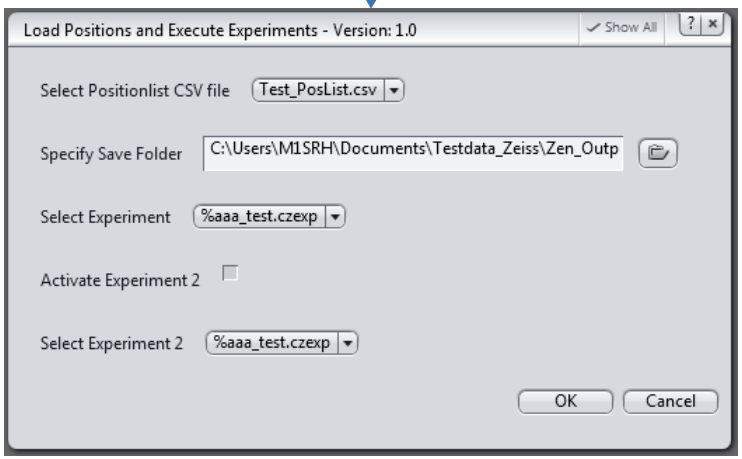
	ParentID	ID	WellID	RowID	ColumnID	Index	Area	NucMeanDapi	NucMe
1	291.0	292.0	A1	1.0	1.0	1.0	2.61	134.620690	86.
2	293.0	294.0	A1	1.0	1.0	2.0	1.52	106.664474	36.
3	295.0	296.0	A1	1.0	1.0	3.0	2.92	173.167808	9.
4	297.0	298.0	A1	1.0	1.0	4.0	1.14	110.403509	41.
5	299.0	300.0	A1	1.0	1.0	5.0	1.77	79.909605	0.
6	301.0	302.0	A1	1.0	1.0	6.0	2.46	146.914634	52.



OAD Workflows – Dialogs to make life easy

- Create Position List with Tiles & Positions and export coordinates as CSV file
- Re-Import position list, specify **Multiblock Experiment(s)** and Save Folder
- Move to 1st XYZ position
- Run experiment(s) for all experiments and all blocks and move no next position

Write OAD script to create simple GUI



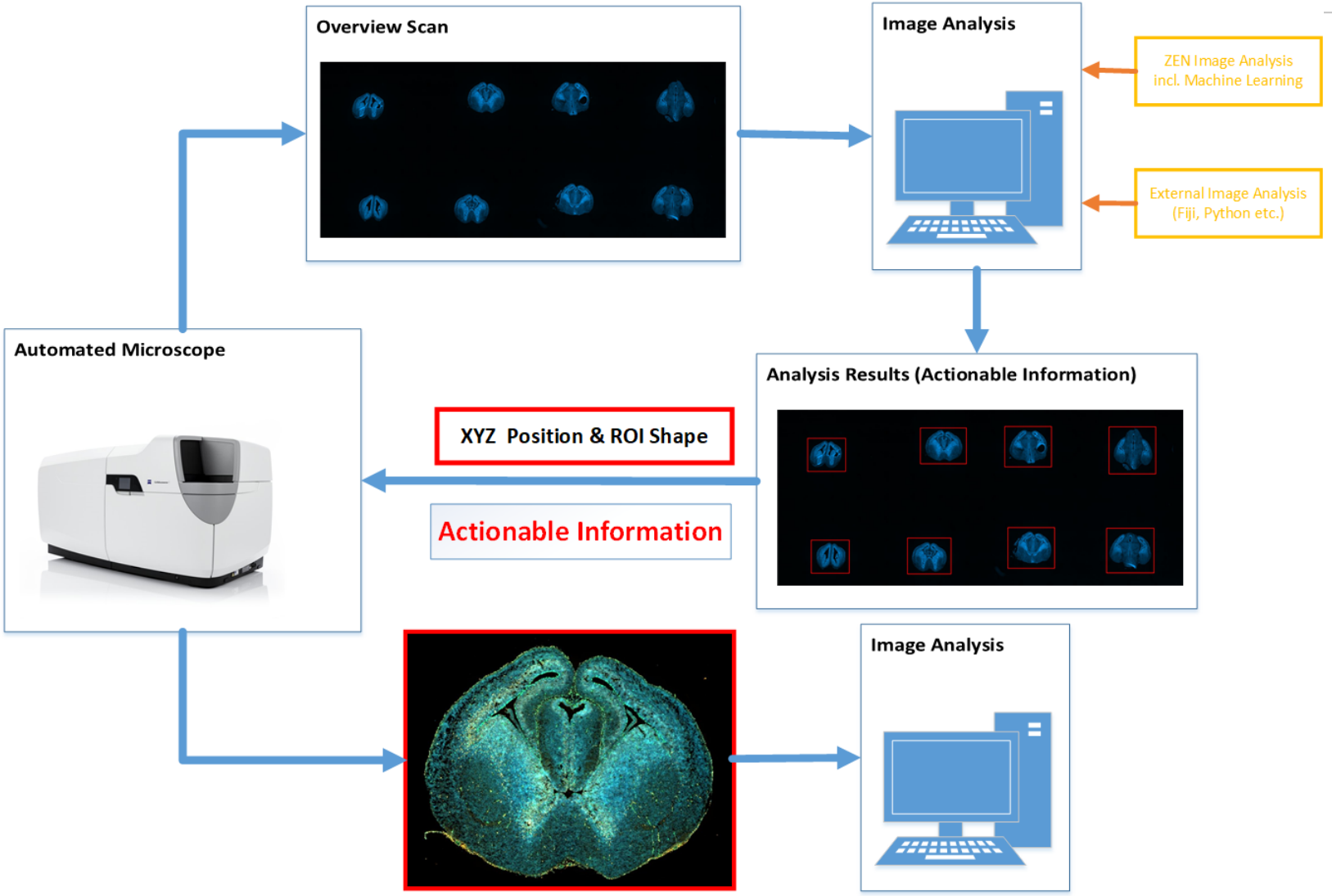
OAD - Adaptive Feedback Microscopy?

There are obviously different ways how to define **Adaptive Feedback Microscopy**, but in general one or more of the points below apply:

- **Automatically guide the system to the correct places inside a sample**
- **React on changes inside the sample during a running acquisition**
- **Optimized acquisition parameters based on the analysis of the current sample**
- **React on external signals from the “outside” and adapt the acquisition or the workflow based on those**
- **Send signal to the “outside” based on online or offline image analysis to modify the sample and continue with the workflow**

Adaptive Feedback Microscopy

Use Image Analysis to Automate the workflows



Interfaces and Tools for Adaptive Feedback Microscopy

- ZEN Image Analysis to extract Actionable Information
  - classical segmentation and Machine Learning methods can be used

- OAD Python Environment
  - powerful and flexible python scripts to automate workflows
  - Call external software and exchange data
  - Create dialogs and User Interfaces for Ease-of-Use
  - Automate Image Analysis workflows

- Experiment Feedback
  - Online Image Analysis during running experiments
  - Modification of Experiments on runtime
  - Sending or receiving signals to the “outside”
  - Powerful combinations with Experiment designer for heterogeneous acquisition workflows

- TCP/IP Interface to control ZEN from the “outside” or vice versa

Adaptive Feedback Microscopy
General Considerations

- What is the actual **nature of the desired feedback** and upon what **event** it should be triggered?
- What exactly is the **Actionable Information** to be extracted?
- On what **timescale** this feedback is required?
- Is **Online Image Analysis** available and is it sufficient to detect the feedback event?
- Which **interfaces** can be used to communicate with **external image analysis** tools or **external devices**?
- What is right **choice of hardware** and is it ready to be automated?
- What could go potentially **wrong** inside such an **automated workflow** and what be the **consequences**?

Adaptive Feedback Microscopy
Data Format Considerations

Especially in the case where external image analysis is needed the “painless” exchange of data becomes crucial!

- CZI can be read easily by many open source and commercial software packages
  - Fiji, ImageJ, Python, KNIME, Icy
  - MATLAB, Imaris, Arivis, ORS
  - …
- Constant exchange with BioFormats team to keep **their** CZIReader up-to-date
- ZEN has option for BioFormats import to read 3rd party images (paid module)
- Zeiss offers two open available APIs to read CZI on any platform
  - libCZI (C++) for cross-platform applications (Windows, Linux, MacOS)
  - ZeissImgLib (c#) for Windows only
  - Python wrapper for libCZI is in progress

What is the right system for Adaptive Feedback?

Well, it depends from your application …

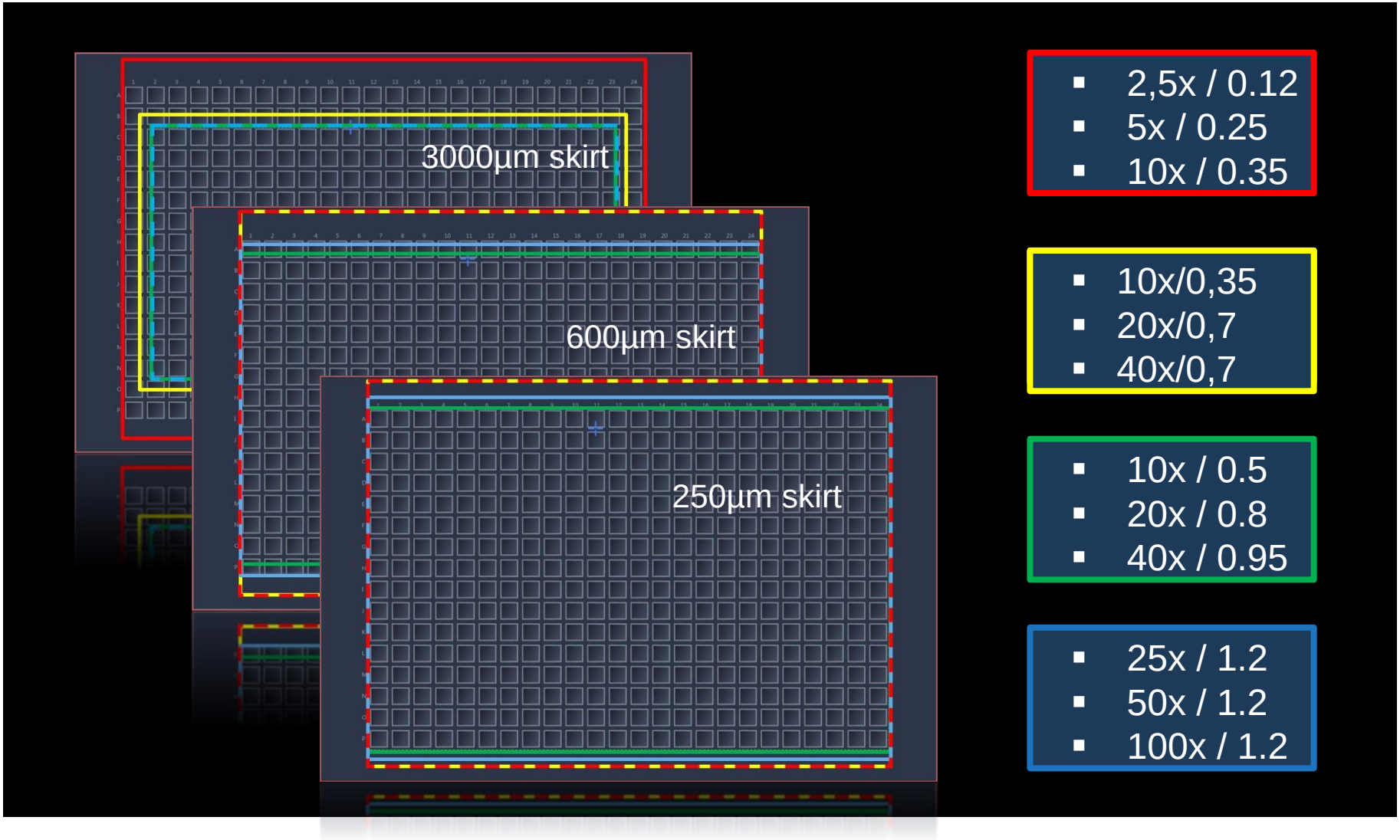
… but all motorized ZEN Blue systems can be used

Adaptive Feedback Microscopy – Potential Pitfalls

When things go wrong …

Adaptive Feedback Microscopy – Potential Issues

Adaptive Lens Guard of Celldiscoverer 7

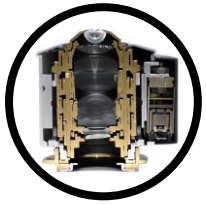


Celldiscoverer 7

Ready for Adaptive Feedback Microscopy and Automation

Unique Optical Concept

Unparallelled Sensitivity

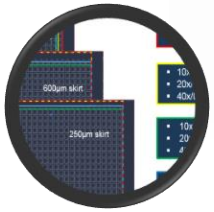


AutoCorrection

Ideal Image Quality

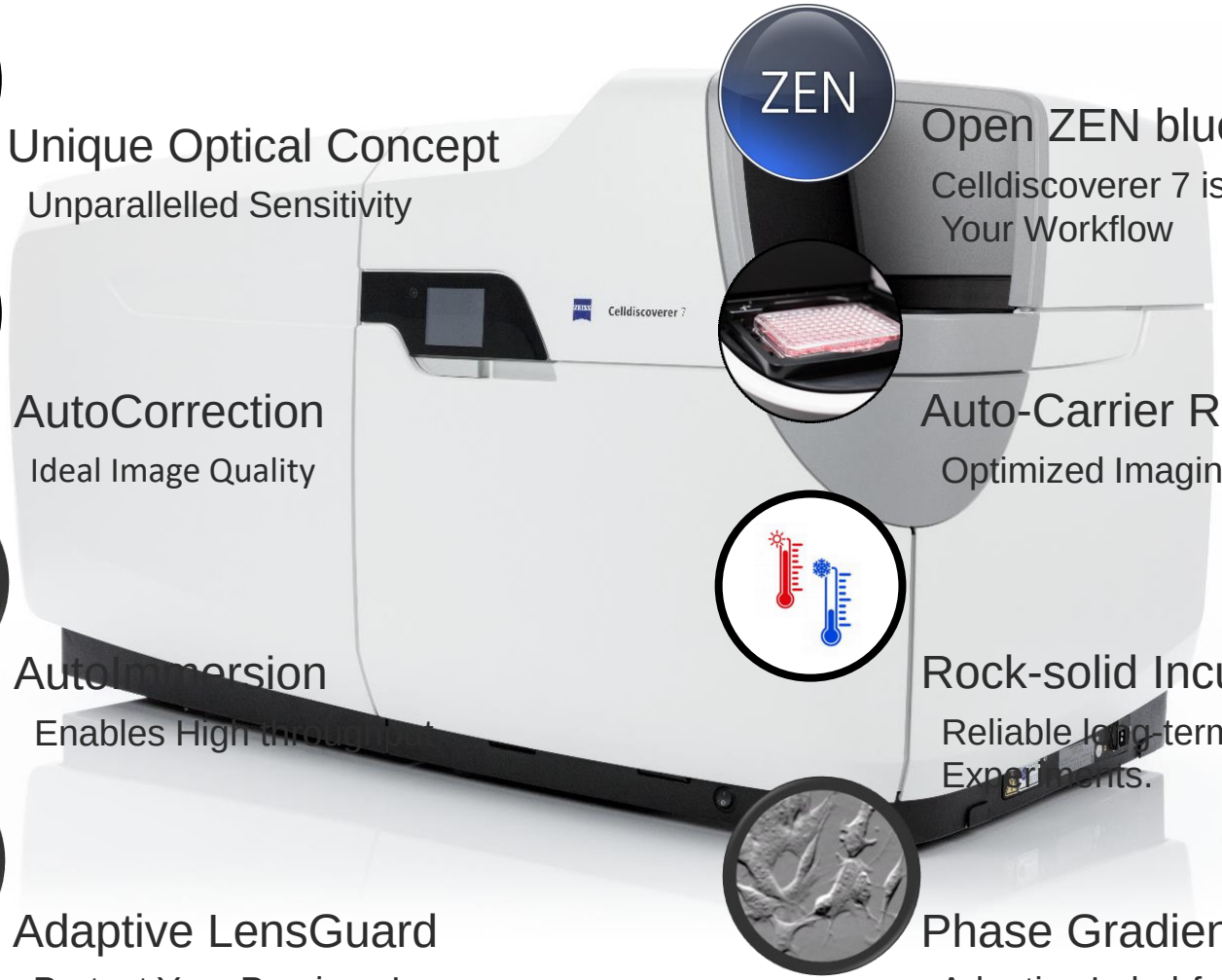
AutoImmersion

Enables High throughput



Adaptive LensGuard

Protect Your Precious Lenses



Open ZEN blue SW

Celldiscoverer 7 is Part of Your Workflow

Auto-Carrier Recognition

Optimized Imaging Conditions

Rock-solid Incubation

Reliable long-term Timelapse Experiments.

Phase Gradient Contrast

Adaptive Label-free Imaging

Adaptive Feedback Microscopy

Make use of scriptable Software Autofocus

SWAF.py

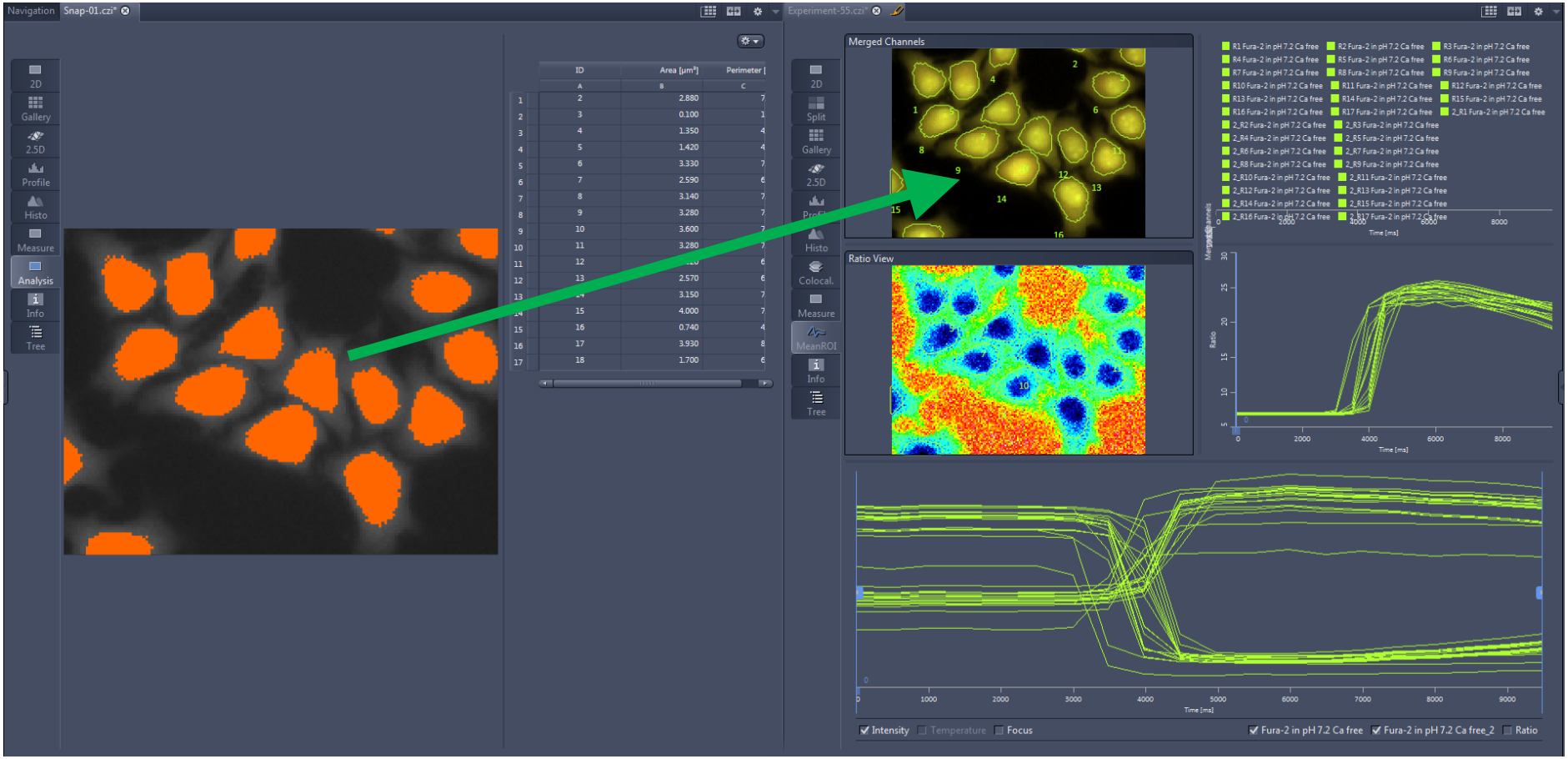
```python
def runSWAF_special(expname,
                    delay=5,
                    searchStrategy='Full',
                    sampling=ZenSoftwareAutofocusSampling.Coarse,
                    relativeRangeIsAutomatic=False,
                    relativeRangeSize=500,
                    timeout=0):

    # get current z-Position
    zSWAF = Zen.Devices.Focus.ActualPosition

    # run intial SWAF with the DetailScan experiment settings
    SWAF_exp = Zen.Acquisition.Experiments.GetByName(expname)
    # set DetailScan active and wait for moving hardware due to settings
    SWAF_exp.SetActive()
    time.sleep(delay)

    # set SWAF parameters
    SWAF_exp.SetAutofocusParameters(searchStrategy=searchStrategy,
                                    sampling=sampling,
                                    relativeRangeIsAutomatic=relativeRangeIsAutomatic,
                                    relativeRangeSize=relativeRangeSize)

    zSWAF = Zen.Acquisition.FindAutofocus(SWAF_exp, timeoutSeconds=timeout)
    SWAF_exp.Close()

    return zSWAF

########################################

zSWAF = runSWAF_special(DetailExpName,
                        delay=hwdelay,
                        searchStrategy='Full',
                        sampling=ZenSoftwareAutofocusSampling.Coarse,
                        relativeRangeIsAutomatic=False,
                        relativeRangeSize=SWAF_beforeDT_range,
                        timeout=0)

```

- SWAF is now completely scriptable
- Define special SWAF runs via python scripting
- Use it to define a safe starting point for a Focus Strategy

Adaptive Feedback Microscopy

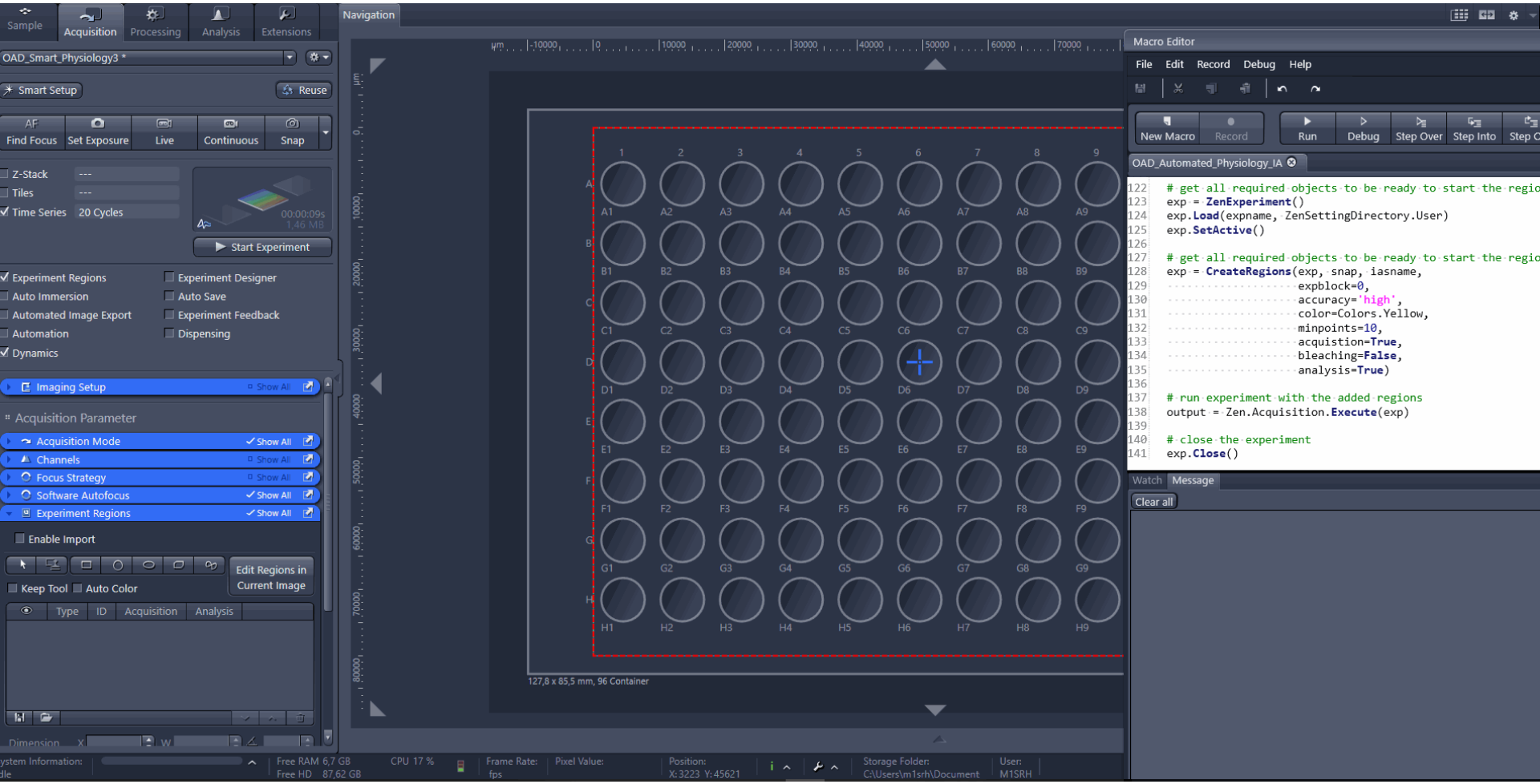
Convert Image Analysis Results into Actionable Information

- Create regions for Dynamics automatically based on IA results



Adaptive Feedback Microscopy

Convert Image Analysis Results into Actionable Information

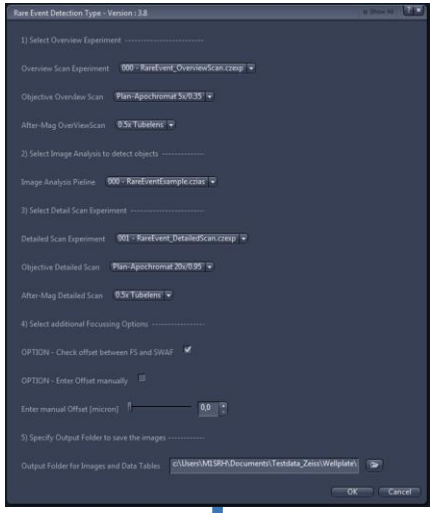


Adaptive Feedback Microscopy
Convert Image Analysis Results into Actionable Information

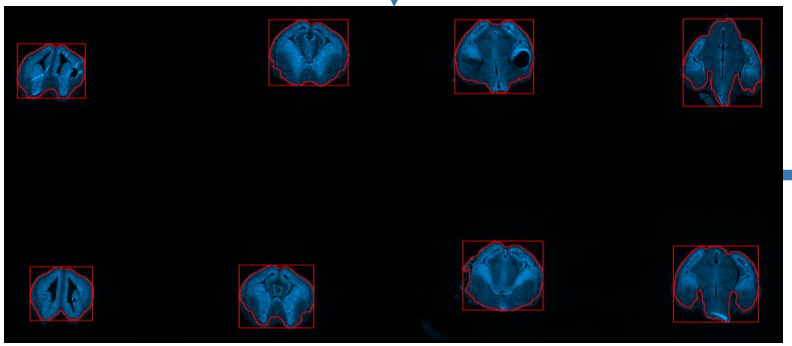
- commands to modify experiments based on Image Analysis results interactively or using python scripting
- Create regions for Dynamics (Online Ratio) automatically based on segmented objects
- Use Image Analysis results (detected regions) and convert them to
  - **Experiment Regions**
  - **FRAP Regions**
  - **New Tile Regions**

Adaptive Feedback Microscopy

Complete workflow scheme



OAD - Automated Acquisition and Image Analysis

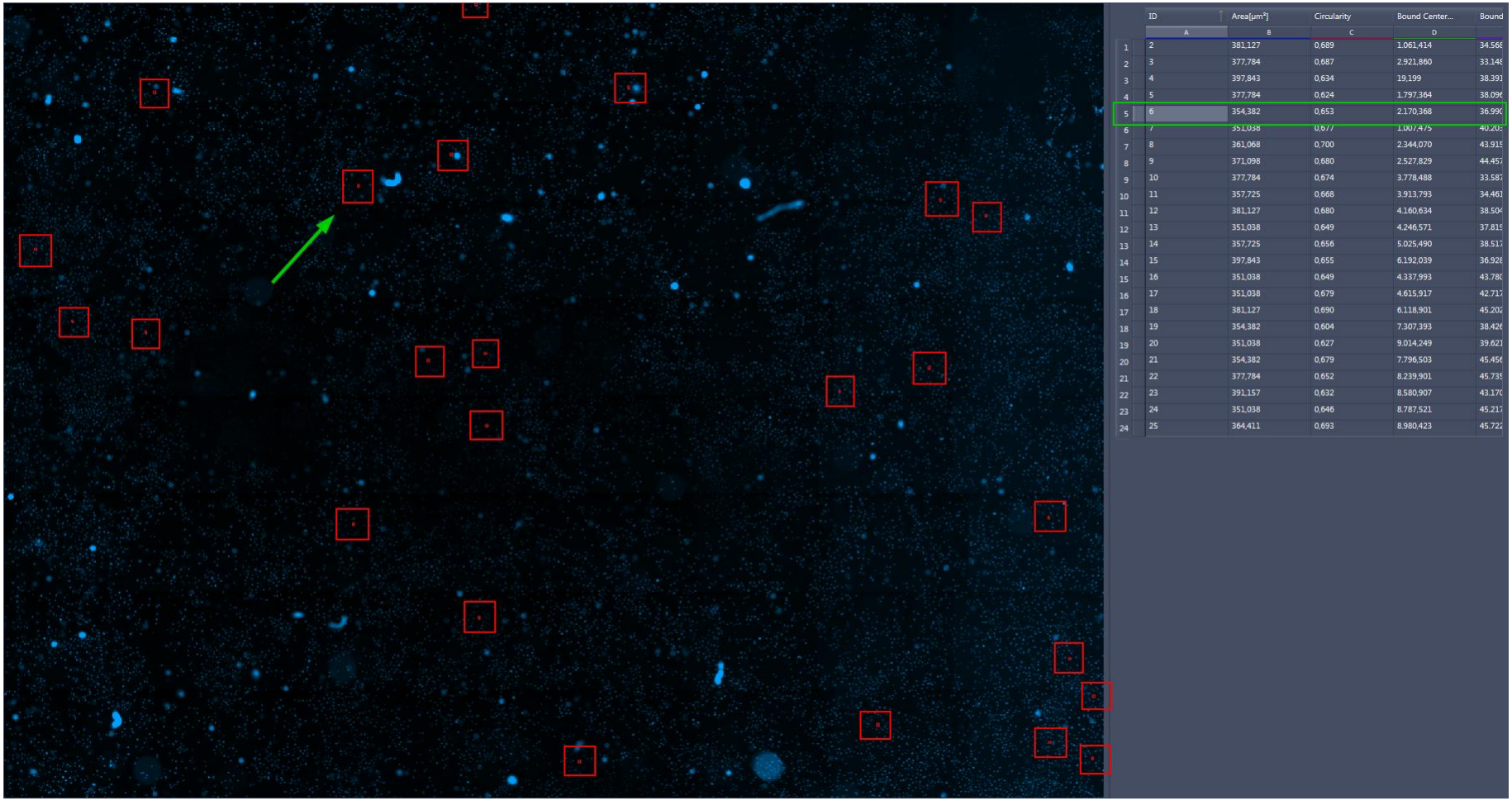


	ID	Bound Center X Stage[µm]	Bound Center Y Stage[µm]	Bound Width[µm]	Bound Height[µm]
	A	B	C	D	E
1	2	40.786,876	10.236,734	3.014,505	2.511,325
2	3	47.835,969	10.371,678	2.996,208	2.799,510
3	4	30.956,571	10.934,324	2.584,515	2.085,909
4	5	56.540,981	10.604,970	2.996,208	3.320,987
5	6	31.372,838	19.403,757	2.392,392	2.035,591
6	7	48.176,759	18.745,048	3.046,525	2.639,407
7	8	56.309,976	19.044,669	3.238,649	2.909,295
8	9	39.556,373	19.508,967	2.831,530	2.383,243

OAD – Cycle through list, modify experiment and acquire Detailed Scan

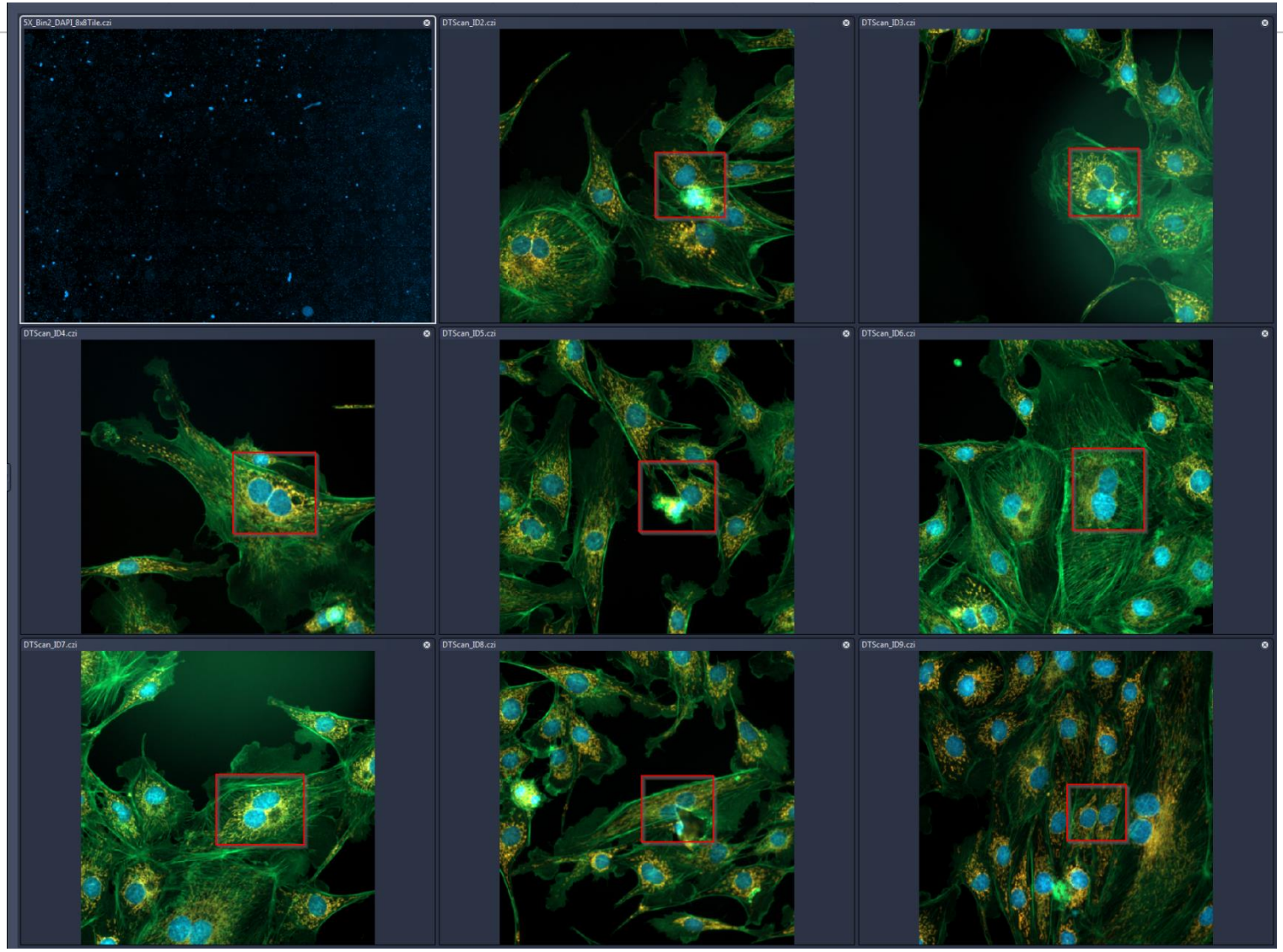
Adaptive Feedback Microscopy

Detect “*Rare*” objects automatically



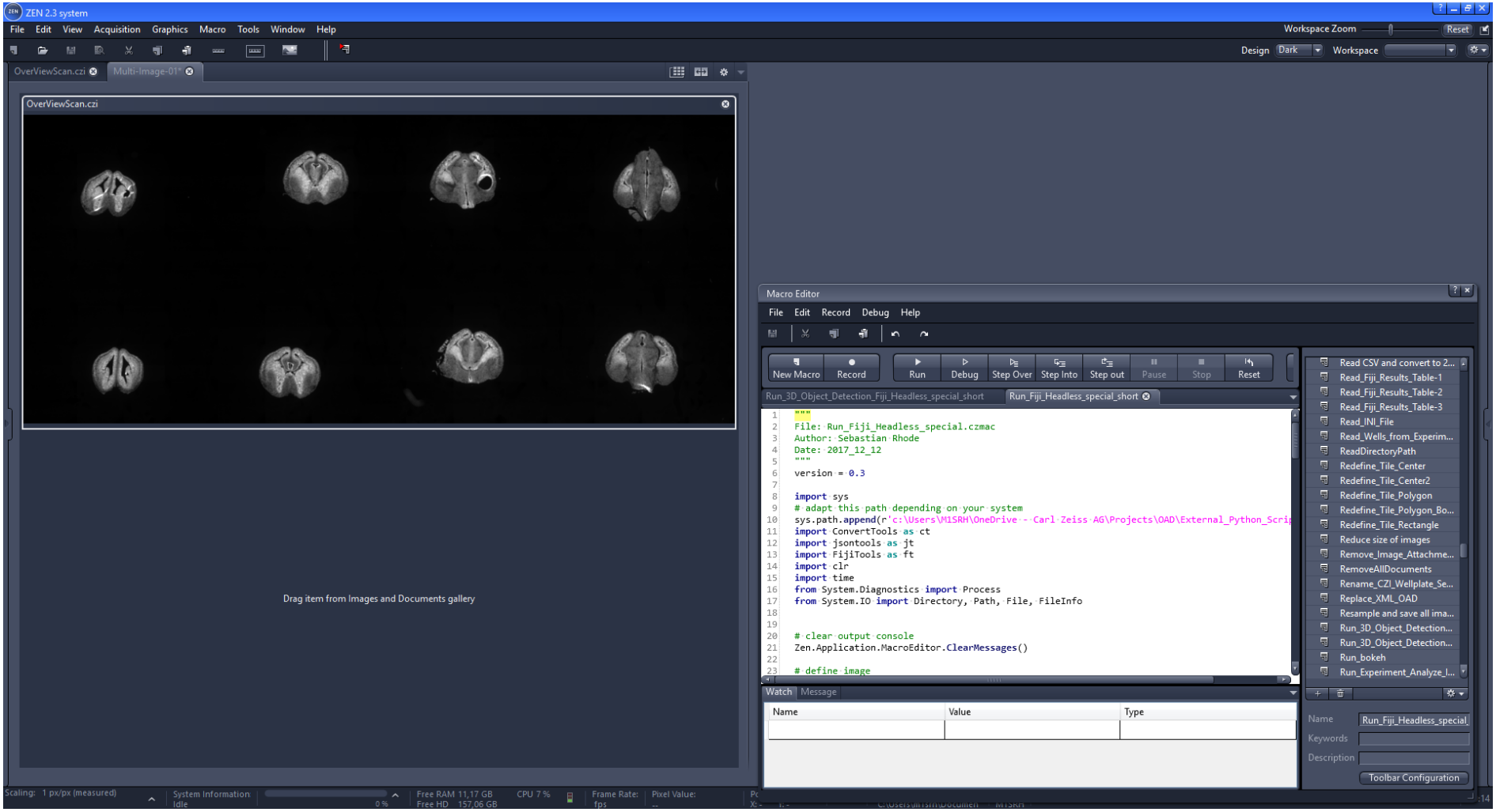
	ID	Area[µm²]	Circularity	Bound Center...	Bound
	A	B	C	D	
1	2	381,127	0,689	1.061,414	34.566
2	3	377,784	0,687	2.921,860	33.148
3	4	397,843	0,634	19,199	38.391
4	5	377,784	0,624	1.797,364	38.096
5	6	354,382	0,653	2.170,368	36.990
6	7	351,038	0,677	1.007,475	40.205
7	8	361,068	0,700	2.344,070	43.915
8	9	371,098	0,680	2.527,829	44.457
9	10	377,784	0,674	3.778,488	33.587
10	11	357,725	0,668	3.913,793	34.461
11	12	381,127	0,680	4.160,634	38.504
12	13	351,038	0,649	4.246,571	37.815
13	14	357,725	0,656	5.025,490	38.517
14	15	397,843	0,655	6.192,039	36.928
15	16	351,038	0,649	4.337,993	43.780
16	17	351,038	0,679	4.615,917	42.717
17	18	381,127	0,690	6.118,901	45.202
18	19	354,382	0,604	7.307,393	38.426
19	20	351,038	0,627	9.014,249	39.621
20	21	354,382	0,679	7.796,503	45.456
21	22	377,784	0,652	8.239,901	45.735
22	23	391,157	0,632	8.580,907	43.170
23	24	351,038	0,646	8.787,521	45.217
24	25	364,411	0,693	8.980,423	45.722

Adaptive Feedback Microscopy



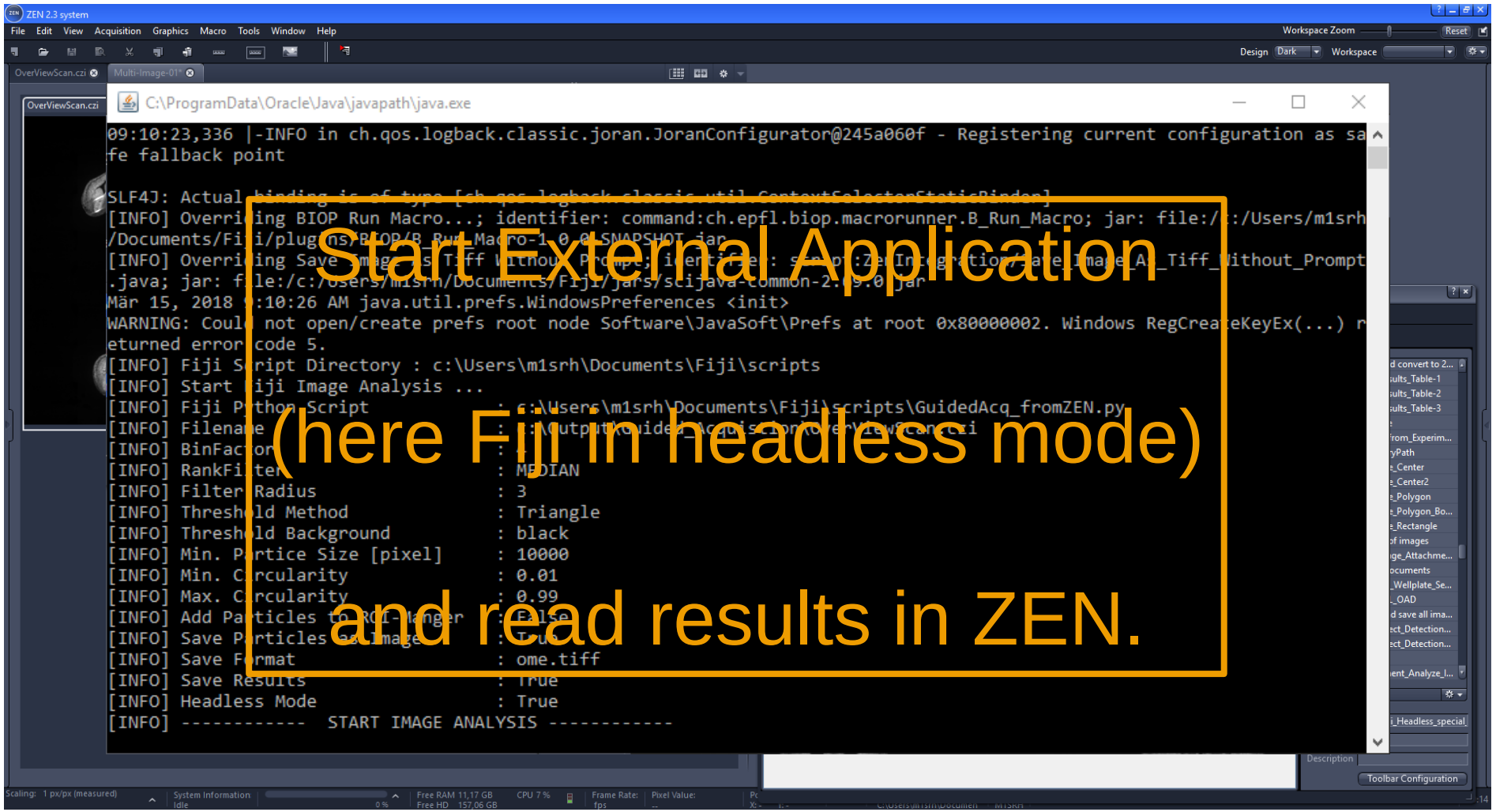
Adaptive Feedback Microscopy

Run Experiment



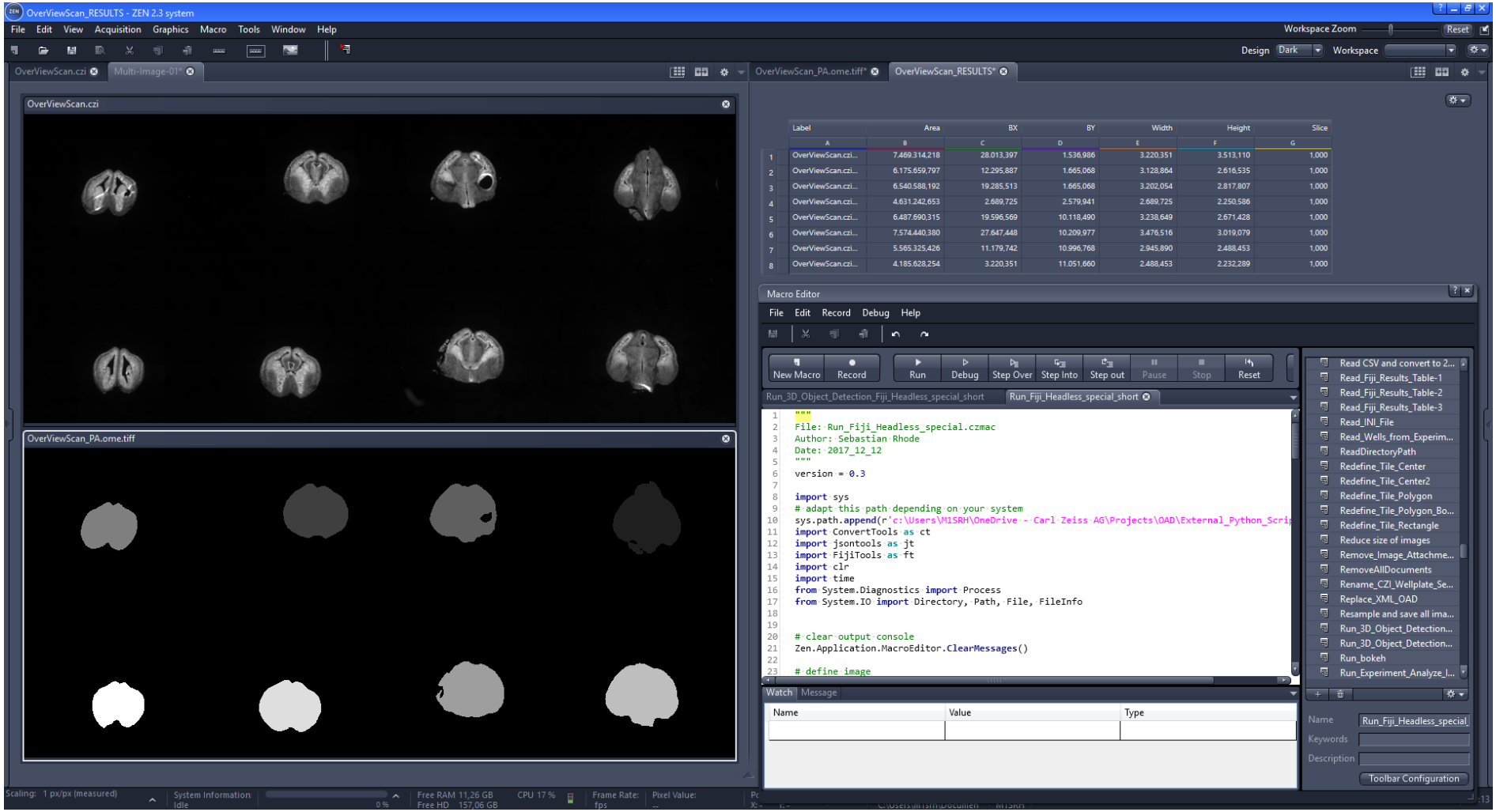
Adaptive Feedback Microscopy

Run External Analysis



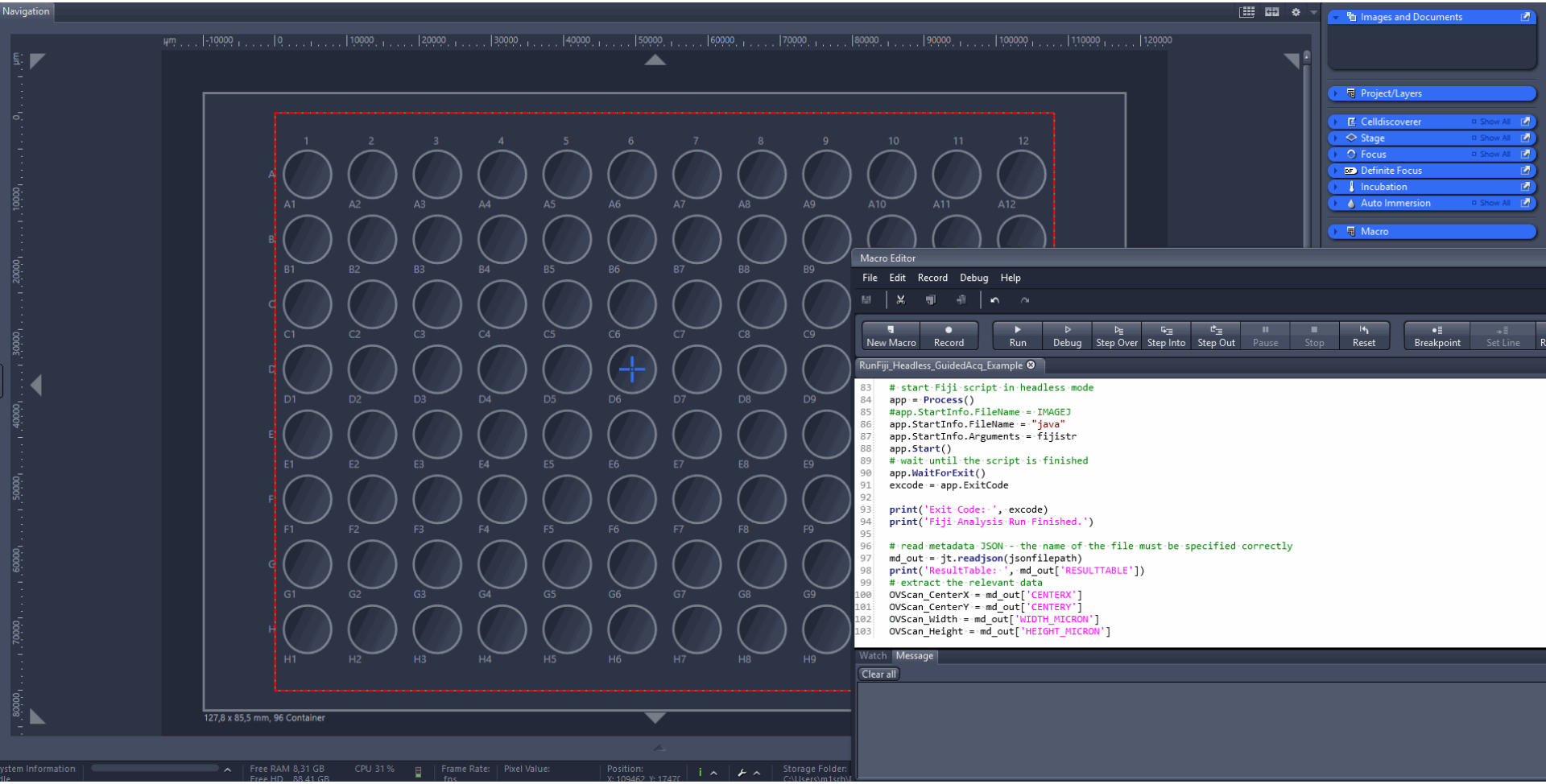
Adaptive Feedback Microscopy

Read Results and take Action



Adaptive Feedback Microscopy

Read Results and take Action



Experiment Feedback
Important Features

- **Adaptive Acquisition Engine** allows modifying **running experiments** using Python scripts.

- Python Scripts can access the **current system status** and results from **Online Image Analysis** on runtime during the experiment.

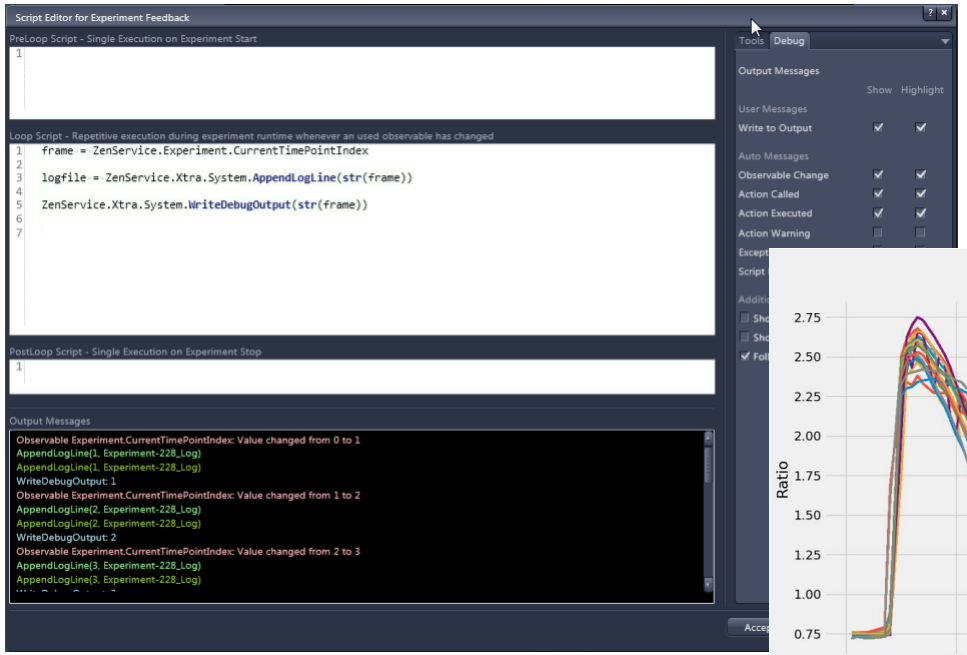
- **Data Logging** or starting an **External Application** (Python, Fiji, MATLAB, …), directly from within the imaging experiment is possible.

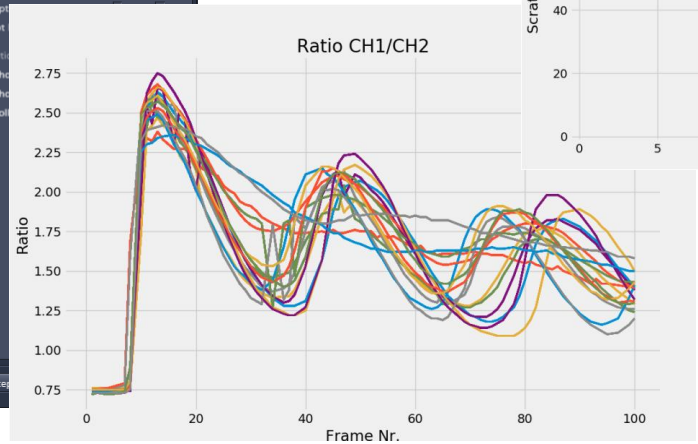
Experiment Feedback
Important Features

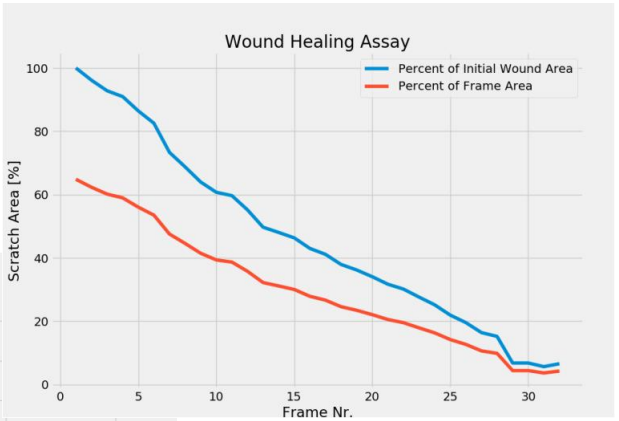
- Create dynamic acquisition-experiments
- Observe parameters during acquisition via online image analysis
- Monitor the status of the microscope and/or the sample
- Automatically react on changes of the sample or other parameters
- Modify the hardware/experiment parameters during the acquisition
  - increase integration time when the sample bleaches
  - stop the acquisition after a certain number of objects was detected
- Create custom log-files, integrate data logging in the ZEN experiment
- Start external applications (Python, Matlab,….)
- Display measurement results already during image acquisition

Experiment Feedback
New Features in ZEN 2.5

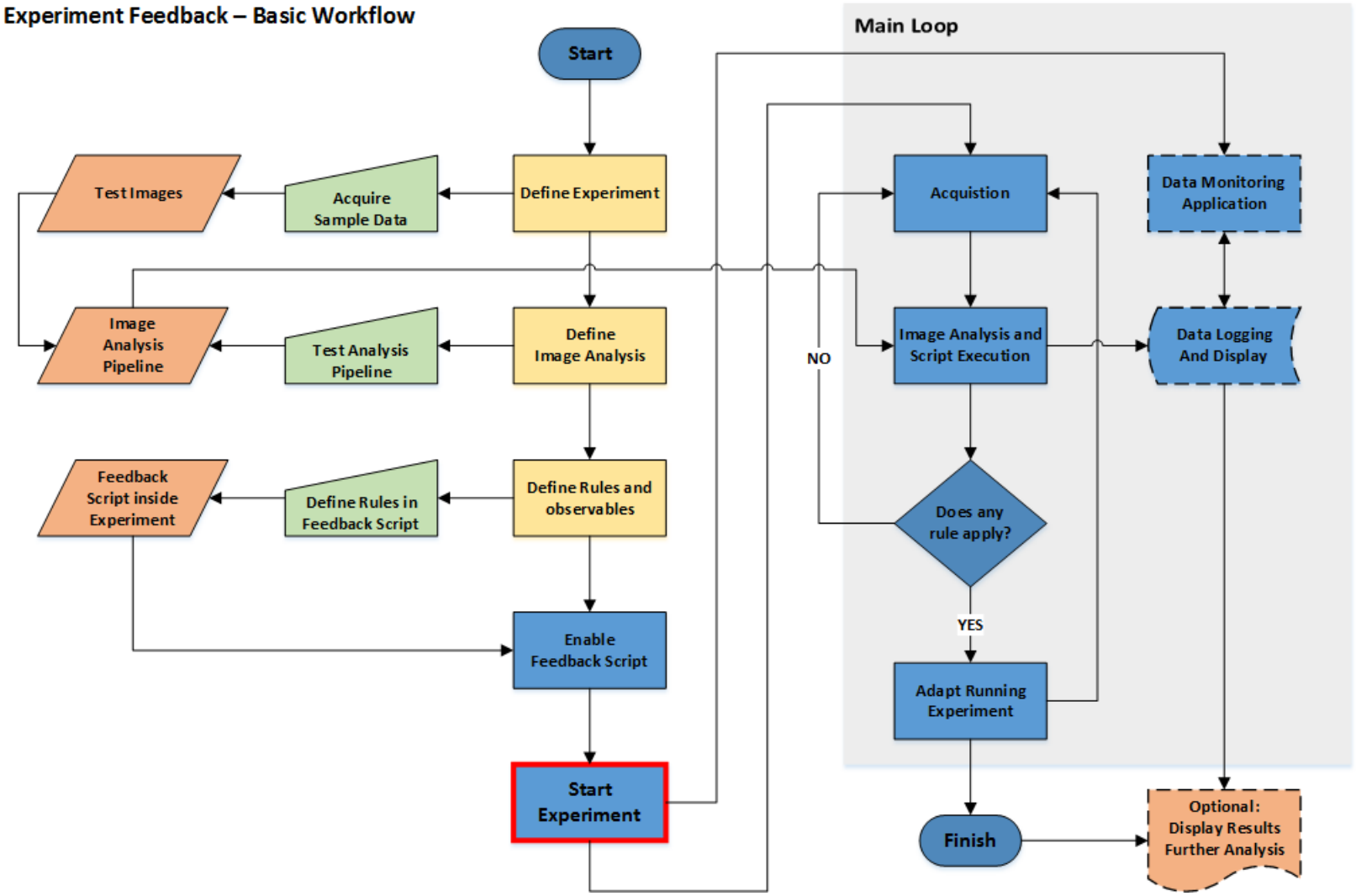
- New commands to update tiles and position list
- Integrated logging window for full control over the feedback scripts
- Updated and extensive tutorial with example for Online Assays



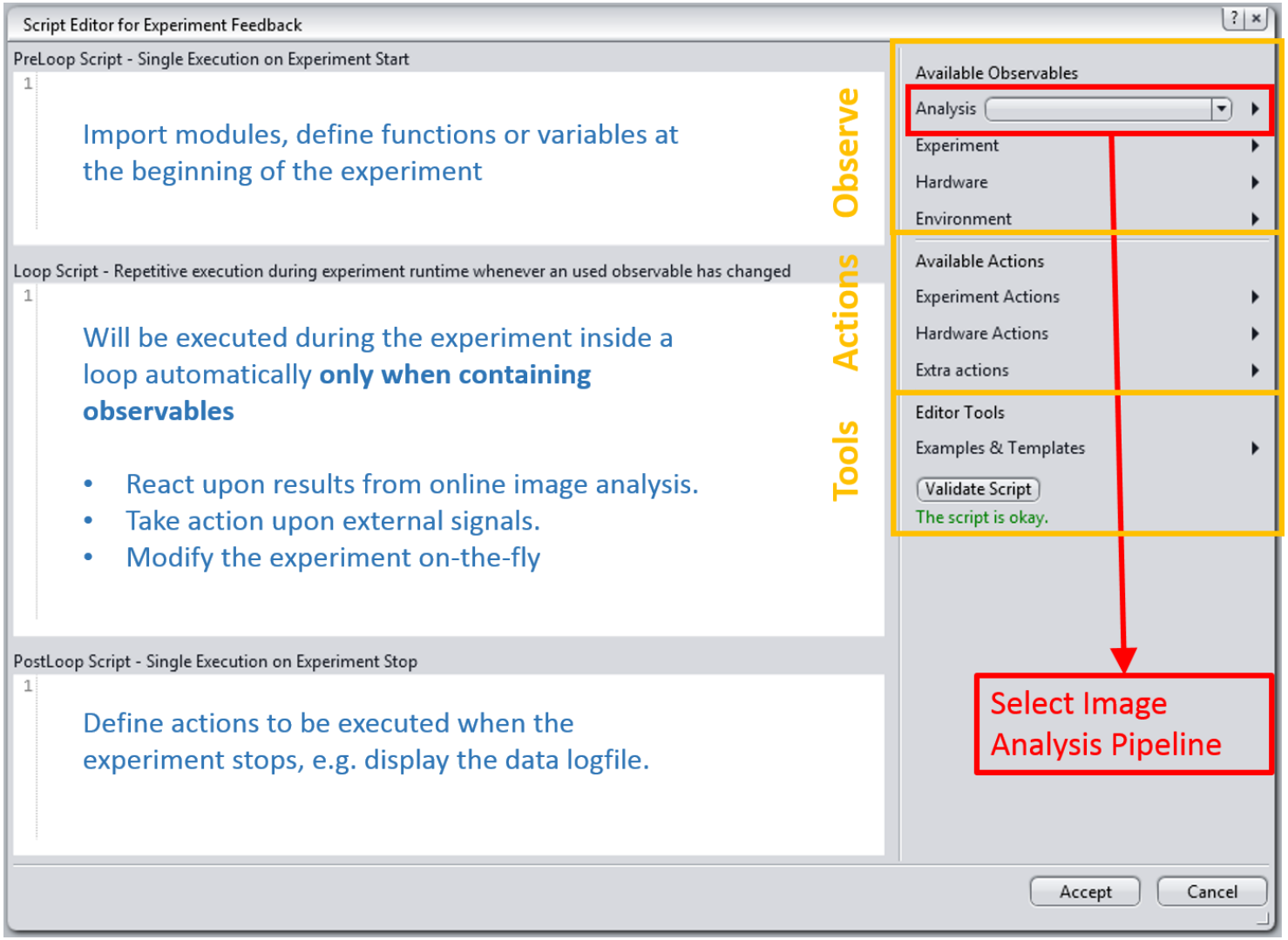




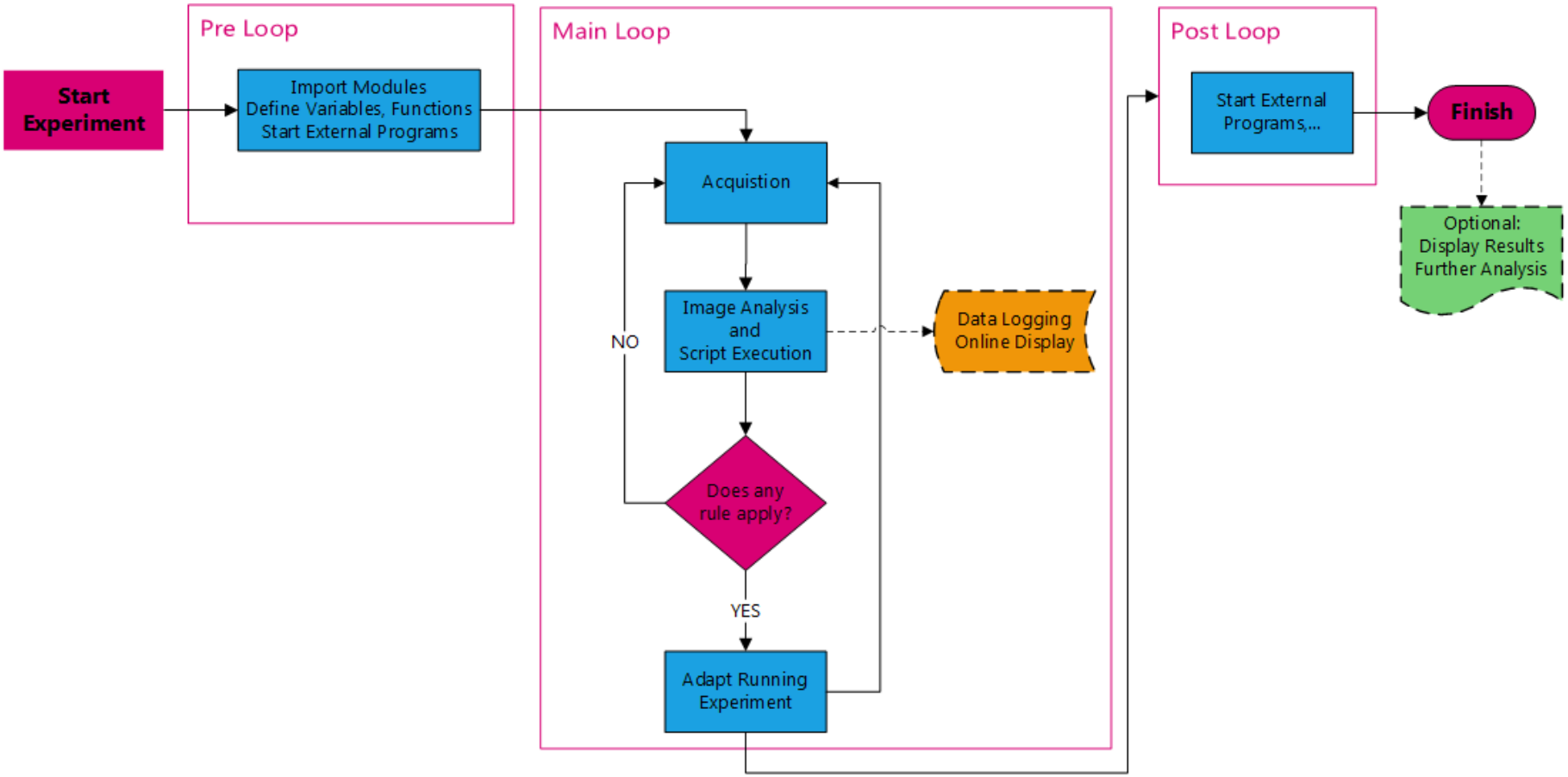
Experiment Feedback
General Workflow



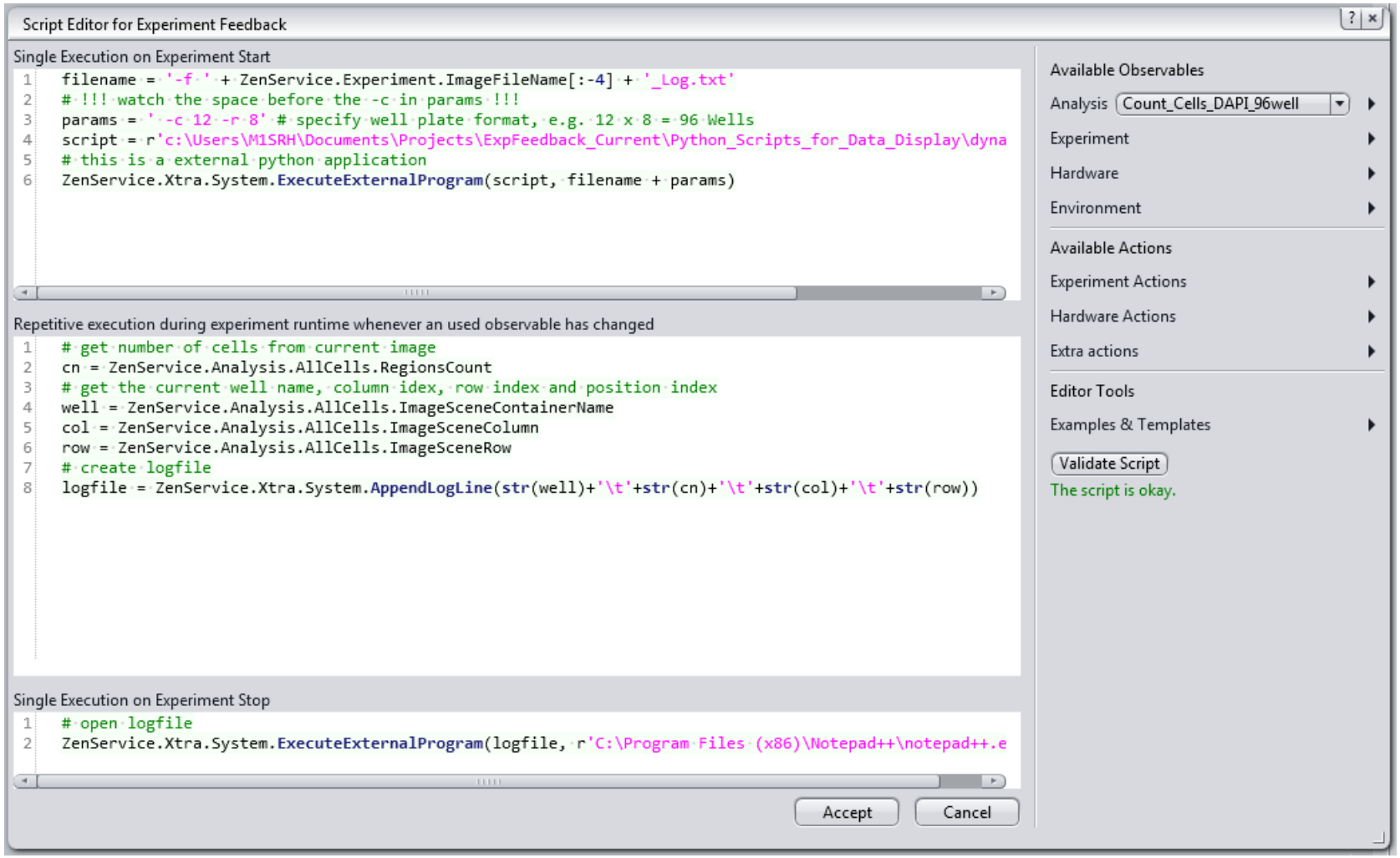
Experiment Feedback
Adaptive Acquisition Engine



Experiment Feedback
Adaptive Acquisition Engine

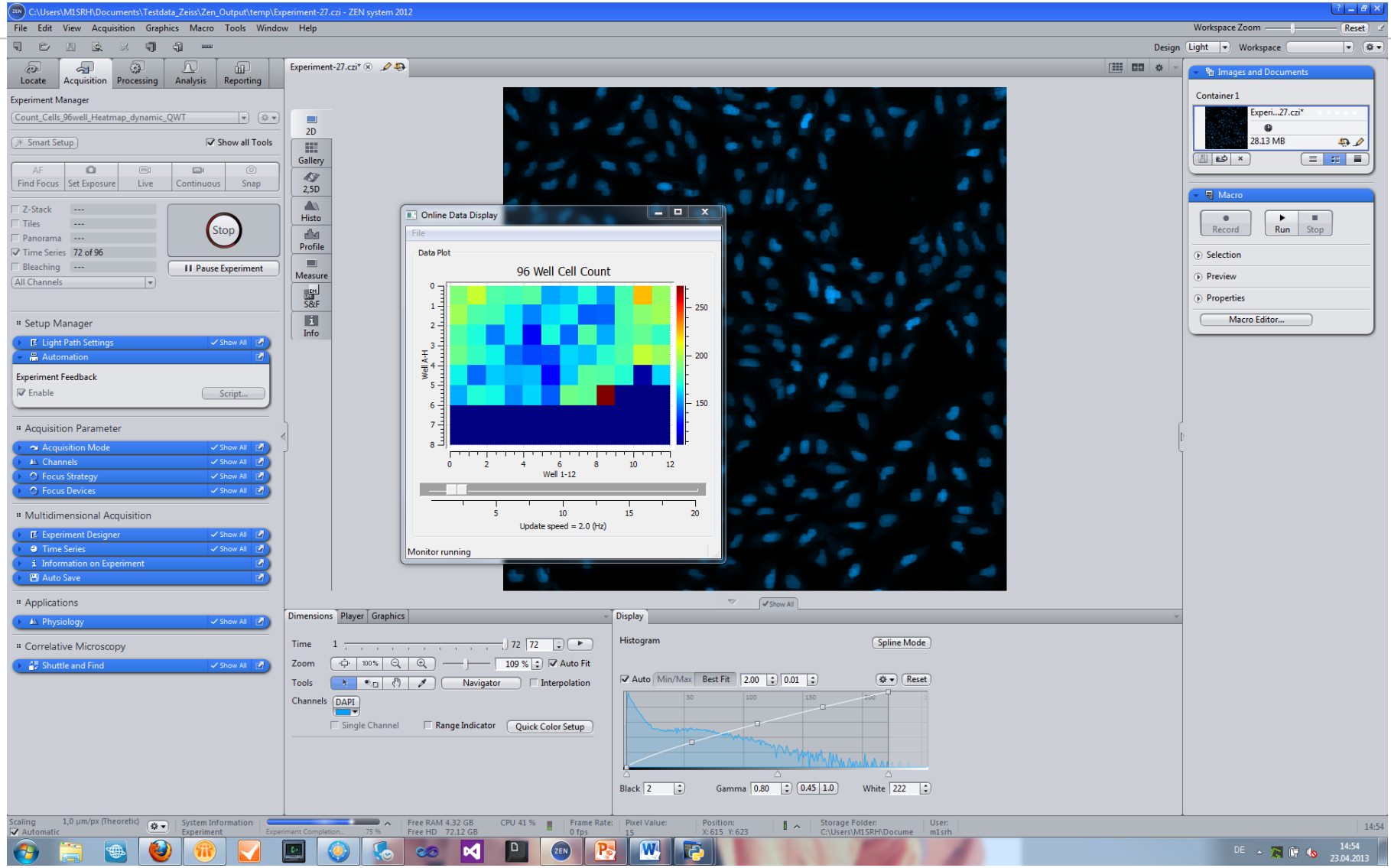


Experiment Feedback
Adaptive Acquisition Engine



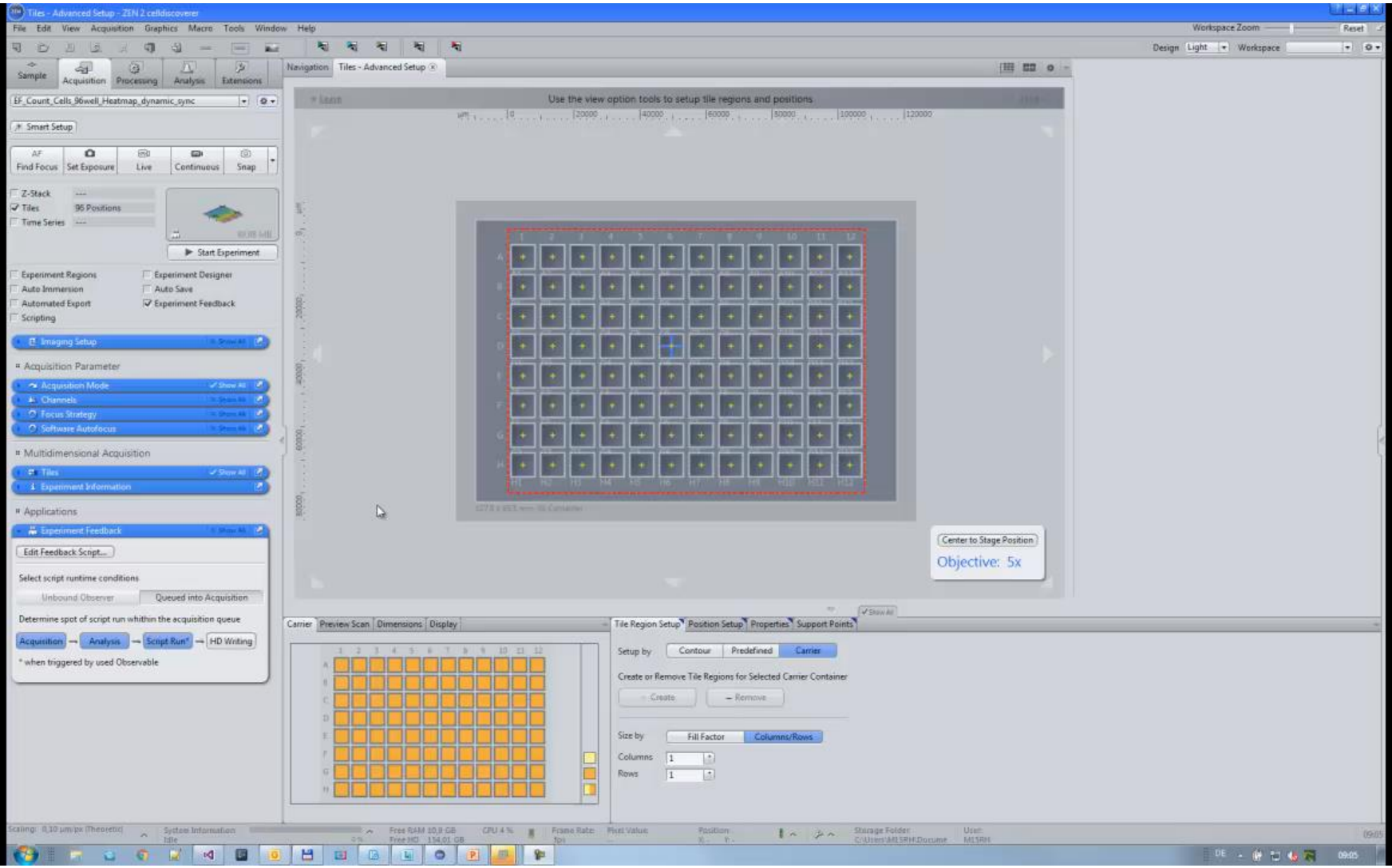
Experiment Feedback Example

Count objects or measure parameters online



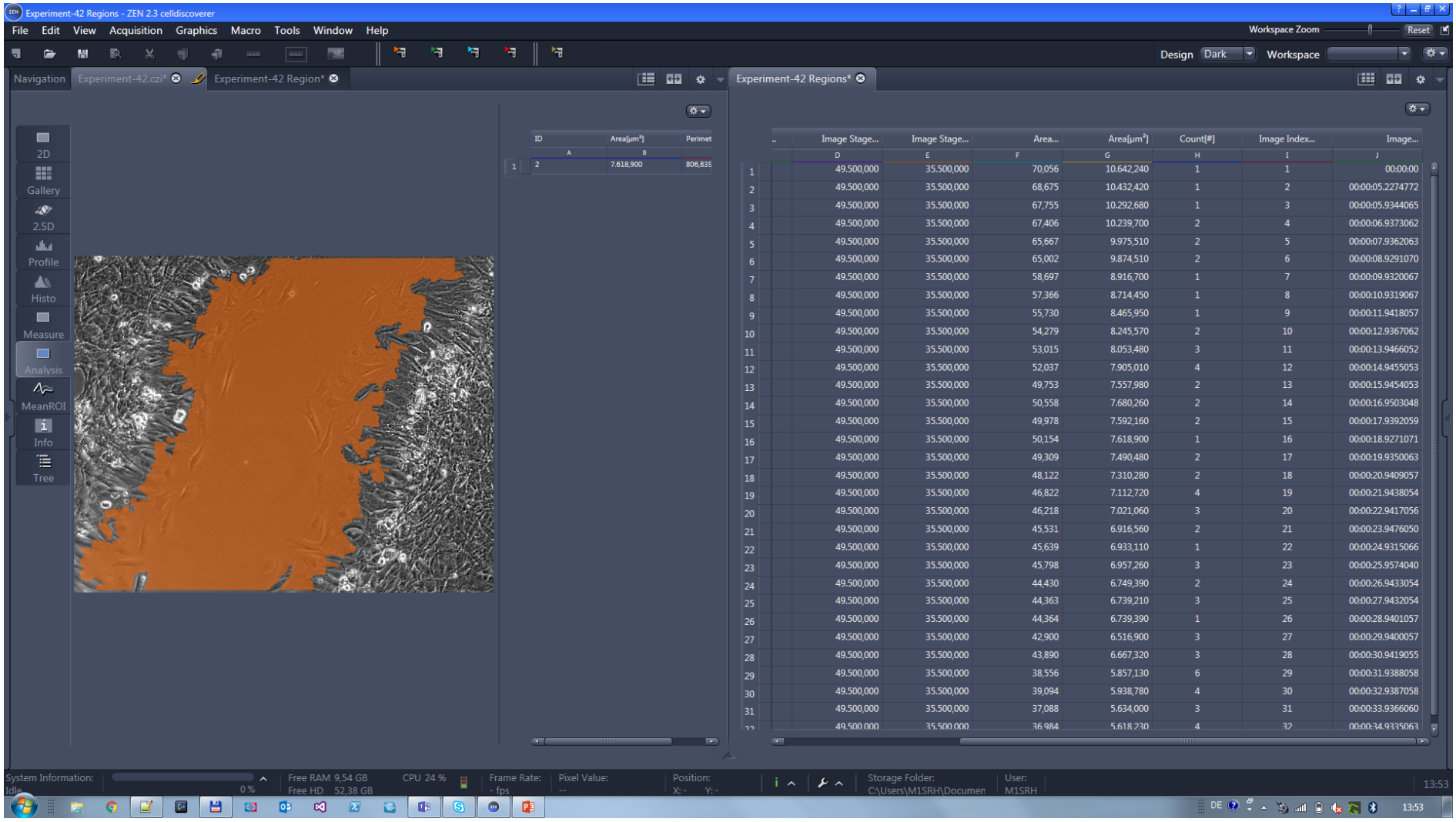
Experiment Feedback Example

Count objects or measure parameters online



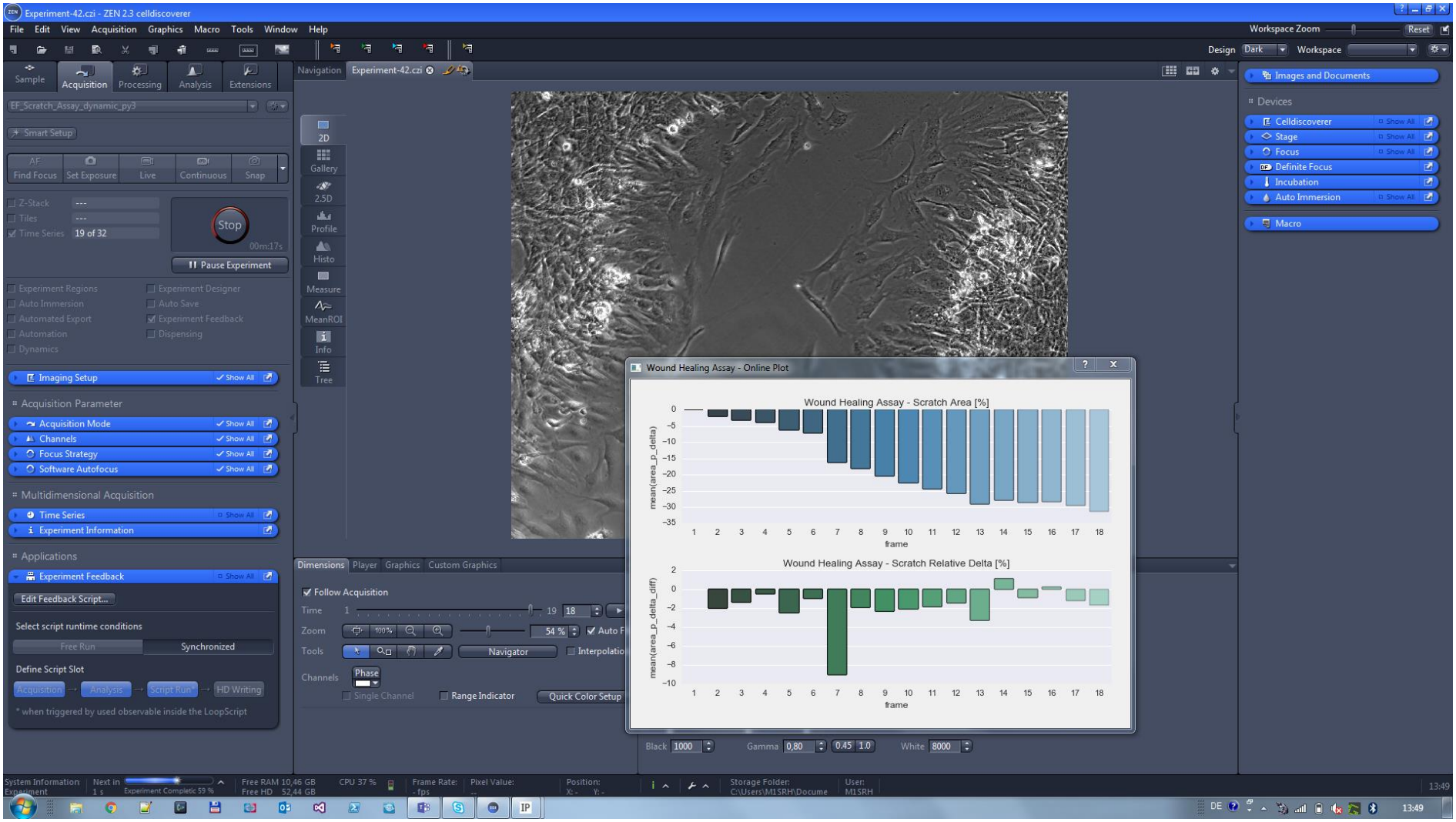
Experiment Feedback

Configure analysis offline and use it online

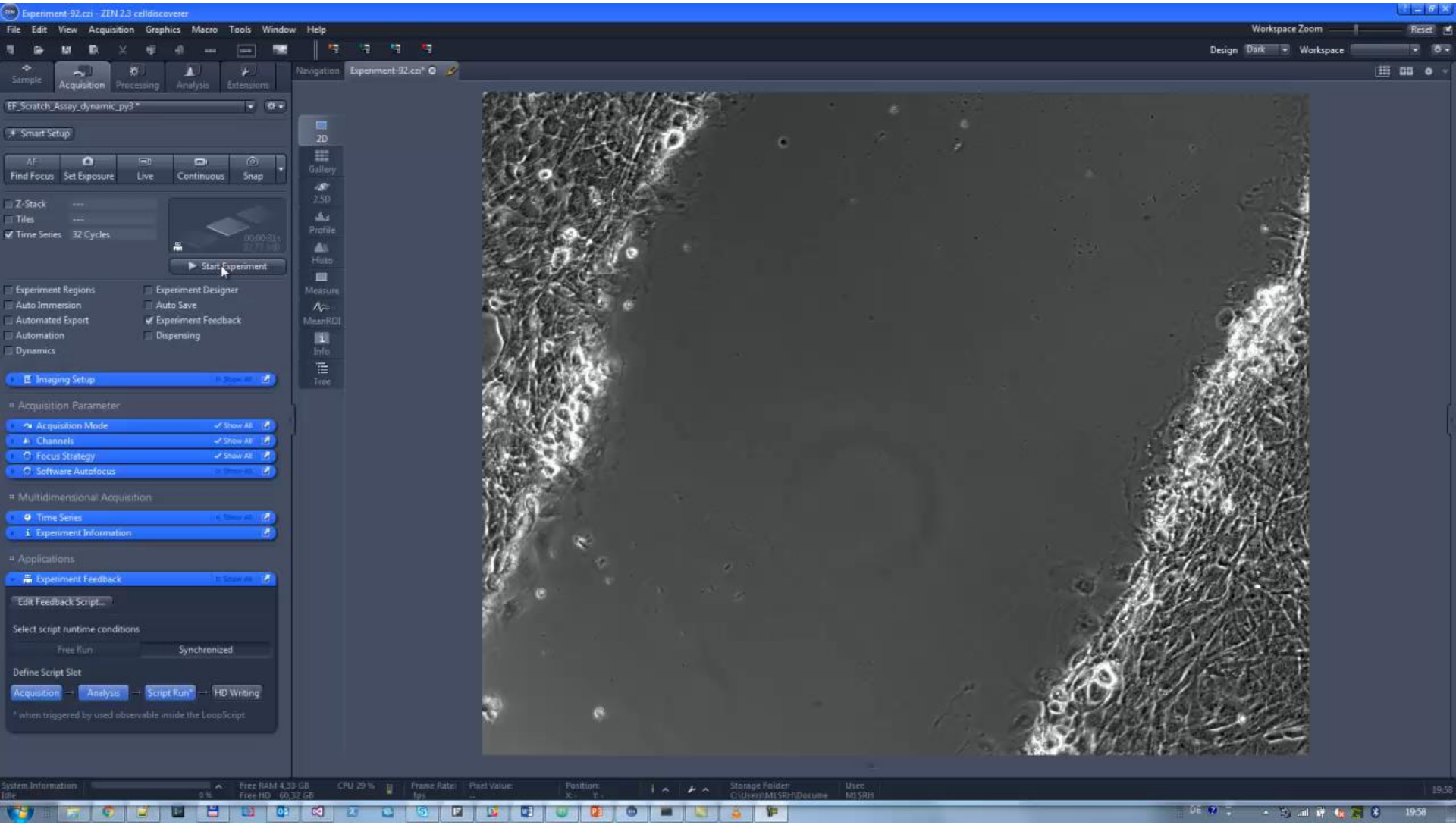


Experiment Feedback

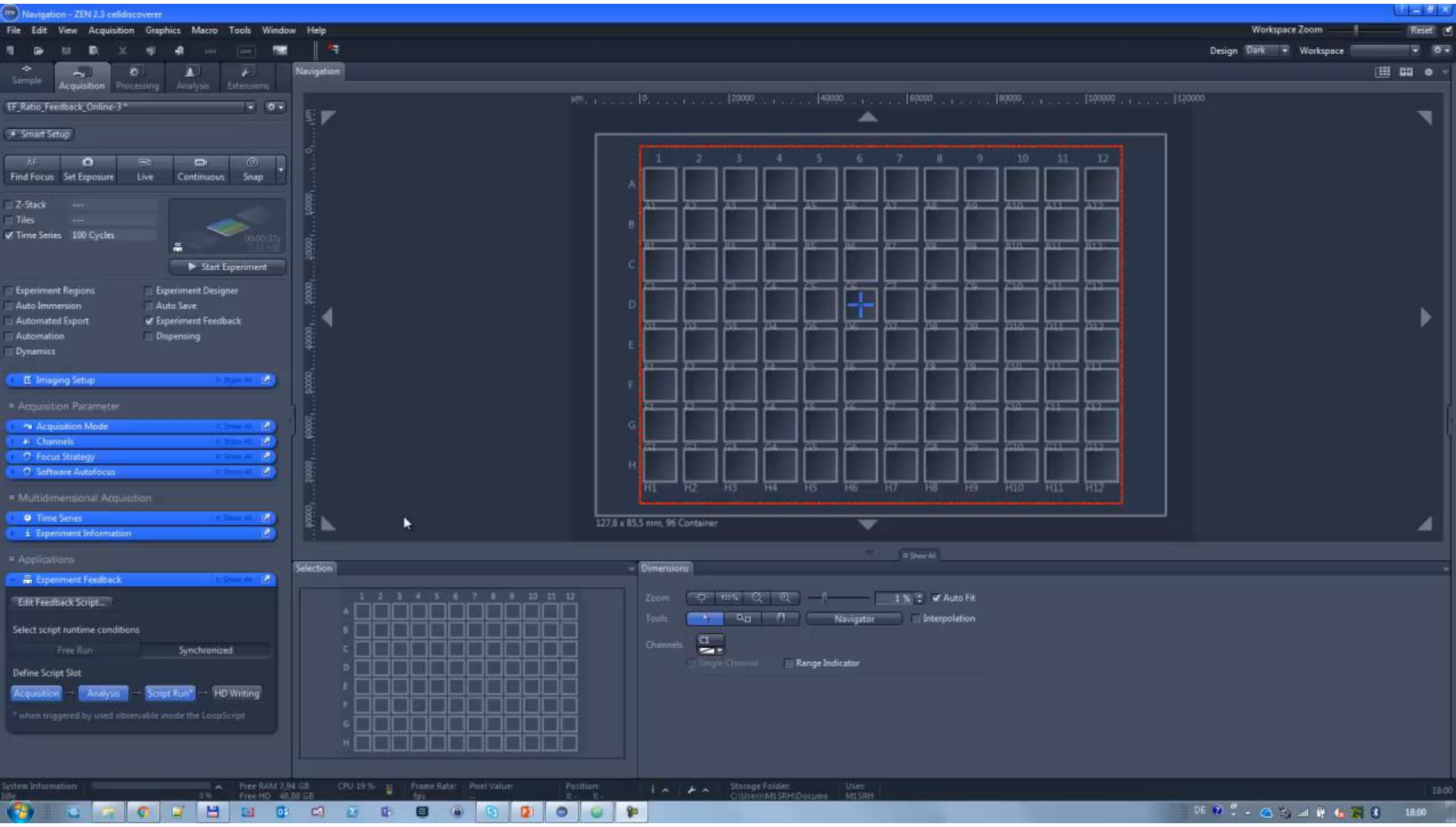
Configure analysis offline and use it online



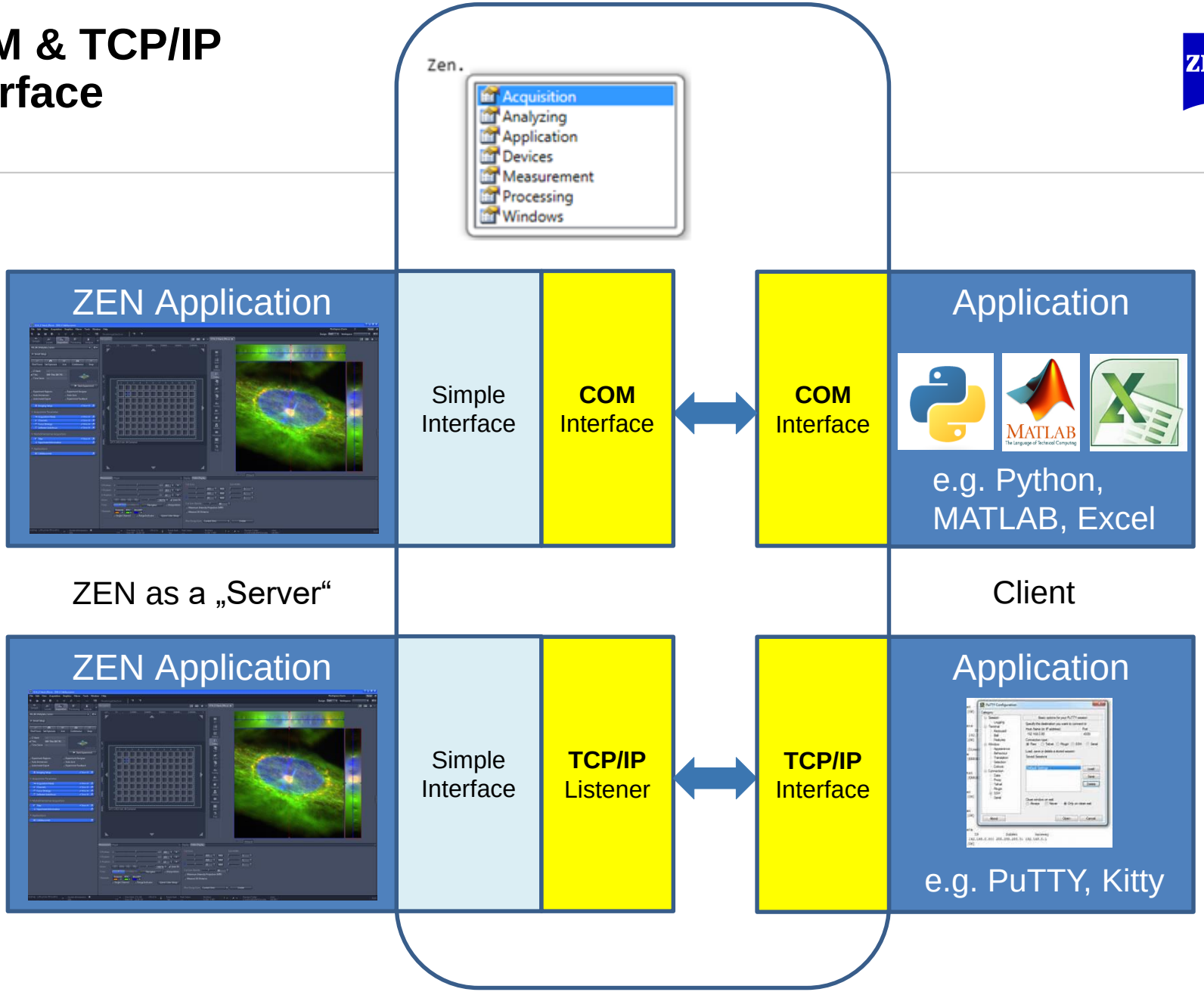
Experiment Feedback
Scratch Assay Online



Experiment Feedback
Online Ratio



COM & TCP/IP Interface



TCP / IP Interface
General Remarks

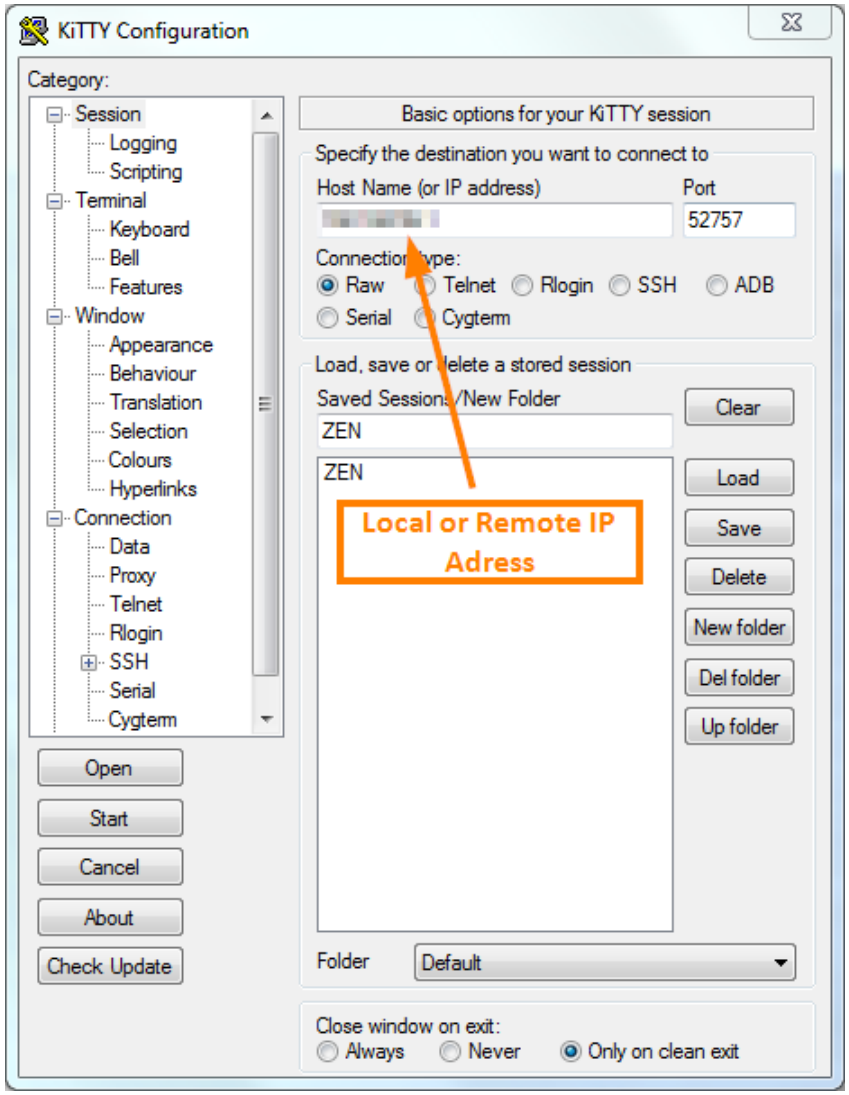
- **TCP / IP is a general interface** to the ZEN Blue software platform allowing to **combine** ZEN functionality with **any software** on **any platform**

- can be used TCP / IP **locally** as an alternative to the COM interface (not every software has COM interface and Microsoft does not allow COM between two .NET applications for unknown reasons …)

- can be used TCP / IP to **drive ZEN from another computer** that has an **arbitrary operating system** and **any programming language**

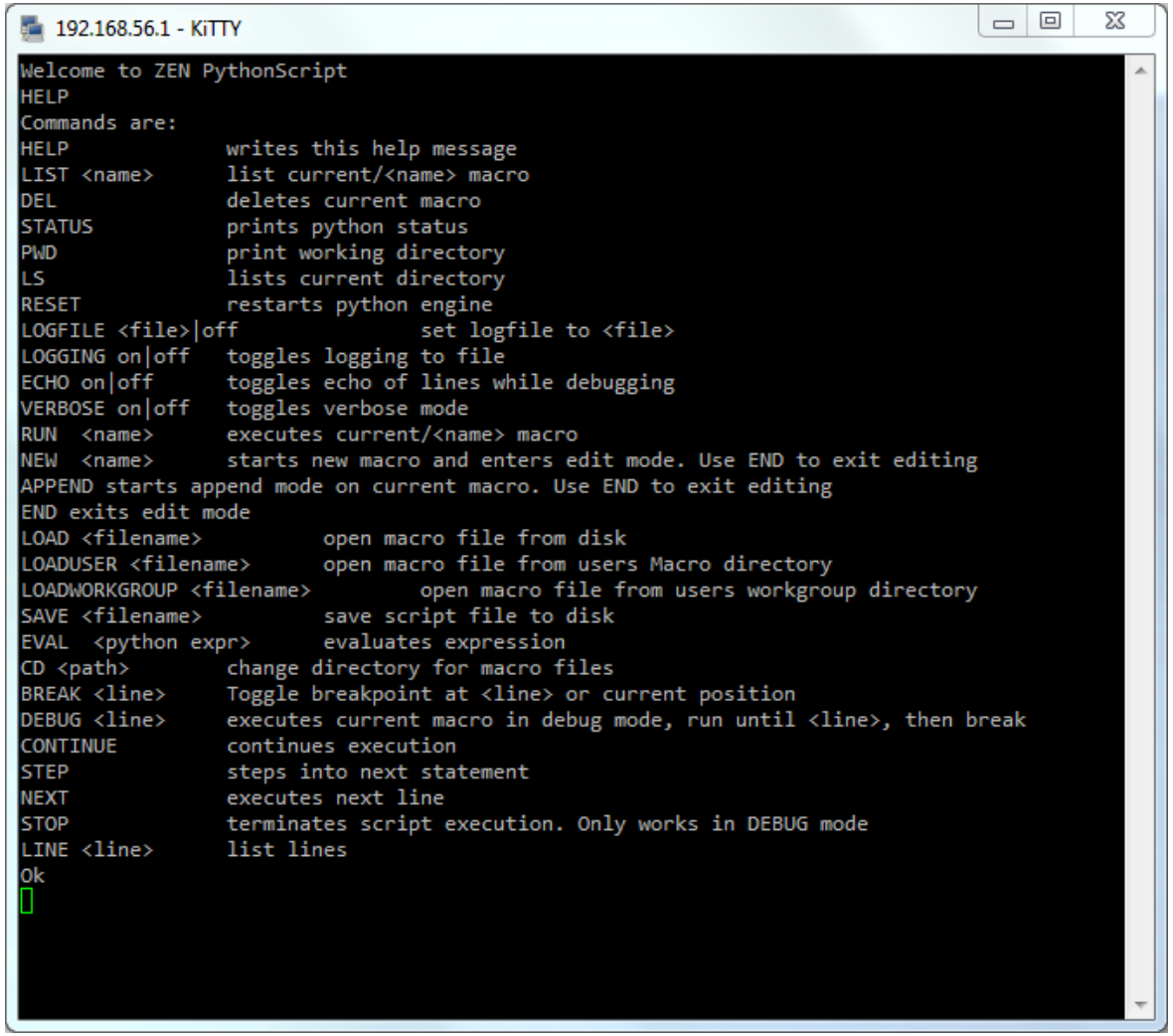
- **both systems have to be connected via a TCP / IP network** and the 2nd client system must be able to send and receive messages via TCP / IP

TCP / IP Interface
Simple Example



- **Kitty** is just an example for a simple client application
- local or remote IP address must be entered
- Connection must be configure correctly
- In case of the remote use case, firewalls etc. must be configured accordingly

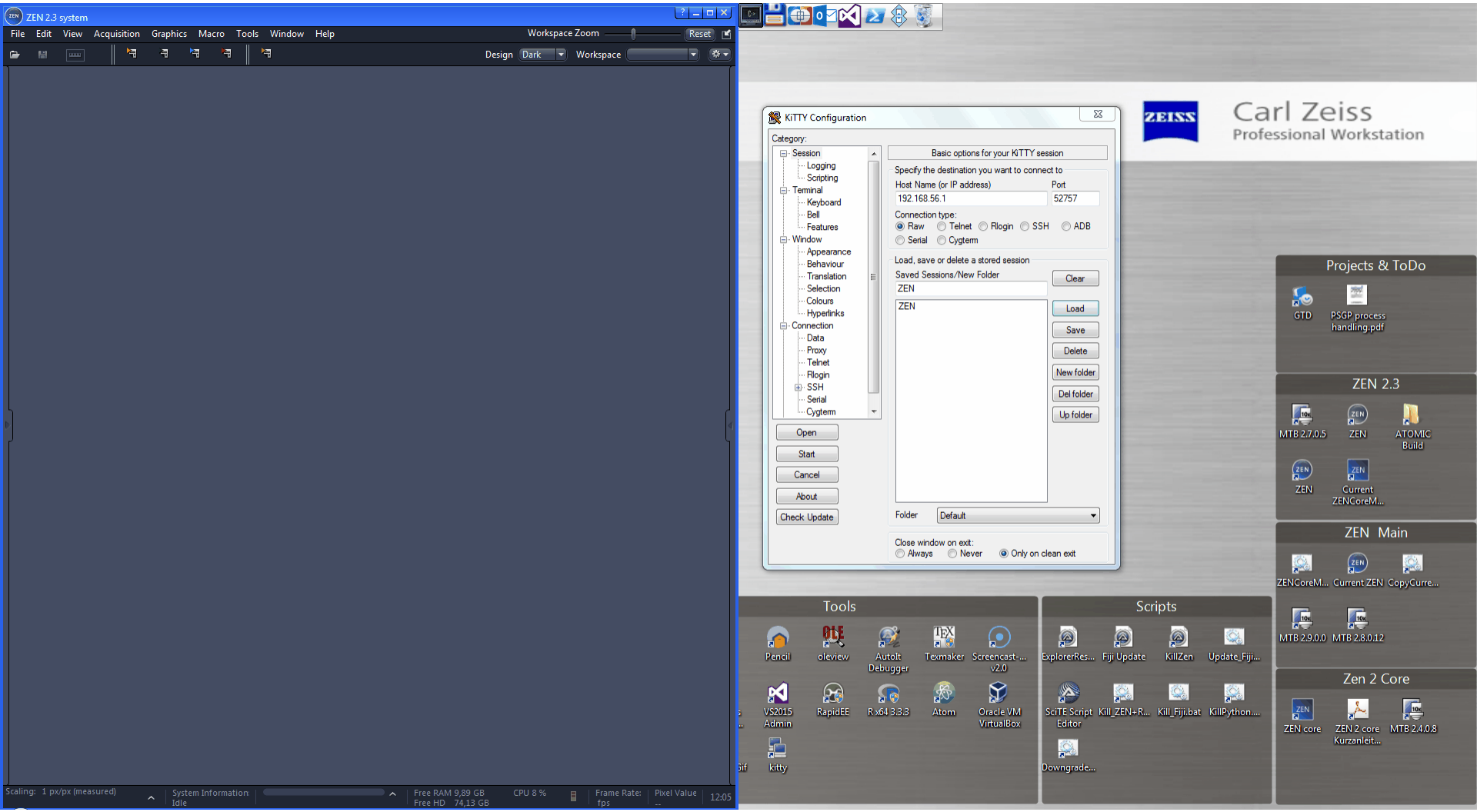
TCP / IP Interface
Simple Example



- send single OAD commands
- Load or create new OAD macros
- execute entire OAD macros

TCP / IP Interface

Simple Example: Open Image



TCP / IP Interface

Control ZEN from the „outside“

tcpip_runscript.py

```python
# coding: utf-8

import tcpoad as tcp

timeout = 100
zentcpip = tcp.tcp_open_port(timeout=timeout)

runpy = True
runczmac = True

if runpy:
    # specify python script (*.py) containg the OAD commands
    inputfile = r'c:\Users\m1srh\OneDrive - Carl Zeiss AG\Python_Projects\ZEN_TCPIP\
    oadcommandlist = tcp.create_oadlist_frompy(inputfile, extension='.py')

if runczmac:
    # define OAD macro
    inputfile = r'c:\Users\m1srh\OneDrive - Carl Zeiss AG\Python_Projects\ZEN_TCPIP\
    # convert it to text
    oadpyscript = tcp.read_macro(inputfile)
    # creat a list of tuples out of that
    oadcommandlist = tcp.create_oadlist_fromczmac(oadpyscript)

# iterate over list and send the OAD macro line-by-line
for oadcommand in oadcommandlist:
    # print the current command
    print(oadcommand)
    # and execute
    rt, an = tcp.tcp_eval_expression_and_wait_for_ok(zentcpip, oadcommand, timeout)

# finish and close connection
zentcpip.close()
```

tcpoad.py

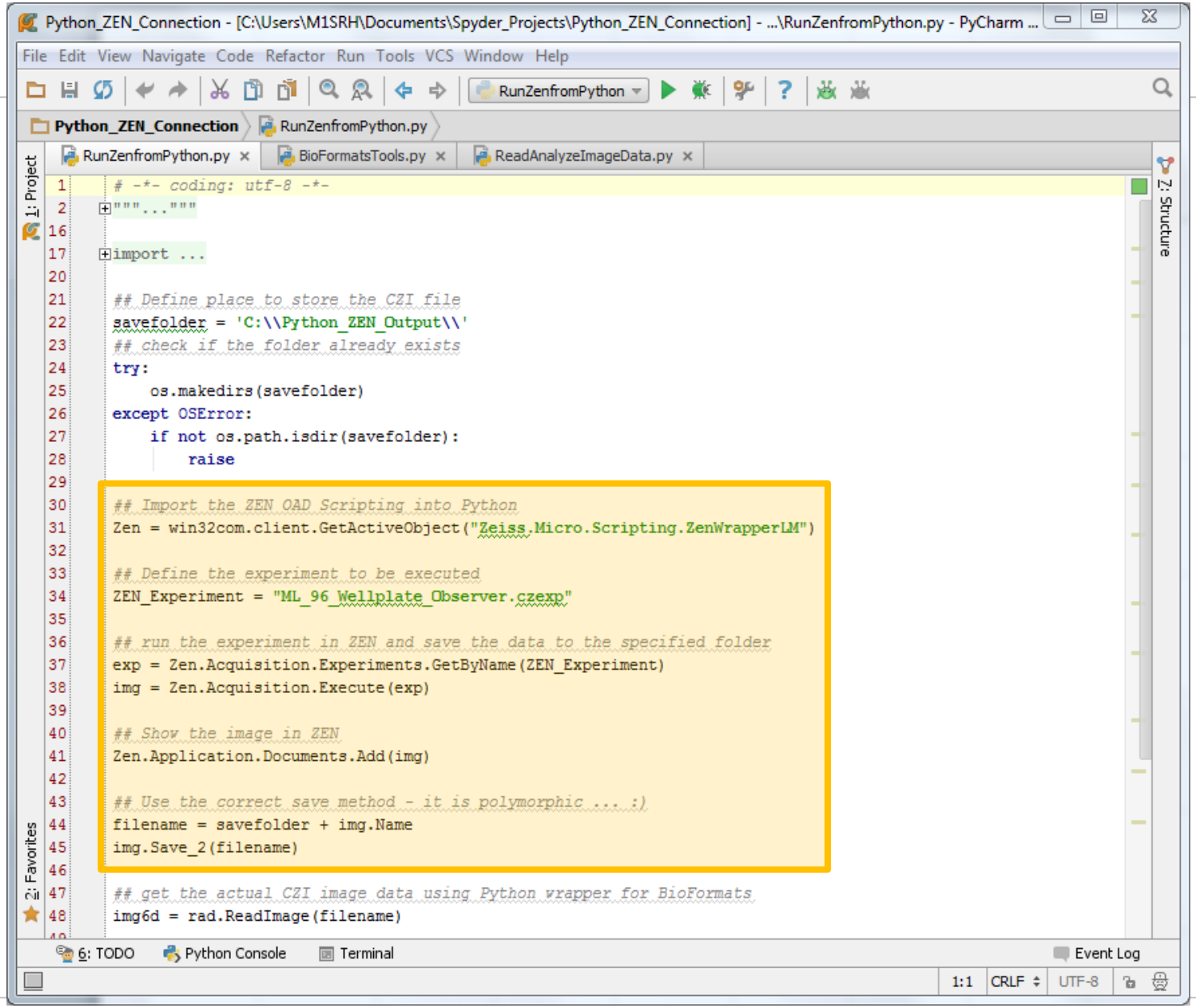
```python
Date: 06.11.2017
Version. 1.1

Some useful TCP OAD helpers
"""

import os
import telnetlib
import time
import sys
from xml.etree.cElementTree import Element, ElementTree


def tcp_open_port(timeout=60, port=52757):


def tcp_eval_expression(telnet, expression):

    # Evaluate the given python expression within ZEN
    telnet.write(("EVAL " + expression).encode('ascii'))


def tcp_eval_expression_and_wait_for_ok(telnet, expression, timeout):


def tcp_read_all(telnet):


def tcp_read_answer(telnet, timeout=0.1):


def tcp_clear_all(telnet):
```

zo05n002qg M1SRH 14:03:31 pm

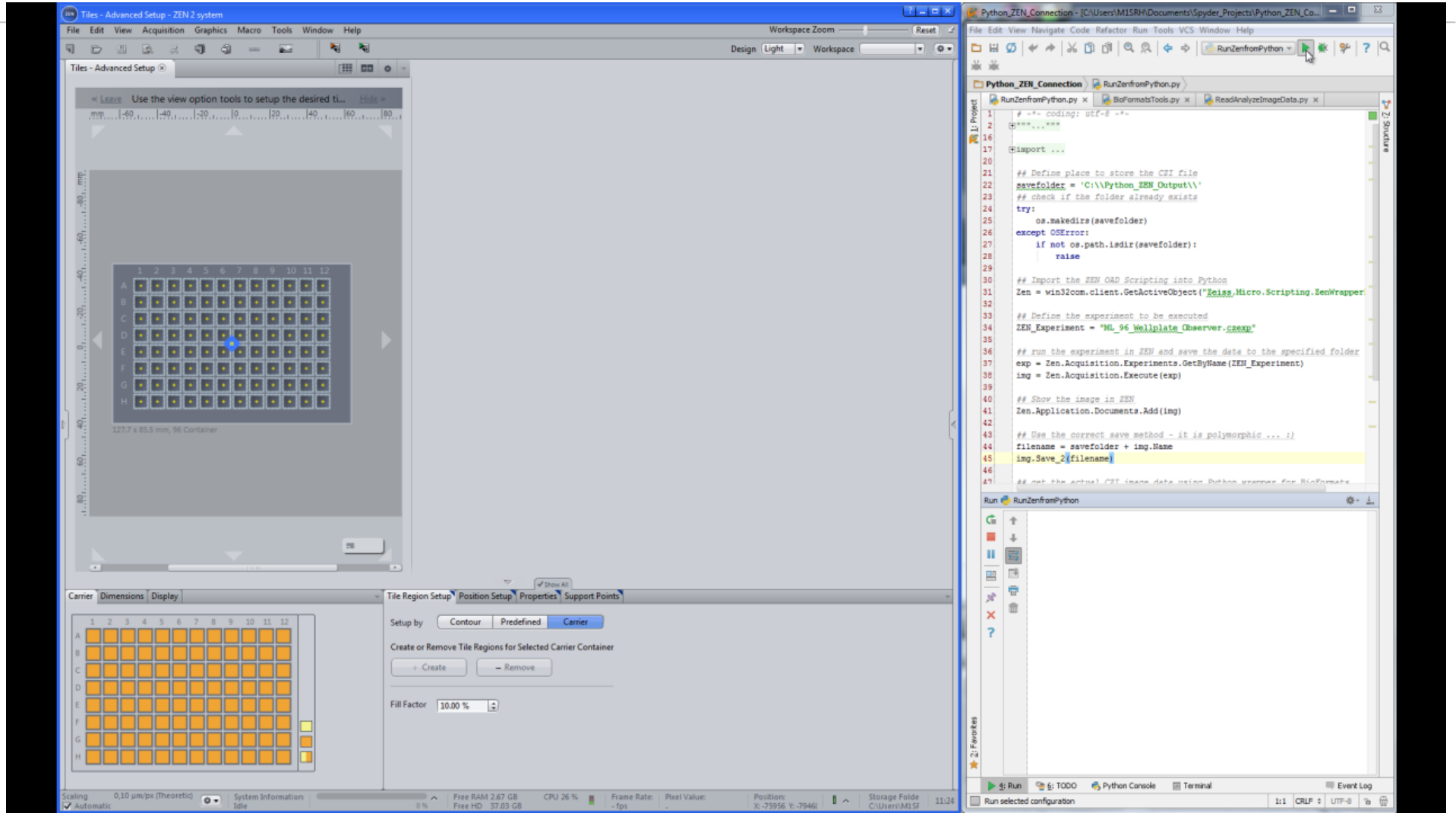
C:\Users\m1srh\OneDrive - Carl Zeiss AG\Python_Projects\ZEN_TCPIP\tcpoad.py 1:1 1 CRLF 1 deprecation UTF-8 Python 0 files

Sending single commands or whole scripts from the „outside“

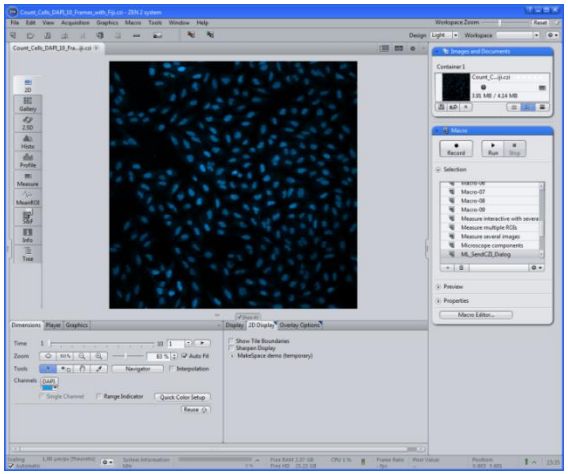
ZEN Remote via COM Interface – Python & MATLAB

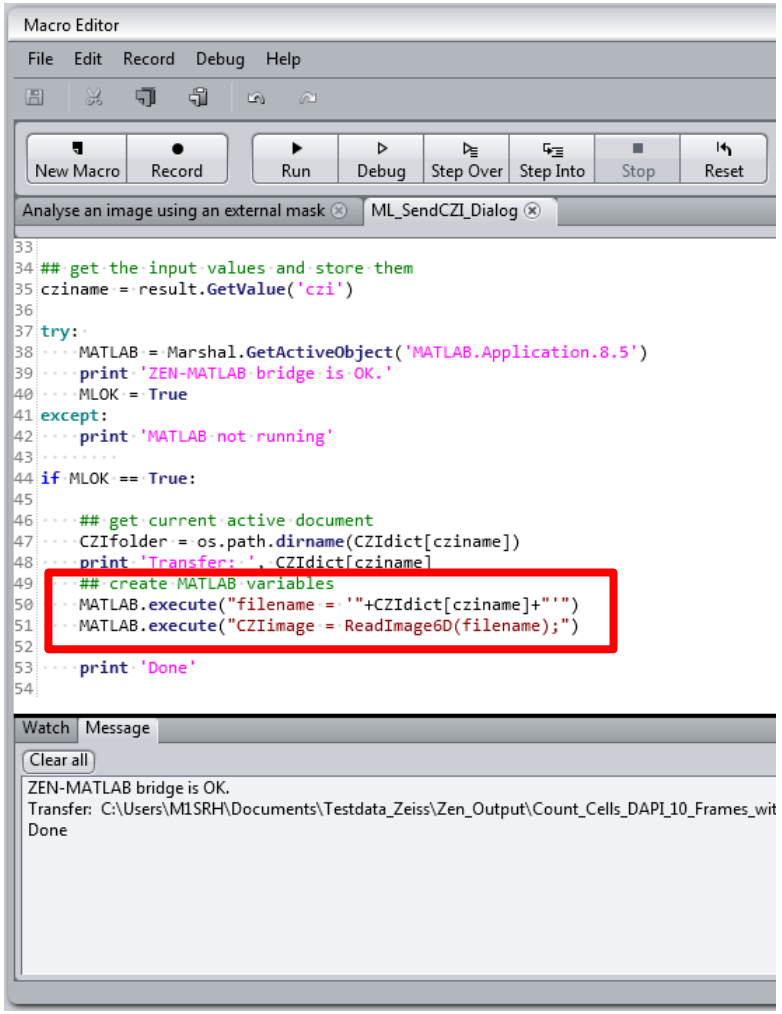


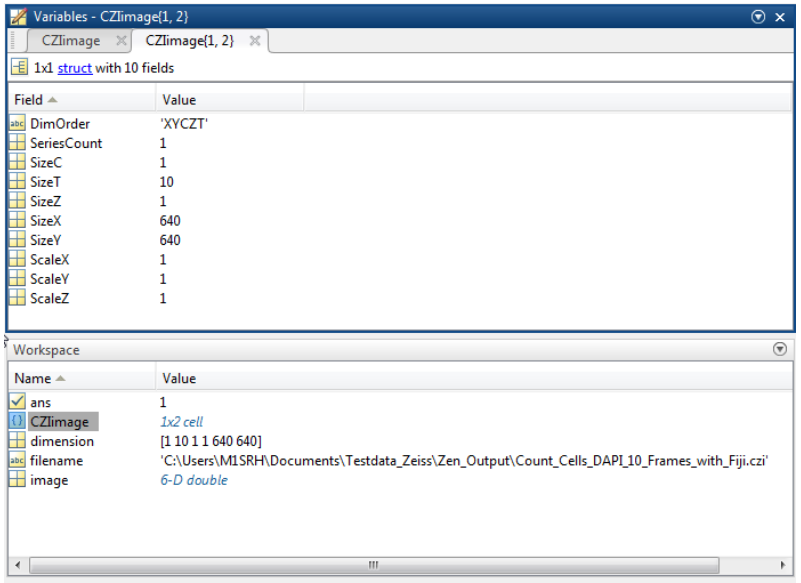
ZEN Remote via COM Interface – Python & MATLAB



Run MATLAB code from within ZEN

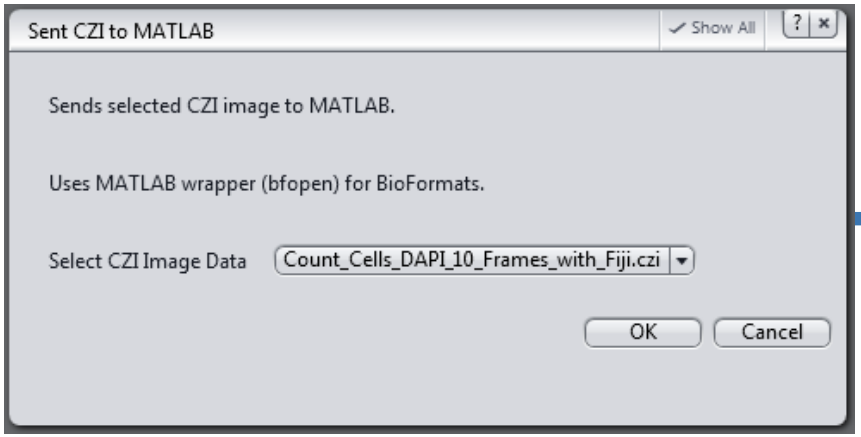


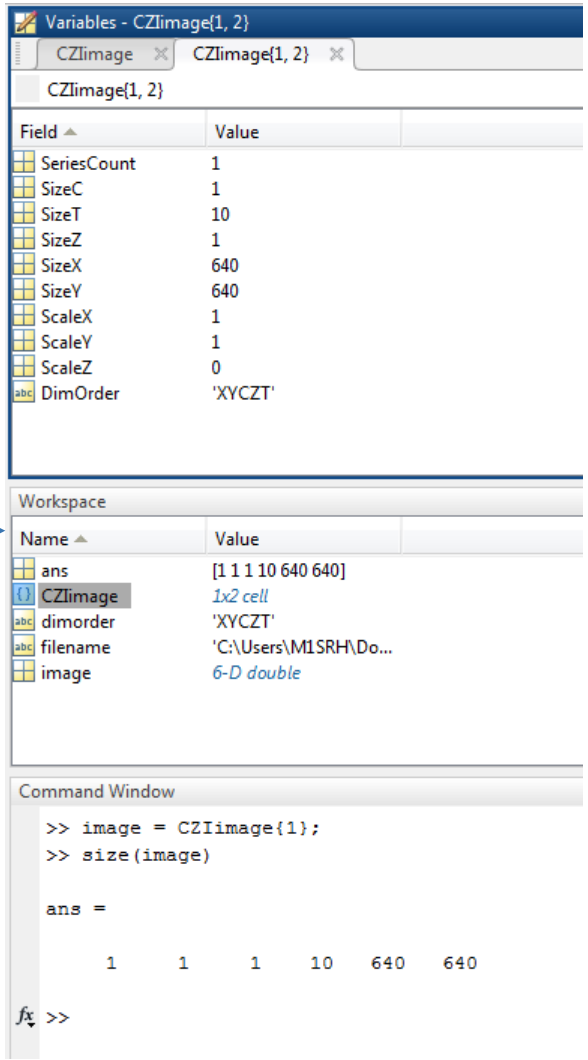




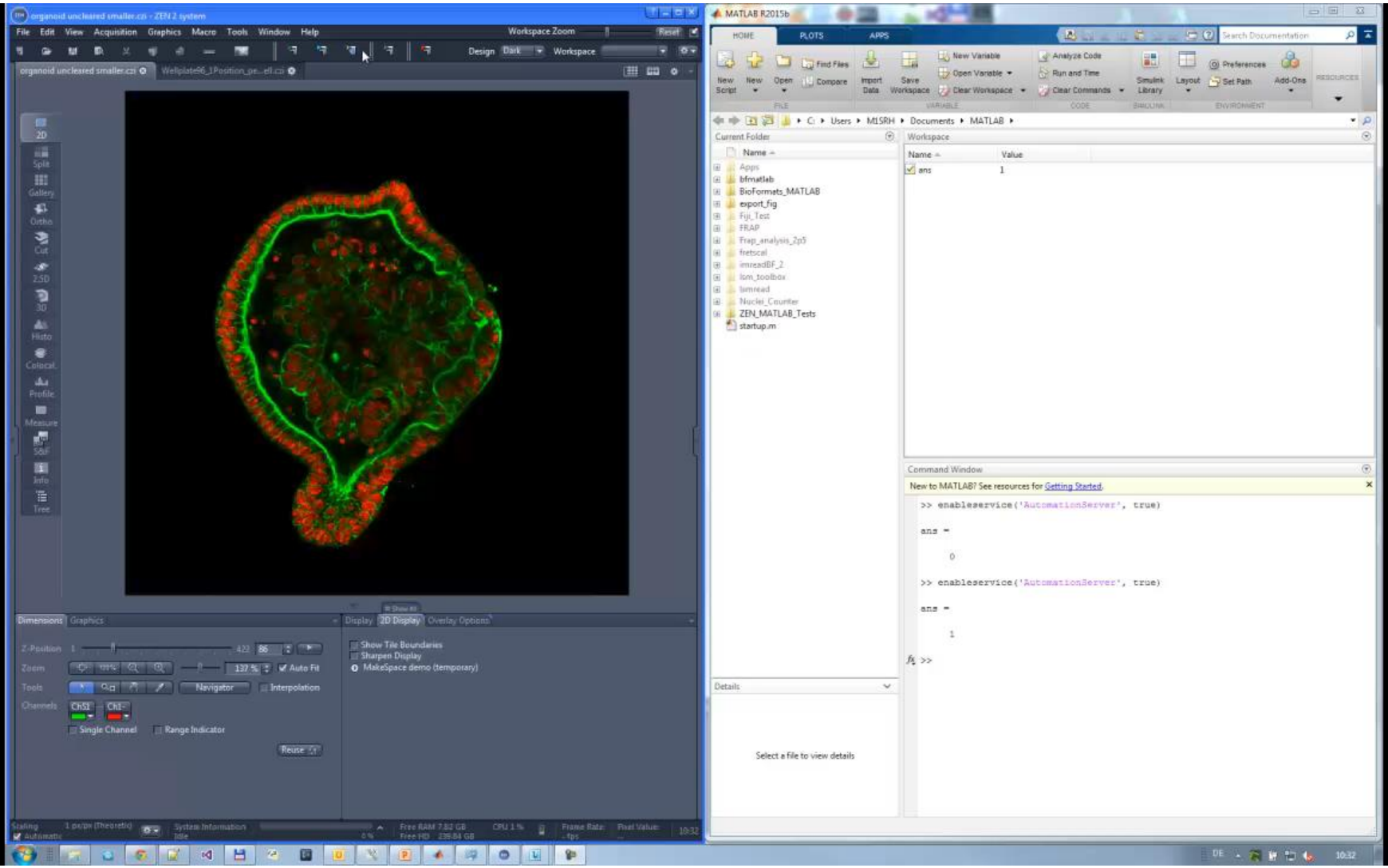
Run MATLAB code from within ZEN

OAD Macro → Dialog





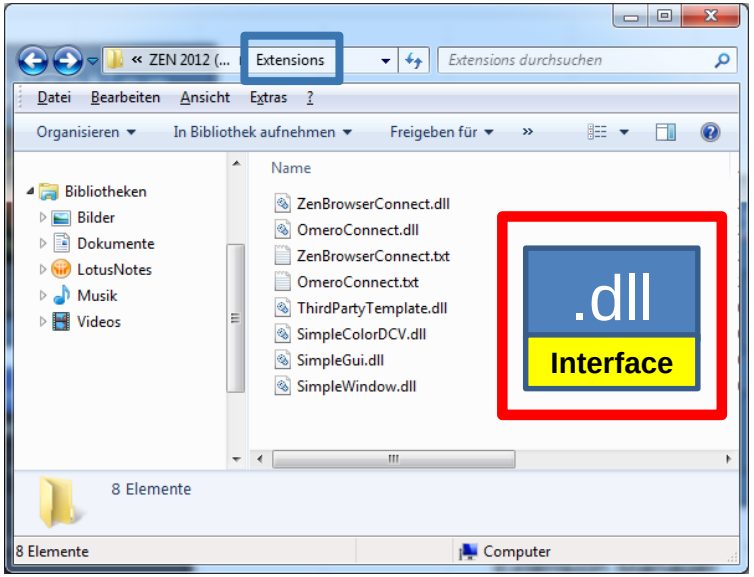
Run MATLAB code from within ZEN



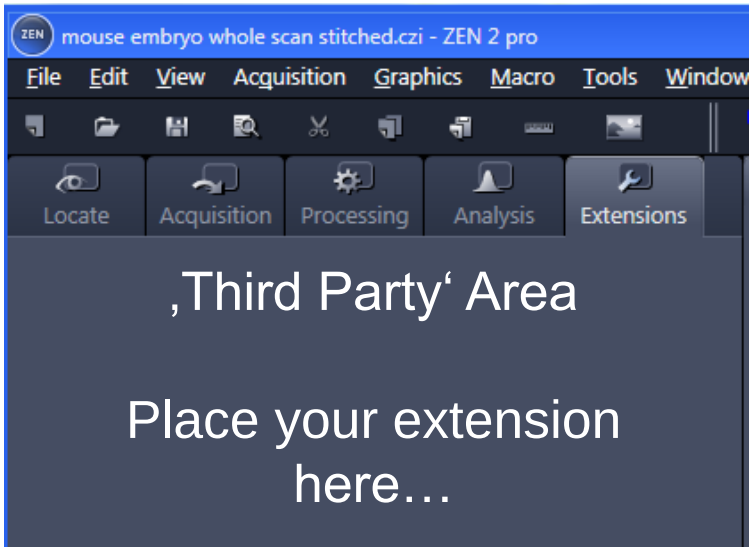
ZEN Extensions Interface for Advanced Topics

SharpDevelop

Visual Studio



ZEN - LeftToolArea

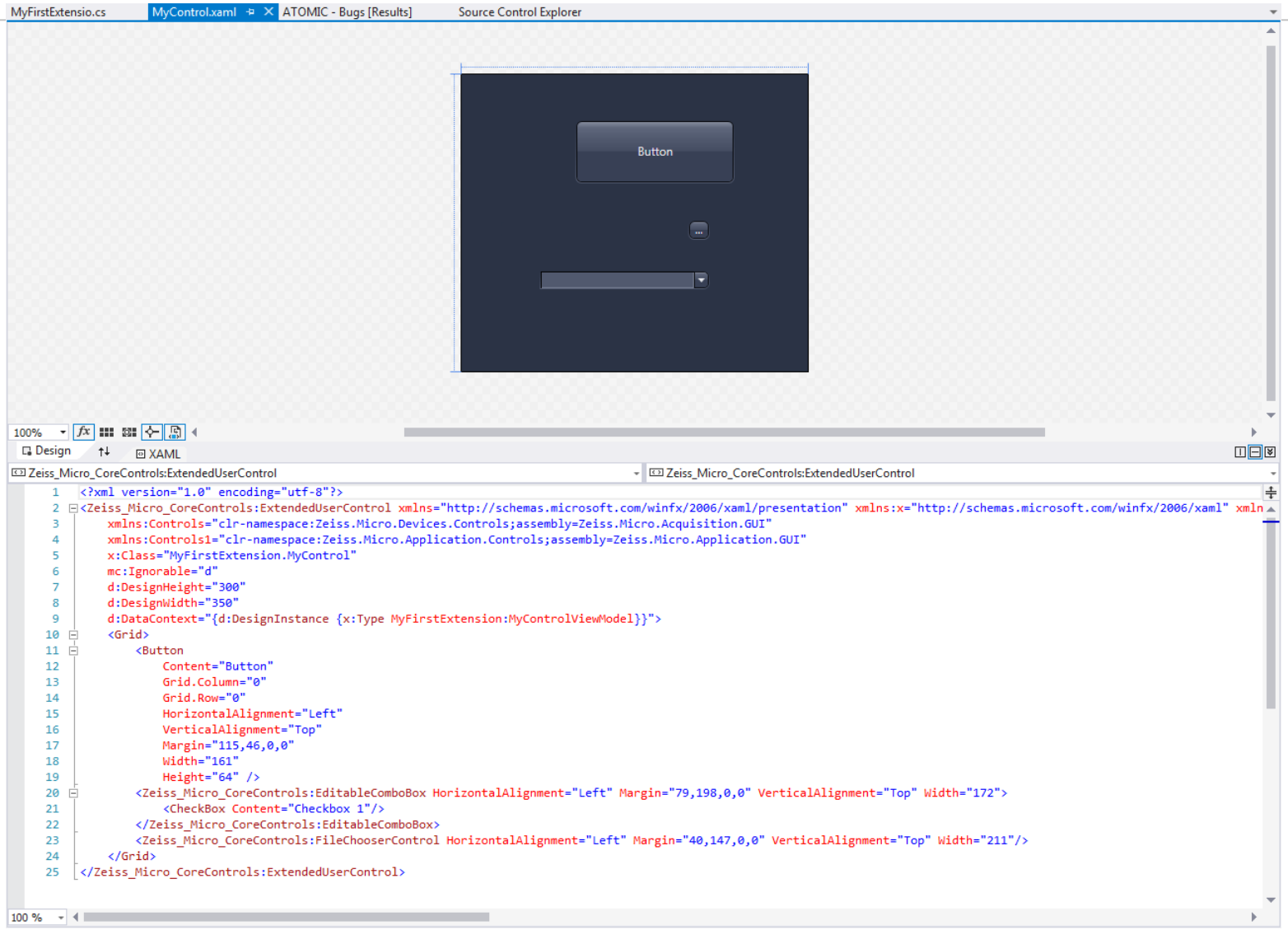


ZEN - Extension Manager

Advanced Topics – ZEN Extensions

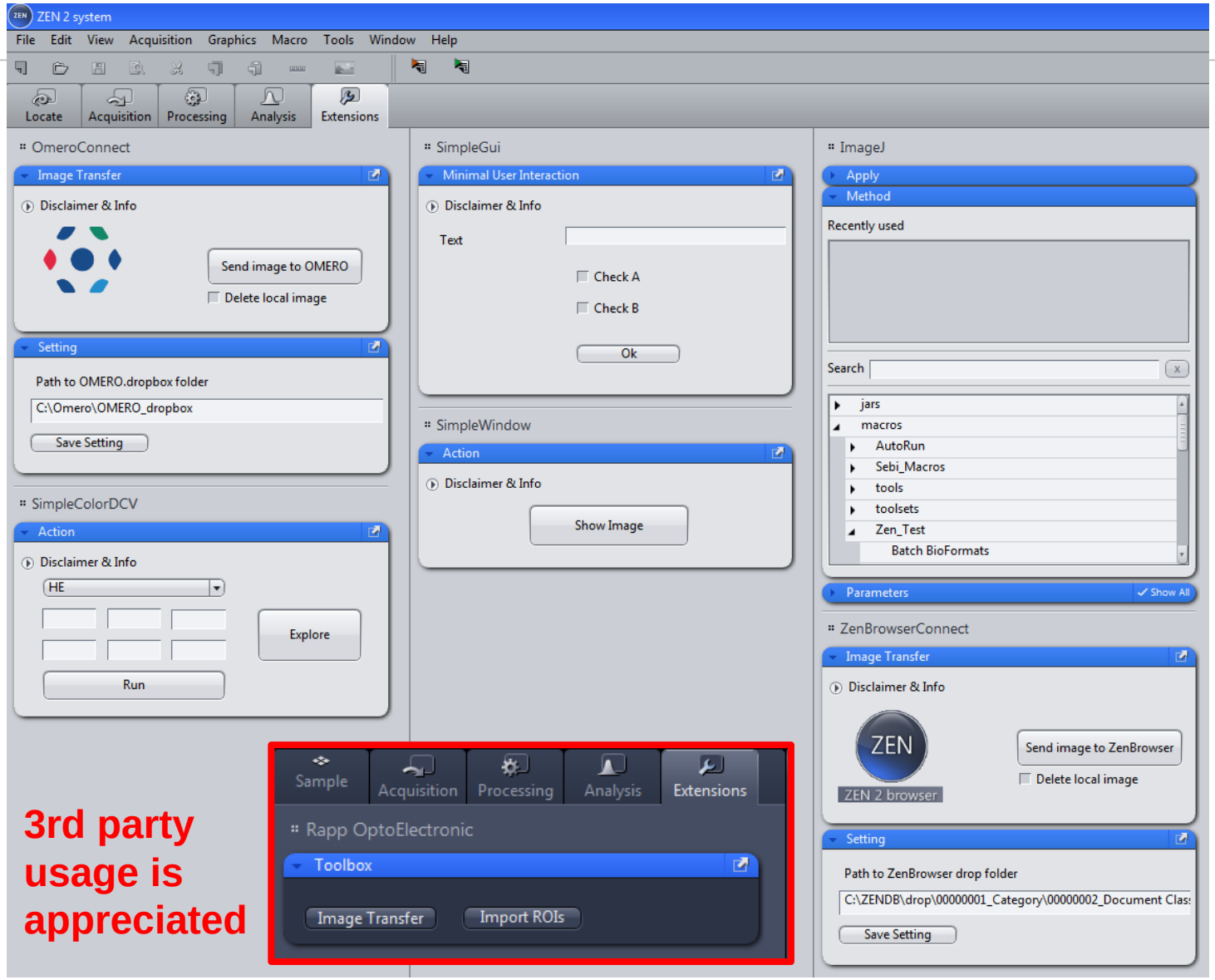
```
namespace MyFirstExtension
{
    using Zeiss.Micro;
    using Zeiss.Micro.Reflection;
    using Zeiss.Micro.Scripting;
    using Zeiss.Micro.Application.Configuration;
    using Zeiss.Micro.Diagnostics;

    [ApplicationExtension("6CB9D0FC-0EFA-4B74-A8D4-17411DB835CF", "MyFirstExtension",
        ExtensionFlags.Free | ExtensionFlags.SetupUI, "First Extension for Demo")]

    internal class MyFirstExtension : ApplicationExtension
    {
        readonly ZenWrapperLM zen = ZenWrapperLM.Instance;

        const string ExtensionName = "MyFirstExtension";
        const string MyControlName = "First Extension";
        MyControlViewModel myControlViewModel;

        // The functions to Initialize/Uninitialize the Extension are responsible to add and remove ZEN-specific WPF windows.
        // Therefore we have to add as many controls and their Action Classes to the project as we will need.
        protected override void InitializeCore(object application,
                                               ExtensionInitializeMode extensionInitializeMode,
                                               object parameters)
        {
            this.myControlViewModel = new MyControlViewModel();
            // add extension to ZEN when activated inside the extension manager
            UIToolItem toolAction = zen.Windows.AddToolWindow
                (ExtensionName, MyControlName, "MyFirstExtension; MyFirstExtension.MyControl", this.myControlViewModel);
            // this parameter manages whether the control will be expanded when the extension is started
            toolAction.IsExpanded = true;
        }

        protected override void UninitializeCore(ExtensionUninitializeMode extensionUninitializeMode)
        {
            // add extension to ZEN when activated inside the extension manager
            zen.Windows.RemoveToolWindow(ExtensionName, MyControlName);
        }

    }
}
```

Advanced Topics – ZEN Extensions GUI Design




```
<?xml version="1.0" encoding="utf-8"?>
<Zeiss_Micro_CoreControls:ExtendedUserControl xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmln
    xmlns:Controls="clr-namespace:Zeiss.Micro.Devices.Controls;assembly=Zeiss.Micro.Acquisition.GUI"
    xmlns:Controls1="clr-namespace:Zeiss.Micro.Application.Controls;assembly=Zeiss.Micro.Application.GUI"
    x:Class="MyFirstExtension.MyControl"
    mc:Ignorable="d"
    d:DesignHeight="300"
    d:DesignWidth="350"
    d:DataContext="{d:DesignInstance {x:Type MyFirstExtension:MyControlViewModel}}">
    <Grid>
        <Button
            Content="Button"
            Grid.Column="0"
            Grid.Row="0"
            HorizontalAlignment="Left"
            VerticalAlignment="Top"
            Margin="115,46,0,0"
            Width="161"
            Height="64" />
        <Zeiss_Micro_CoreControls:EditableComboBox HorizontalAlignment="Left" Margin="79,198,0,0" VerticalAlignment="Top" Width="172">
            <CheckBox Content="Checkbox 1"/>
        </Zeiss_Micro_CoreControls:EditableComboBox>
        <Zeiss_Micro_CoreControls:FileChooserControl HorizontalAlignment="Left" Margin="40,147,0,0" VerticalAlignment="Top" Width="211"/>
    </Grid>
</Zeiss_Micro_CoreControls:ExtendedUserControl>
```

ZEN Extensions - Examples



We make it visible.

www.zeiss.com/czi

www.zeiss.com/zen-oad

Tutorials

Application Notes

Examples

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

http://github.com/zeiss-microscopy/libCZI