

Guided Acquisition

Advanced Automated Microscopy



Carl Zeiss Microscopy GmbH
Carl-Zeiss-Promenade 10
07745 Jena
Germany

microscopy@zeiss.com
www.zeiss.com/microscopy

Carl Zeiss Microscopy GmbH
ZEISS Group
Kistlerhofstr. 75 81379 München
Germany

Effective from: 01 / 2018

©2018 This document or any part of it must not be translated, reproduced, or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information or retrieval system. Violations will be prosecuted. The use of general descriptive names, registered names, trademarks, etc. in this document does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use. Software programs willfully remain the property of ZEISS. No program, documentation, or subsequent upgrade thereof may be disclosed to any third party, unless prior written consent of ZEISS has been procured to do so, nor may be copied or otherwise duplicated, even for the customer's internal needs, apart from a single back-up copy for safety purposes.

ZEISS reserves the right to make modifications to this document without notice.

Contents

1	Introduction	4
1.1	Basic Workflow Definition	5
1.2	General Workflow Definition	6
1.3	General Workflow - Complete Overview	8
2	Workflow - Single Steps	9
2.1	Acquire Sample Data with Overview Scan	9
2.2	Create Image Analysis Pipeline	10
2.3	Define Overview Experiment	13
2.4	Define Detailed Scan Experiment	14
3	Automation via OAD	15
3.1	Prerequisites	15
3.2	Pure Scripting or User Dialog	15
3.3	User Dialog Requirements	15
3.4	Prerequisites	16
3.5	Create the User Dialog	19
3.6	Overview Scan Experiment	21
3.7	Find the interesting Objects	22
3.8	Modify the Tile Dimensions	24
3.9	Run the Detailed Scan	25
4	Testrun with FluoCells Slide	27
5	Test Run with Brain Slide	29
6	Mitosis Detection with Camera and LSM	31
7	Appendix	33
7.1	Complete Workflow Diagram	33
7.2	Python Script: Guided_Acquisition_shortUI.py	34
8	ToDo's and Limitations	41
9	Disclaimer	41

1 Introduction

For a growing number of applications, it will be crucial to acquire data in a smart way. One way to achieve this goal is to build a smart microscope, which essentially means creating smart software workflows to control the hardware based on image analysis results.

Idea or Task:

- Scan or inspect a large area.
- Detect an "interesting" object.
- Acquire detailed data for every event.
- Automate the workflow to minimize user interference.

First of all it is important to define what a **Object of Interest** can actually be. Example would be an object that meets specific criteria, for example:

- size
- brightness
- shape
- intensity
- combinations of the above

It could be something quite simple. For instance one can have lots of cells, that are stained with blue dye, and only a few of them (maybe where the transfection worked ...) are also expressing GFP. The idea here would be to detect all cells that are positive for both colors and acquire an z-Stack for every cell (blue & green) that meets those criteria. Therefore this kind of application requires three major tasks:

1. Define the Overview Scan Experiment.
2. Define the object detection rules, e.g. setup image analysis.
3. Define the Detailed Scan(s) to be carried out in case of a "positive" object.

1.1 Basic Workflow Definition

The main workflow can be summarized as follows:

1. Acquire some sample data showing a object of interest.
2. Setup an Image Analysis Pipeline to detect those events.
3. Define an experiment which does the Overview Scan.
4. Define an experiment which does the Detailed Scan.
5. Automate the entire workflow.

The goal of this tutorial is to create an automated workflow that can be used to easily setup a **Guided Acquisition**. This requires some knowledge about the OAD macro environment and its scripting language Python.

1.2 General Workflow Definition

This is the outline of the general workflow, which would be automated.

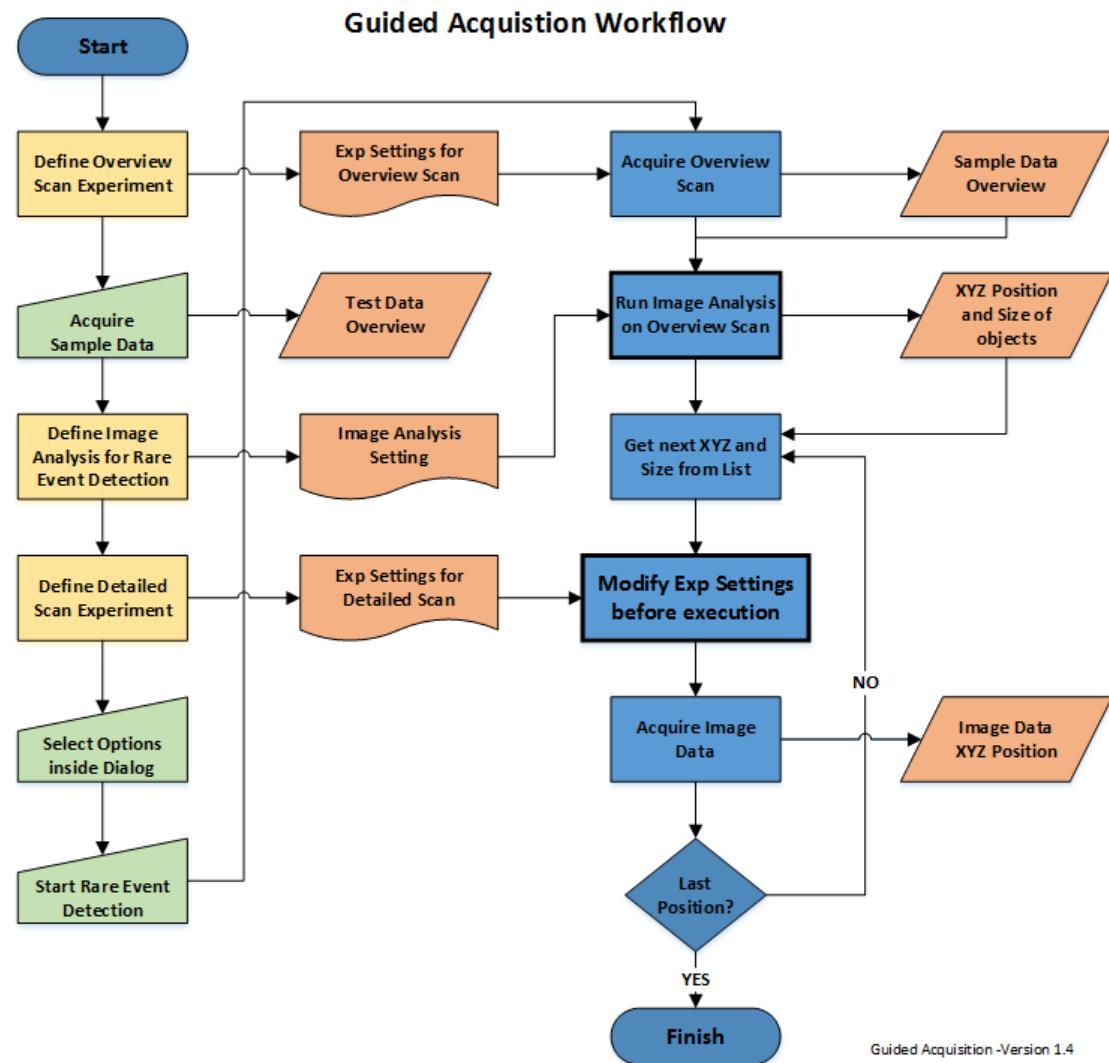


Figure 1: General workflow for Guided Acquisition.

A different way to visualize this is shown in the figure below.

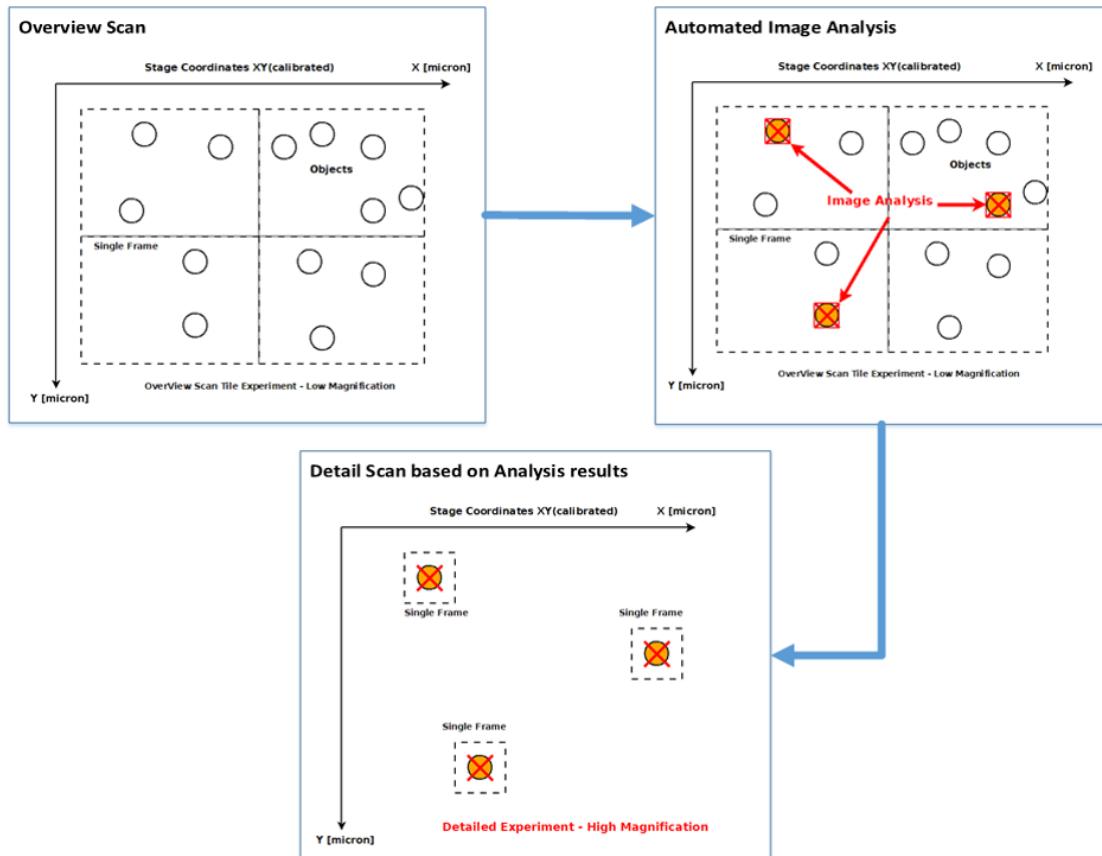


Figure 2: Larger area with some candidates for a "interesting" objects.

1.3 General Workflow - Complete Overview

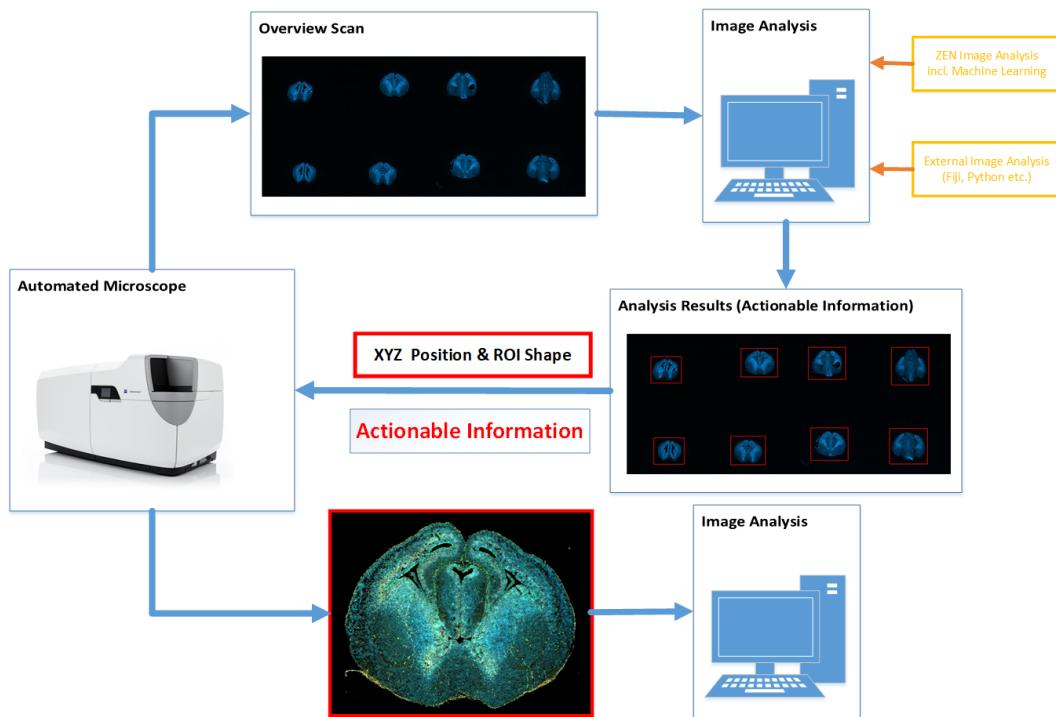


Figure 3: Workflow - Actionable Information 1.

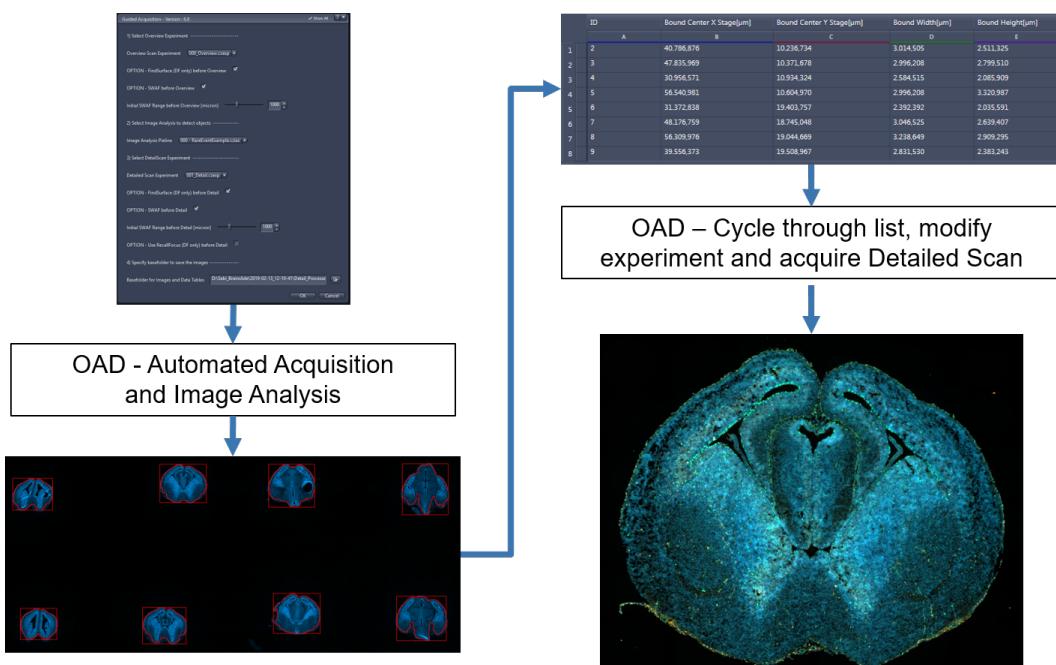


Figure 4: Workflow - Actionable Information 2

2 Workflow - Single Steps

This part of the tutorial will explain all required steps to set up the complete workflow in more detail. Some knowledge about image analysis is required here.

2.1 Acquire Sample Data with Overview Scan

The first crucial step is to have a basic idea of what the actual rare event will look like, inside an real image acquired, with the appropriate parameters (light intensity, detector settings, filters, objective, ...). An ideal sample data set should be acquired using the "real" acquisition parameters will be used to define the overview scan experiment later on.

Typically such an overview scan is acquired using a lower magnification in combination with a tile experiment. The exact parameters will be specific for the application, but the main idea is always the same:

The overview image must contain the information to locate the objects of interest based on "some" features that can be retrieved via an appropriate image analysis.

Once a representative sample data set is available, one can start setting up an image analysis pipeline. ZEN offers currently two ways do to so:

1. The easy way – Use the Analysis Wizard.
2. Program your own image analysis pipeline using an OAD macro (this is not covered by this tutorial).
3. Program your own image analysis using an external software (e.g. Fiji, Python etc.) and use it from within ZEN (this is not covered by this tutorial).

The most comfortable way is to use the wizard; this offers enough flexibility to cover most of the applications.

2.2 Create Image Analysis Pipeline

Here one already sees a "half-ready" image analysis pipeline. In order to automate, the wizard step containing the feature selection is crucial.

In order to relocate all detected objects, it is of course required to retrieve the current XY(Z) position of every detected event.

The main idea is to determine all parameters required to get access to the stage coordinates of a detected object.

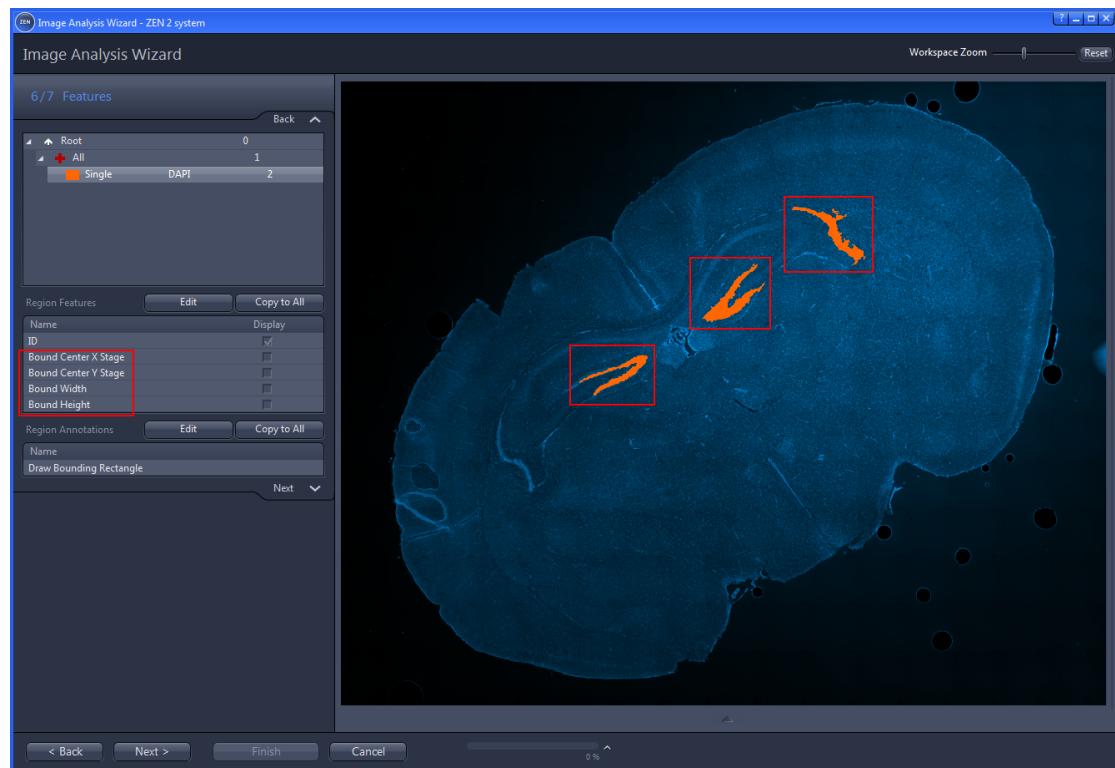


Figure 5: Select the required positional features inside the wizard.

Currently one needs to select the following parameters:

- **Object ID**
- **Bound Center Stage X**
- **Bound Center Stage Y**
- **Bound Width**
- **Bound Height**
- **ImageSceneContainerName**

The object ID is needed to keep track of the executed detail scans and to save the resulting image files with the correct ID as part of the used filename.

The parameters **Bound Center Stage X** and **Bound Stage Center Y** yield absolute stage coordinates of the detected objects. These will be used to relocate the objects for the detailed scan later on.

The parameters **Bound Width** and **Bound Height** yield in the absolute width and height of the bounding rectangle. This is required if the detected object is bigger than a single frame. In such a case the tile region of the detailed experiment will be adapted to the size of the bounding rectangle automatically.

The parameter **ImageSceneContainerName** is needed to keep track of the respective wells, because the wellID will be used as part of the filename for the detail scan. Depending on your application, it might be useful to measure additional parameters. Feel free to add whatever might be useful.

INFO:

The parameter **ImageSceneContainerName** is not required under all circumstances and only makes sense if there really is a physical well or container on your sample.

2.3 Define Overview Experiment

Usually this is done using the Tiles and Positions module to scan a large area. This tutorial assumes that one is already familiar with this ZEN module.

Important Remark: Since one wants to relocate the detected objects, it is required to calibrate the XY stage before the overview scan, and a rigid and stiff sample holder should be used.

For this tutorial one can use an already acquired image representing the real overview scan (instead of a real experiment). The shown image is a 16x13 tile image acquired with a 10X magnification. The result of such a scan followed by the image analysis is shown here:

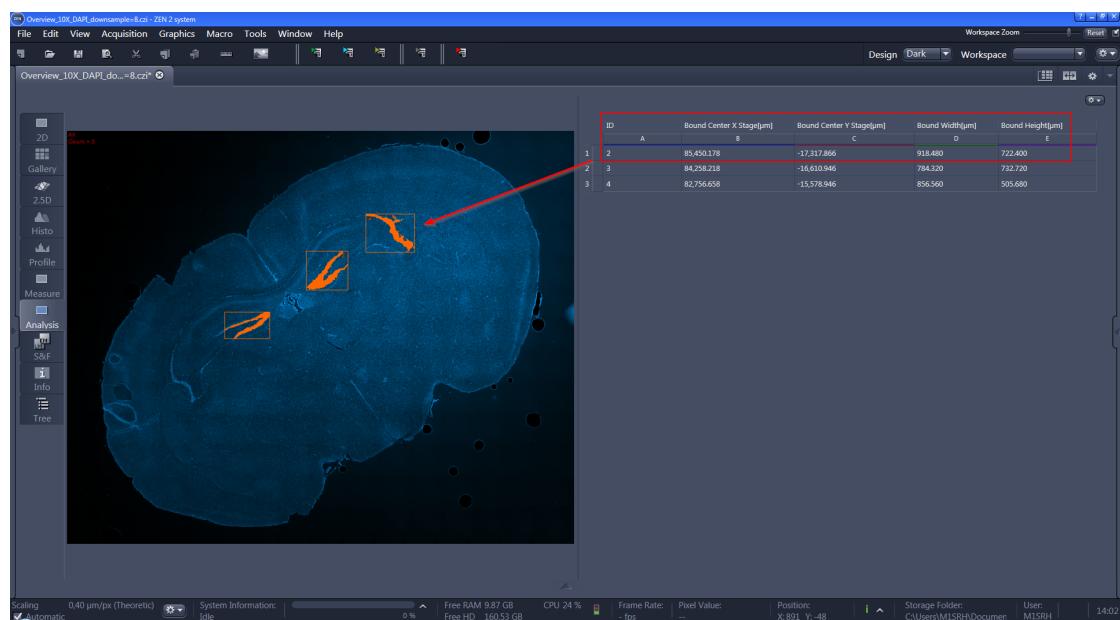


Figure 6: Result of the image analysis on the overview scan image.

2.4 Define Detailed Scan Experiment

First of all it is important to understand, that there is no such thing as a typical "Detailed Scan". What this experiment (or even a workflow) might be, depends on the application. In a general sense such a detailed scan is "some kind" of experiment carried at a specific position based upon the results of the image analysis. Examples could be:

- Simple Z-Stack with a high-NA objective lens.
- Multi-Channel Z-Stack using an optical sectioning method like SD, Apotome or AiryScan
- One of the above combined with a tile experiment.
- A series of subsequent experiments with different image modalities.

Since this concept is based on OAD, it is possible to automate almost any kind of workflow at a given position.

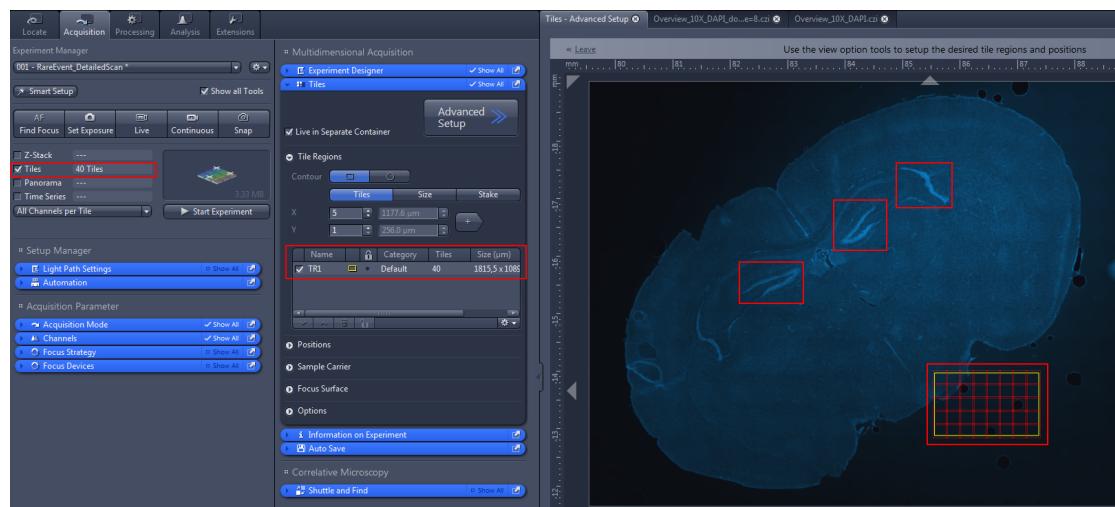


Figure 7: Setup of a simple detailed scan experiment.

Now the preparation is complete. So far we have available:

- The image analysis pipeline based on some sample data
- The Overview Scan Experiment = tile experiment created by the Tiles & Positions Module
- The Detailed Scan Experiment, which is also a tile experiment.

And now the real interesting part begins – how to setup a complete workflow out of those building blocks using an OAD Python script inside ZEN Blue.

3 Automation via OAD

3.1 Prerequisites

In order to be able to run the script one needs to copy it into the correct folder, which is located at:

"c:/Users/XXXX/Documents/Carl Zeiss/ZEN/Documents/Macros"

where XXXX stand for the actual user name.

3.2 Pure Scripting or User Dialog

First of all one has to decide if a pure script is sufficient or if there should be some kind of simple user interface (very basic wizard).

- A pure script is of course the faster solution, but is may be difficult to use for less advanced users.
- A basic wizard offers less flexibility, but may be easier to use for most users.

The choice clearly depends on the user, since there is no right or wrong approach here. For this tutorial we will focus on the 2nd approach. Once one has figured out how to realize this, the 1st approach will be straight forward since the basic workflow is identical.

3.3 User Dialog Requirements

The basic user interface of the dialog should have the following functionality:

- **Select the overview scan experiment.**
- **Select the image analysis pipeline** (to analyze the overview image)
- **Select the detailed scan experiment** (to acquire at "found" positions)
- **Define options for focussing** (depends on available hardware)
- **Select a folder to store the image data and analysis results.**

3.4 Prerequisites

Now we are ready to create the OAD macro required to control the complete workflow. The first step is (as always) to import the required modules and define the folder locations, etc. Additionally there are several python functions defined here for later usage.

```
1 ######
2 # File      : Guided_Acquisition_shortUI.py
3 # Version   : 7.0
4 # Author    : czsrh, czmla
5 # Date      : 12.04.2019
6 # Institution : Carl Zeiss Microscopy GmbH
7 #
8 # !!! Requires with ZEN >=2.6 HF3 - Use at your own Risk !!!
9 #
10 # Optimized for the use with Celldiscoverer 7 and DF2, but
11 # applicable for all motorized stands running in ZEN Blue.
12 # Please adapt focussing commands, especially FindSurface
13 # when using with other stands.
14 #
15 # 1) - Select Overview Scan Experiment
16 # 2) - Select appropriate Image Analysis Pipeline
17 # 3) - Select Detailed Scan Experiment
18 # 4) - Specify the output folder for the image and data tables
19 #
20 #
21 # Copyright(c) 2019 Carl Zeiss AG, Germany. All Rights Reserved.
22 #
23 # Permission is granted to use, modify and distribute this code,
24 # as long as this copyright notice remains part of the code.
25 #####
26
27 import time
28 from datetime import datetime
29 import errno
30 from System import Array
31 from System import ApplicationException
32 from System import TimeoutException
33 from System.IO import File, Directory, Path
34 import sys
35
36
37 # version number for dialog window
38 version = 7.0
39 # file name for overview scan
40 ovscan_name = 'OverviewScan.czi'
41
42 """
43 Additional XY offset for possible 2nd port relative to the 1st port
44 Example: 1st port Camera and 2nd port LSM
45 Can be set to Zero, when system is correctly calibrated
46 !!! Only use when OverView and DetailedScan are using different detector !!!
47 """
48 dx_detector = 0.0
49 dy_detector = 0.0
50
51 # experiment blockindex
52 blockindex = 0
53 # delay for specific hardware movements in [seconds]
54 hwdelay = 1
55 # postprocessing switch
56 do_postprocess = False
57
58
59 def dircheck(basefolder):
60
61     # check if the destination basefolder exists
62     base_exists = Directory.Exists(basefolder)
63
64     if base_exists:
65         print('Selected Directory Exists: ', base_exists)
66         # specify the desired output format for the folder, e.g. 2017-08-08_17-47-41
67         format = '%Y-%m-%d_%H-%M-%S'
68         # create the new directory
69         newdir = createfolder(basefolder, formatstring=format)
70         print('Created new directory: ', newdir)
71     if not base_exists:
72         Directory.CreateDirectory(basefolder)
73         newdir = basefolder
74
75 return newdir
76
```

```

77
78 def createfolder(basedir, formatstring='%-Y-%m-%d_%H-%M-%S'):
79     # construct new directory name based on date and time
80     newdir = Path.Combine(basedir, datetime.now().strftime(formatstring))
81     # check if the new directory (for whatever reasons) already exists
82     try:
83         newdir_exists = Directory.Exists(newdir)
84         if not newdir_exists:
85             # create new directory if it does not exist
86             Directory.CreateDirectory(newdir)
87         if newdir_exists:
88             # raise error if it really already exists
89             raise SystemExit
90     except OSError as e:
91         if e.errno != errno.EEXIST:
92             newdir = None
93             raise # This was not a "directory exist" error..
94
95     return newdir
96
97
98 def getshortfiles(filelist):
99     files_short = []
100    for short in filelist:
101        files_short.append(Path.GetFileName(short))
102
103    return files_short
104
105
106 def cloneexp(expname, prefix='GA_', save=True, reloadexp=True):
107
108     exp = Zen.Acquisition.Experiments.GetByName(expname)
109     exp_newname = prefix + expname
110
111     # save experiment
112     if save:
113         exp.SaveAs(exp_newname, False)
114         print('Saved Temporay Experiment as : ', exp_newname)
115         # close the original experiment object
116         exp.Close()
117         time.sleep(1)
118     # reload experiment
119     if reloadexp:
120         exp_reload = Zen.Acquisition.Experiments.GetByName(exp_newname)
121     elif not reloadexp:
122         exp_reload = None
123
124     return exp_reload
125
126
127 def runSWAF_special(SWAF_exp,
128                      delay=5,
129                      searchStrategy='Full',
130                      sampling=ZenSoftwareAutofocusSampling.Coarse,
131                      relativeRangeIsAutomatic=False,
132                      relativeRangeSize=500,
133                      timeout=0):
134
135     # get current z-Position
136     zSWAF = Zen.Devices.Focus.ActualPosition
137     print('Z-Position before special SWAF :', zSWAF)
138
139     # set DetailScan active and wait for moving hardware due to settings
140     SWAF_exp.SetActive()
141     time.sleep(delay)
142
143     # set SWAF parameters
144     SWAF_exp.SetAutofocusParameters(searchStrategy=searchStrategy,
145                                     sampling=sampling,
146                                     relativeRangeIsAutomatic=relativeRangeIsAutomatic,
147                                     relativeRangeSize=relativeRangeSize)
148
149     try:
150         print('Running special SWAF ...')
151         zSWAF = Zen.Acquisition.FindAutofocus(SWAF_exp, timeoutSeconds=timeout)
152     except ApplicationException as e:
153         print('Application Exception : ', e.Message)
154     except TimeoutException as e:
155         print(e.Message)
156
157     print('Z-Position after initial SWAF : ', zSWAF)
158
159     return zSWAF
160
161 def checktableentry(datatable, entry2check='ImageSceneContainerName'):
162
163     num_col = datatable.ColumnCount
164     entry_exits = False

```

```
166     for c in range(0, num_col):
167         # get the current column name
168         colname = datatable.Columns[c].ColumnName
169         if colname == entry2check:
170             column = c
171             entry_exists = True
172             break
173
```

3.5 Create the User Dialog

The next task is to create the user dialog using the requirements referred to earlier on. Once the dialog is closed, the results have to be collected.

Additionally one must check if the Detailed Scan experiment is a tile experiment, where the tile size has to be adapted accordingly. This is done using a little tool function.

```
216 # Initialize Dialog
217 GuidedAcqDialog = ZenWindow()
218 GuidedAcqDialog.Initialize('Guided Acquisition - Version : ' + str(version))
219 # add components to dialog
220 GuidedAcqDialog.AddLabel('1) Select Overview Experiment -----')
221 GuidedAcqDialog.AddDropDown('overview_exp', 'Overview Scan Experiment', expfiles_short, 0)
222 GuidedAcqDialog.AddCheckbox('fs_before_overview', 'OPTION - FindSurface (DF only) before Overview', False)
223 GuidedAcqDialog.AddCheckbox('SWAF_before_overview', 'OPTION - SWAF before Overview', False)
224 GuidedAcqDialog.AddIntegerRange('SWAF_ov_initial_range', 'Initial SWAF Range before Overview [micron]', 200, 50, 3000)
225 GuidedAcqDialog.AddLabel('2) Select Image Analysis to detect objects -----')
226 GuidedAcqDialog.AddDropDown('ip_pipe', 'Image Analysis Pipeline', ipfiles_short, 0)
227 GuidedAcqDialog.AddLabel('3) Select DetailScan Experiment -----')
228 GuidedAcqDialog.AddDropDown('detailed_exp', 'Detailed Scan Experiment', expfiles_short, 1)
229 GuidedAcqDialog.AddCheckbox('fs_before_detail', 'OPTION - FindSurface (DF only) before Detail', False)
230 GuidedAcqDialog.AddCheckbox('SWAF_before_detail', 'OPTION - SWAF before Detail', False)
231 GuidedAcqDialog.AddIntegerRange('SWAF_detail_initial_range', 'Initial SWAF Range before Detail [micron]', 100, 10, 1000)
232 GuidedAcqDialog.AddCheckbox('recallfocus_beforeDT', 'OPTION - Use RecallFocus (DF only) before Detail', False)
233 GuidedAcqDialog.AddLabel('4) Specify basefolder to save the images -----')
234 GuidedAcqDialog.AddFolderBrowser('outfolder', 'Basefolder for Images and Data Tables', imgfolder)
235
236 # show the window
237 result = GuidedAcqDialog.Show()
238 if result.HasCanceled:
239     message = 'Macro was canceled by user.'
240     print(message)
241     raise SystemExit
242
```

The resulting dialog window is shown below.



Figure 8: Simple User Dialog to run the Guided Acquisition WorkFlow

The dialog has four main steps where the user must define the desired options:

1. Select **overview experiment** and additional focus options.
2. Select **image analysis** to detect the objects.
3. Select **detailed experiment** and additional focus options.
4. Choose an **output folder** to save the data.

3.6 Overview Scan Experiment

At this point we have all the information we need to actually start the workflow. The first step is to execute the actual overview scan experiment, which is most likely a tile experiment. The resulting image will be saved to the specified folder.

```
276 ##### START OVERVIEW SCAN EXPERIMENT #####
277
278 if fs_beforeOV:
279     # initial focussing via FindSurface to assure a good starting position
280     Zen.Acquisition.FindSurface()
281     print('Z-Position after FindSurface: ', Zen.Devices.Focus.ActualPosition)
282
283 if SWAF_beforeOV:
284     zSWAF = runSWAF_special(OVScan_reloaded,
285                             delay=hwdelay,
286                             searchStrategy='Full',
287                             sampling=ZenSoftwareAutofocusSampling.Coarse,
288                             relativeRangeIsAutomatic=False,
289                             relativeRangeSize=SWAF_beforeOV_range,
290                             timeout=1)
291
292 # get the resulting z-position
293 znew = Zen.Devices.Focus.ActualPosition
294
295 # adapt the Overview Scan Tile Experiment with new Z-Position
296 if OVScanIsTileExp:
297     OVScan_reloaded.ModifyTileRegionsZ(blockindex, znew)
298     print('Adapted Z-Position of Tile OverView. New Z = ', '%.2f' % znew)
299
300 # execute the experiment
301 print('\nRunning OverviewScan Experiment.\n')
302 output_OVScan = Zen.Acquisition.Execute(OVScan_reloaded)
303 # For testing purposes - Load overview scan image automatically instead of executing the "real" experiment
304 # output_OVScan = Zen.Application.LoadImage(r'c:\Temp\input\OverViewScan_8Brains.czi', False)
305
306 # show the overview scan inside the document area
307 Zen.Application.Documents.Add(output_OVScan)
308 ovdoc = Zen.Application.Documents.GetByName(output_OVScan.Name)
309
310 # save the overview scan image inside the select folder
311 output_OVScan.Save(Path.Combine(OutputFolder, ovscan_name))
312
313 ##### END OVERVIEW SCAN EXPERIMENT #####
```

A very important part of such an automated workflow is to find the focus. To ensure this, the desired objective and optovar are changed before the overview starts, followed by a series of **optional** focus actions:

1. Find the surface of the sample carrier.
2. Focus onto the specimen sample via SWAF.
3. Store the resulting new Z-position.

When testing out applications like this it is very useful to use test data sets to save time instead of re-running an acquisition many times. Therefore it can be helpful to **comment** the lines where the real experiment is executed and **uncomment** the line inside the script, where one can load an already acquired overview image.

3.7 Find the interesting Objects

Now it is time to run the image analysis on the overview image. As a result, two tables will be created inside ZEN. The first contains information about all objects and the second contains information about the single objects (for instance their stage positions). This second table is what will be used as an input for the detailed scan later on.

```
315 # Load analysis setting created by the wizard or an separate macro
316 ias = ZenImageAnalysisSetting()
317
318 # for simulation use: 000 - RareEventExample.czias
319 ias.Load(ImageAS)
320
321 # Analyse the image
322 Zen.Analyzing.Analyze(output_OVScan, ias)
323
324 # Create Zen table with results for all detected objects (parent class)
325 AllObj = Zen.Analyzing.CreateRegionsTable(output_OVScan)
326
327 # Create Zen table with results for each single object
328 SingleObj = Zen.Analyzing.CreateRegionTable(output_OVScan)
329
330 # check for existence of required column names inside table
331 soi = SingleObj.GetBoundsColumnInfoFromImageAnalysis(True)
332
333 # 1st item is a bool indicating if all required columns could be found
334 columnsOK = soi.AreRequiredColumnsAvailable
335
336 if not columnsOK:
337     print('Execution stopped. Required Columns are missing.')
338     raise Exception('Execution stopped. Required Columns are missing.')
339
340 # show and save data tables to the specified folder
341 Zen.Application.Documents.Add(AllObj)
342 Zen.Application.Documents.Add(SingleObj)
343 AllObj.Save(Path.Combine(OutputFolder, 'OverviewTable.csv'))
344 SingleObj.Save(Path.Combine(OutputFolder, 'SingleObjectsTable.csv'))
345
346 # check the number of detected objects = rows inside image analysis table
347 num_POI = SingleObj.RowCount
```

As shown in figure 5, it is required to have the correct image analysis features selected. Only the will the respective columns inside the table exist. This has to be checked because otherwise the workflow will be impossible.

The "checking" itself is again done by a tool function. The output will be a boolean and a dictionary in order to look up the column index for the respective image analysis parameters. This has the advantage of being independent from a specific column order. One just has to make sure that all required columns are there.

If the option inside the initial dialog window was checked, the offset between the sample carrier surface and the actual specimen will be measured and stored inside the Definite Focus via **Store Focus**, so that **Recall Focus** can be used later on when needed. The dialog also offers the possibility to enter an arbitrary offset manually, which will only be applied if the offset was not determined automatically.

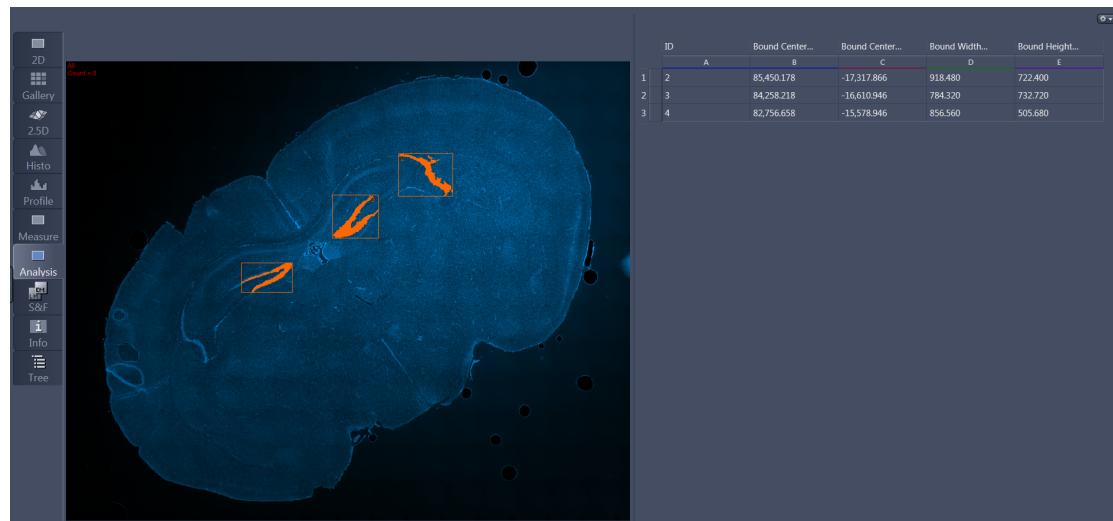


Figure 9: The results of the overview scan and the image analysis.

Finally the number of detected objects is checked by looking up the number of rows inside the table containing the information about the single objects.

3.8 Modify the Tile Dimensions

Once one has the list of objects with the stage coordinates, the next task is to create a loop to cycle through all detected positions.

- Move to the correct XYZ stage positions.
- Run Focus Actions when required.
- Initialize the DetailScan experiment at the new position.
- Adapt TilePosition using Tiles.TileTools.ModifyTileRegionsXYZ.
- Modify TileSize using Tiles.TileTools.ModifyTileRegionsSize.

```

405 ##### START DETAILED SCAN EXPERIMENT #####
406
407 # get the actual Focus position
408 zpos = Zen.Devices.Focus.ActualPosition
409
410 # check for the column 'ID' which is required
411 ID_exits, column_ID = checktableentry(SingleObj,
412                                         entry2check='ID')
413
414
415 # execute detailed experiment at the position of every detected object
416 for i in range(0, num_POI, 1):
417
418     # get the object information from the position table
419     # get the ID of the object - IDs start with 2 !!!
420     #POI_ID = SingleObj.GetValue(i, 0)
421     POI_ID = SingleObj.GetValue(i, column_ID)
422
423     # get XY-stage position from table
424     xpos = SingleObj.GetValue(i, soi.CenterXColumnIndex)
425     ypos = SingleObj.GetValue(i, soi.CenterYColumnIndex)
426
427     # move to the current position
428     Zen.Devices.Stage.MoveTo(xpos + dx_detector, ypos + dy_detector)
429     print('Moving Stage to Object ID:', POI_ID, ' at :', '%.2f' % xpos, '%.2f' % ypos)
430
431     # try to apply RecallFocus (DF only) when this option is used
432     if userrecallfocus:
433         try:
434             # apply RecallFocus for the current position
435             Zen.Acquisition.RecallFocus()
436             zpos = Zen.Devices.Focus.ActualPosition
437             print('Recall Focus (Definite Focus) applied.')
438             print('Updated Z-Position: ', zpos)
439         except ApplicationException as e:
440             print('Application Exception : ', e.Message)
441             print('RecallFocus (Definite Focus) failed.')
442
443         print('New Z-Position before Detail Experiment will start:', zpos)
444
445     # if DetailScan is a Tile Experiment
446     if DetailIsTileExp:
447
448         print('Detailed Experiment contains TileRegions.')
449         # Modify tile center position - get bounding rectangle width and height in microns
450         bcwidth = SingleObj.GetValue(i, soi.WidthColumnIndex)
451         bcheight = SingleObj.GetValue(i, soi.HeightColumnIndex)
452         print('Width and Height : ', '%.2f' % bcwidth, '%.2f' % bcheight)
453         print('Modifying Tile Properties XYZ Position and width and height.')
454
455         # Modify the XYZ position and size of the TileRegion on-the-fly
456         print('Starting Z-Position for current Object: ', '%.2f' % zpos)
457         print('New Tile Properties: ', '%.2f' % xpos, '%.2f' % ypos, '%.2f' % zpos, '%.2f' % bcwidth, '%.2f' % bcheight)
458         DetailScan_reloaded.ClearTileRegionsAndPositions(blockindex)
459         try:
460             DetailScan_reloaded.AddRectangleTileRegion(blockindex, xpos, ypos, bcwidth, bcheight, zpos)
461         except ApplicationException as e:
462             print('Application Exception : ', e.Message)
463
464     if not DetailIsTileExp:
465         print('Detailed Experiment does not contain TileRegions. Nothing to modify.')
466

```

3.9 Run the Detailed Scan

The last steps, which need to be executed now, are:

- Run the (modified) detailed scan experiment here.
- Save the data to the specified folder.
- Close the image document in ZEN.
- Rename the file using the current object ID.

```
467 # execute the DetailScan experiment
468 print('Running Detail Scan Experiment at new XYZ position.')
469 try:
470     output_detailscan = Zen.Acquisition.Execute(DetailScan_reloaded)
471 except ApplicationException as e:
472     print('Application Exception : ', e.Message)
473
474 # get the image data name
475 dtscan_name = output_detailscan.Name
476
477 """
478 Modification for multiscene images: container name needs to be part
479 of the filename of the detailed Scan.
480 First check if Column "Image Scene Container Name" is available
481 and then add Container Name to filename.
482 """
483
484 wellid_exist, column_wellid = checktableentry(SingleObj,
485                                             entry2check='ImageSceneContainerName')
486
487 # in case Image Scene Container Name is not defined
488 if not wellid_exist:
489     message = 'Missing column ImageSceneContainerName in Image Analysis Results.\nPlease Select Features inside the Image
490     ↪ Analysis when needed.'
491     print(message)
492     container_name = 'empty'
493
494 # save the image data to the selected folder and close the image
495 output_detailscan.Save(Path.Combine(OutputFolder, output_detailscan.Name))
496 output_detailscan.Close()
497
498 # rename the CZI regarding to the object ID - Attention - IDs start with 2 !!!
499 if not wellid_exist:
500     # no wellID found inside table
501     newname_dtscan = 'DTScan_ID' + str(POI_ID) + '.czi'
502 if wellid_exist:
503     # wellID was found inside table
504     well_id = SingleObj.GetValue(i, column_wellid)
505     newname_dtscan = 'DTScan_Well_' + str(well_id) + '_ID_' + str(POI_ID) + '.czi'
506
507 print('Renaming File: ' + dtscan_name + ' to: ' + newname_dtscan + '\n')
File.Move(Path.Combine(OutputFolder, dtscan_name), Path.Combine(OutputFolder, newname_dtscan))
```

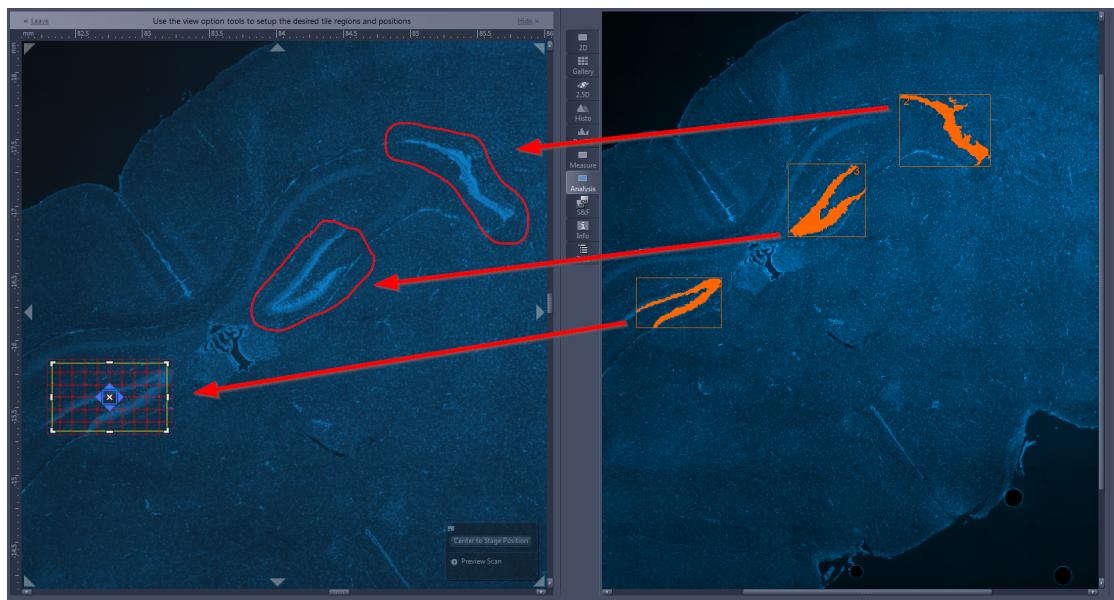


Figure 10: The results of the detailed scan - tile regions adapts to objects.

That is it. All image data sets acquired at the detected positions are stored inside the correct folder using the object IDs as names. The tables are saved in the same location.

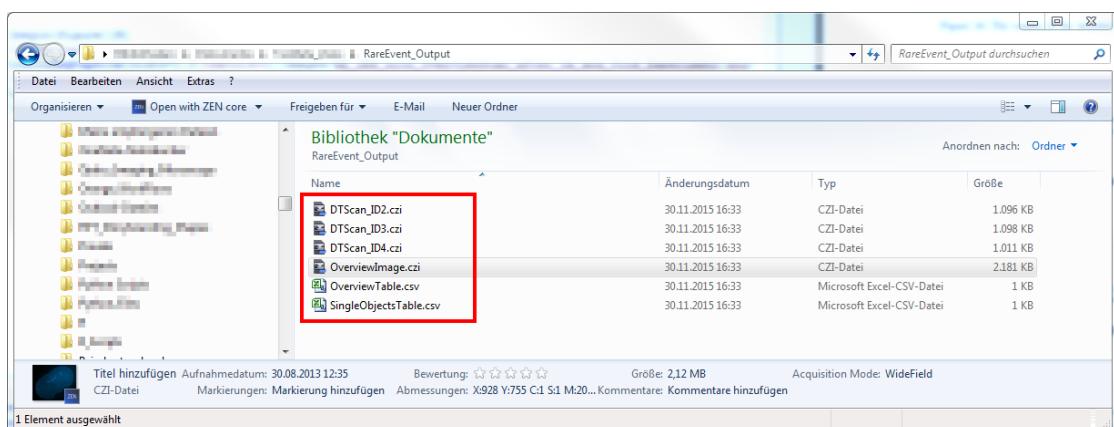


Figure 11: The results of the detailed scan stored inside a folder.

4 Testrun with FluoCells Slide

To test the workflow on a real sample one can use the FluoCells slide. The general idea is to setup an image analysis that detects some "interesting" objects. In order to create a somewhat short test run it is recommended to use an image analysis pipeline that only yields in a few positive detection events.

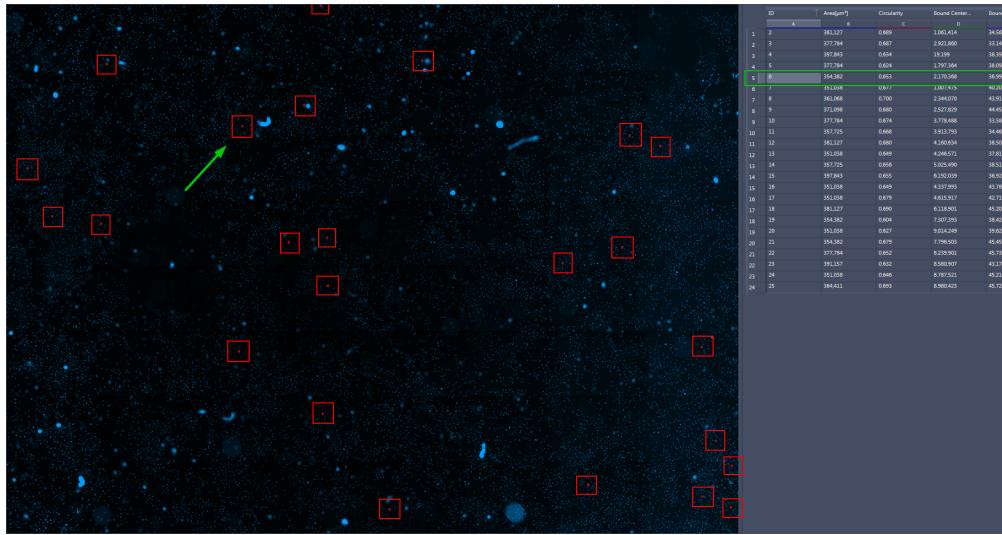


Figure 12: The results of the overview scan and image analysis.

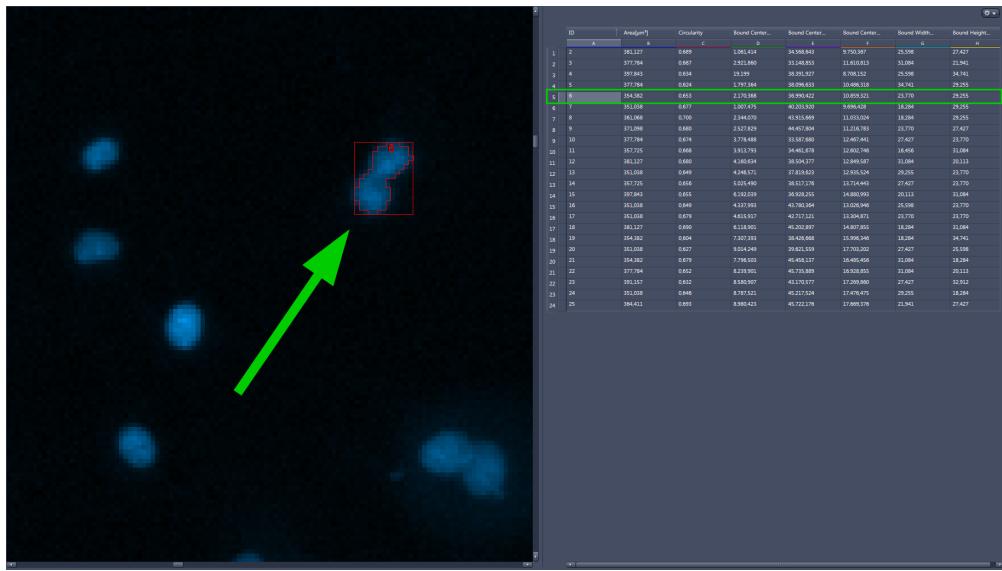


Figure 13: Detailed view on a detected object.

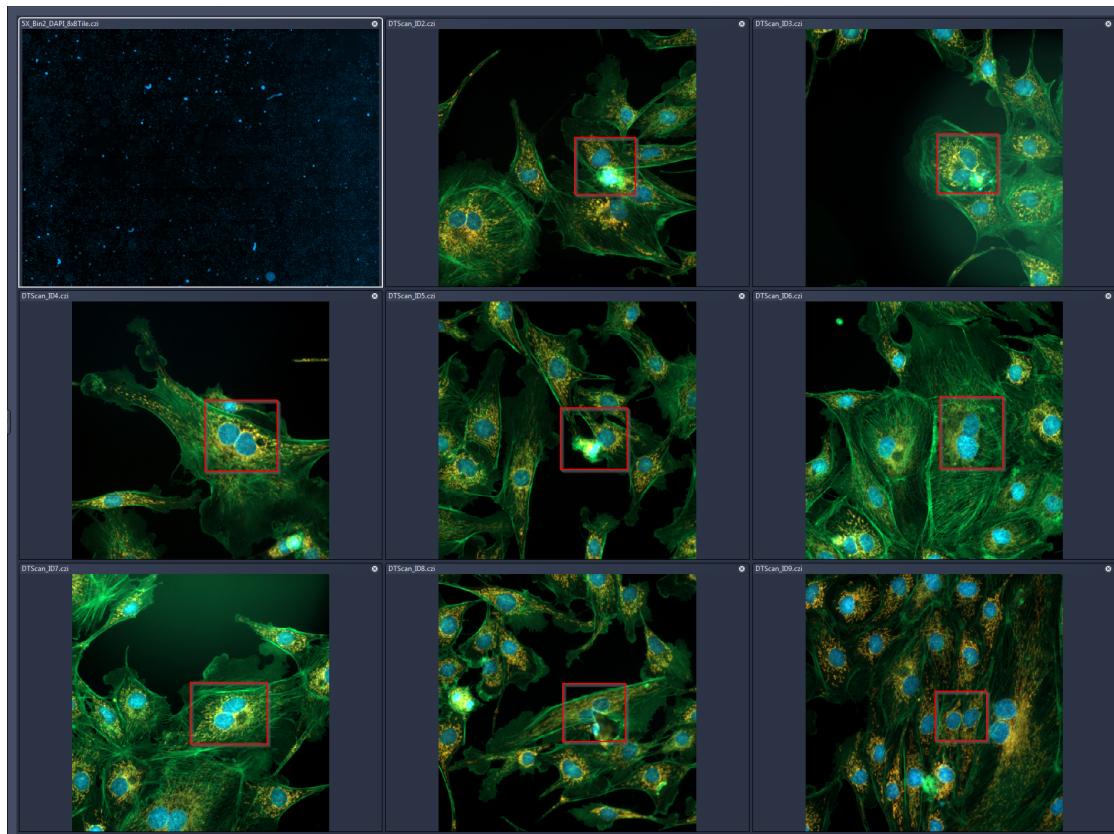


Figure 14: Detailed images from some of the detected objects.

The image analysis yielded **24** positive detections of cells with a large nucleus, preferably cells which were "caught" during cell division. Keep in mind, that the image analysis was not "fine-tuned" to catch all cell divisions. It is all about testing out the general workflow. For this reason it is also good practice to acquire a small overview scan at the beginning, e.g. a 2x2 tile image.

5 Test Run with Brain Slide

The same basic workflow can be also used to detect brain slices on a slide. All one needs to change is the image analysis pipeline in order to detect the slices.

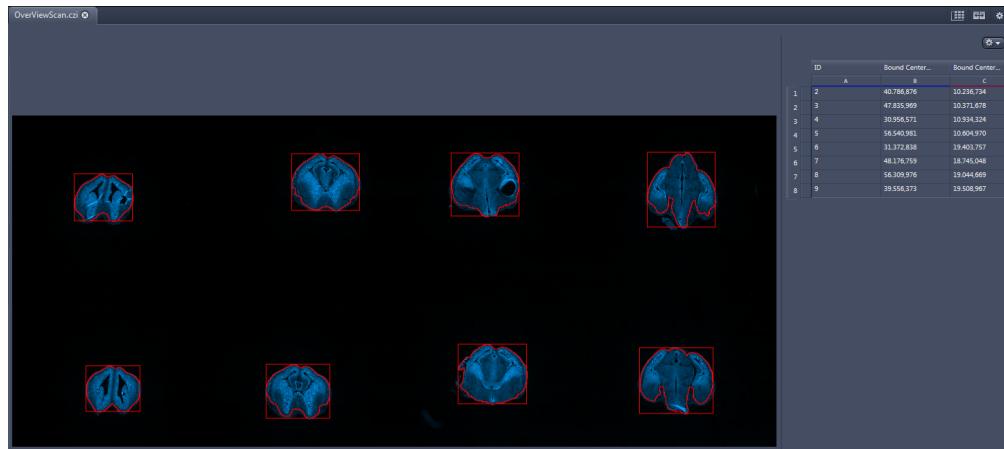


Figure 15: The results of the overview scan and image analysis.

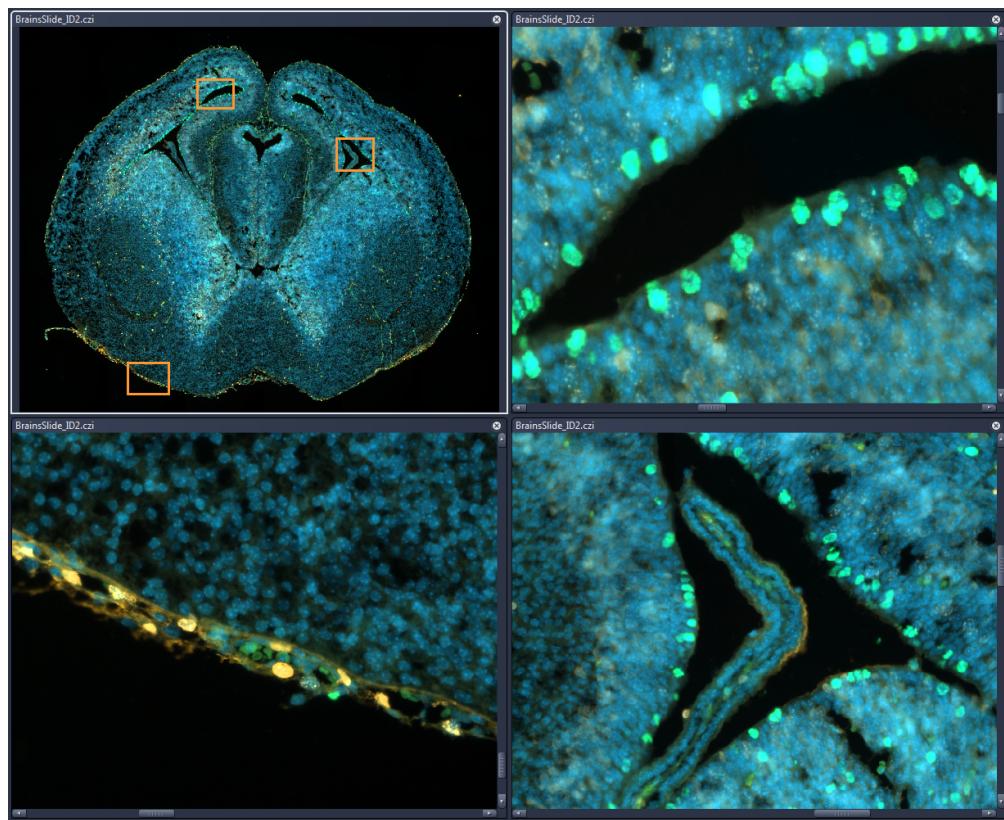


Figure 16: A more detailed image of the first specimen with three zoomed-in regions.

The image analysis yielded **eight** positive regions for brains slices on this slide. The properties of the bounding rectangle were used to modify the required tile experiment. The final detailed scan contained 36 tiles and it was imaged using only a hardware-based focus system.

Keep in mind that the actual focusing during such a experiment must be still part of the focus strategies with the Detail Scan experiment. Which one must be used here greatly depends on the nature of the sample and the actual application. The Guided Acquisition tool only supports finding the initial positions based on the overview scan image.

6 Mitosis Detection with Camera and LSM

An especially interesting option is to combine the power of a camera-based overview scan with the optional sectioning capabilities of an LSM. Such a workflow can be easily configured in ZEN Blue by setting up the respective experiment for the overview scan with camera detection using a low magnification and the LSM-based detail scan, e.g. Z-Stack, using a high NA objective.

Since ZEN Blue 2.5 this software has a module called **ZEN Connect**, which allows combining and correlating images inside one sample-centric workspace. Every acquired image by either the camera or the LSM will be placed here based on the XY stage coordinates. Therefore the **Correlative Workspace (CWS)** is ideally suited to display the results of a Guided Acquisition workflow.

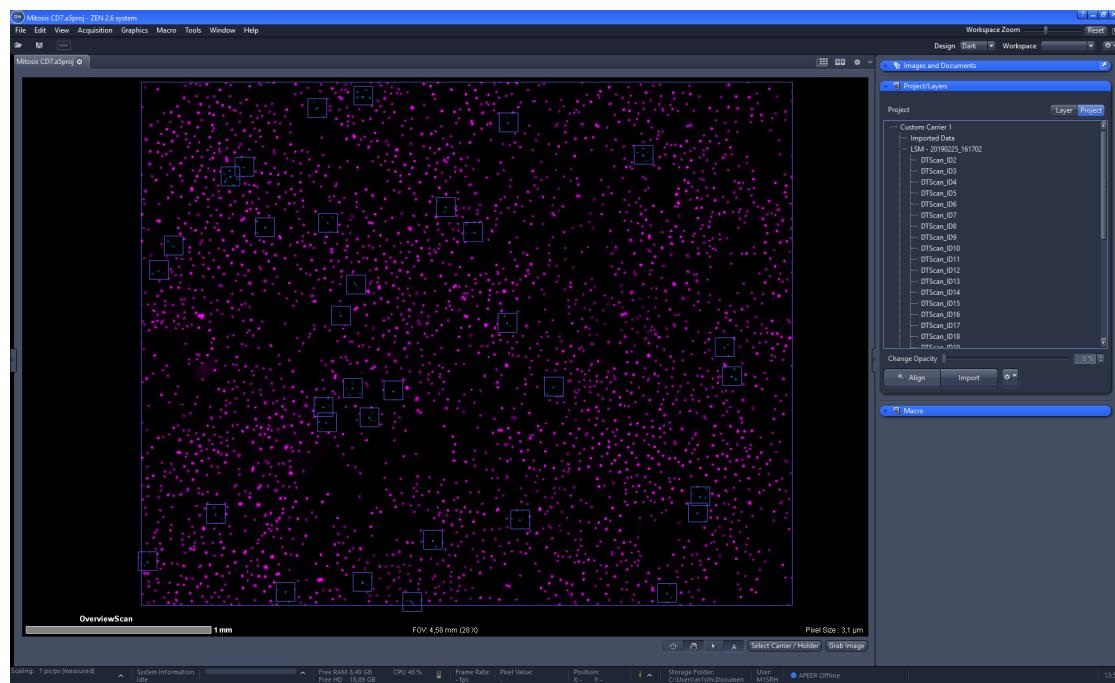


Figure 17: The results of the overview scan and image analysis inside the CWS.

Advanced Automated Microscopy: Guided Acquisition

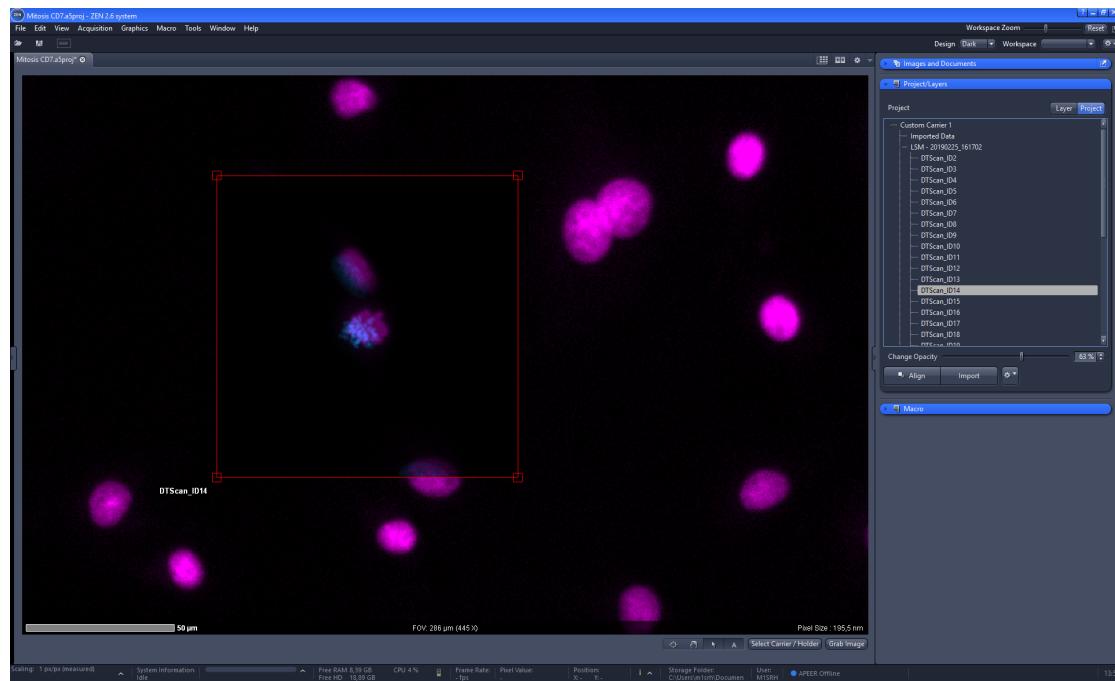


Figure 18: Overlay between positive detection inside overview scan and detailed scan using the Correlative Workspace (CWS).

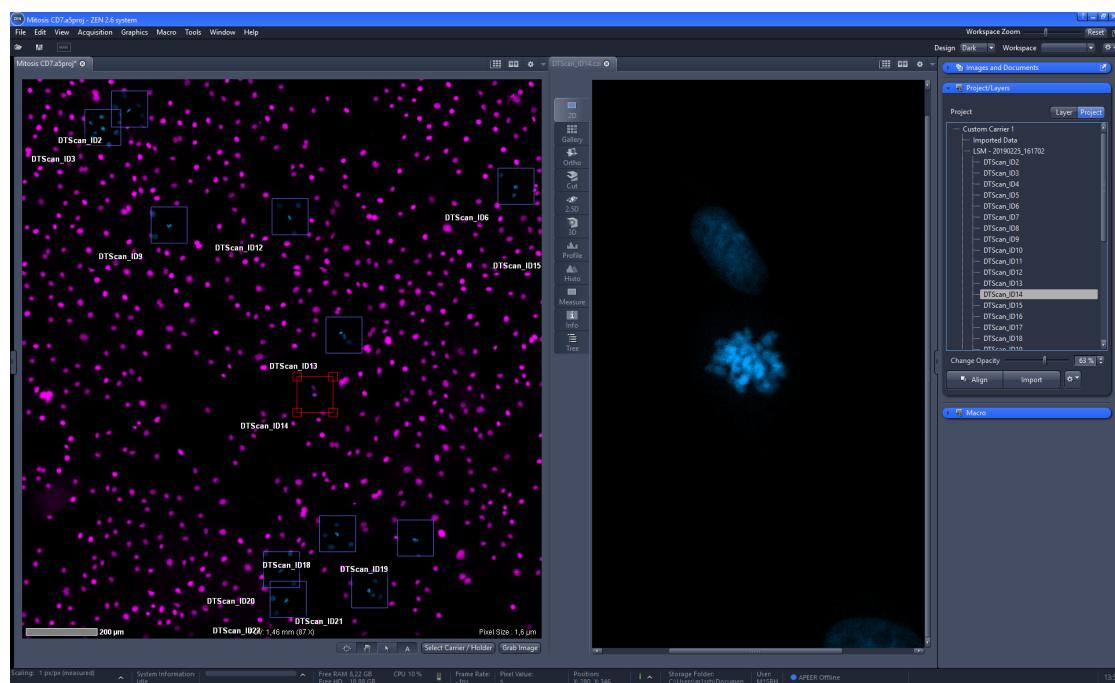


Figure 19: Side-by-Side view of the CWS and the normal view inside ZEN Blue.

7 Appendix

7.1 Complete Workflow Diagram

This is the complete workflow diagram for **Guided Acquisition**.

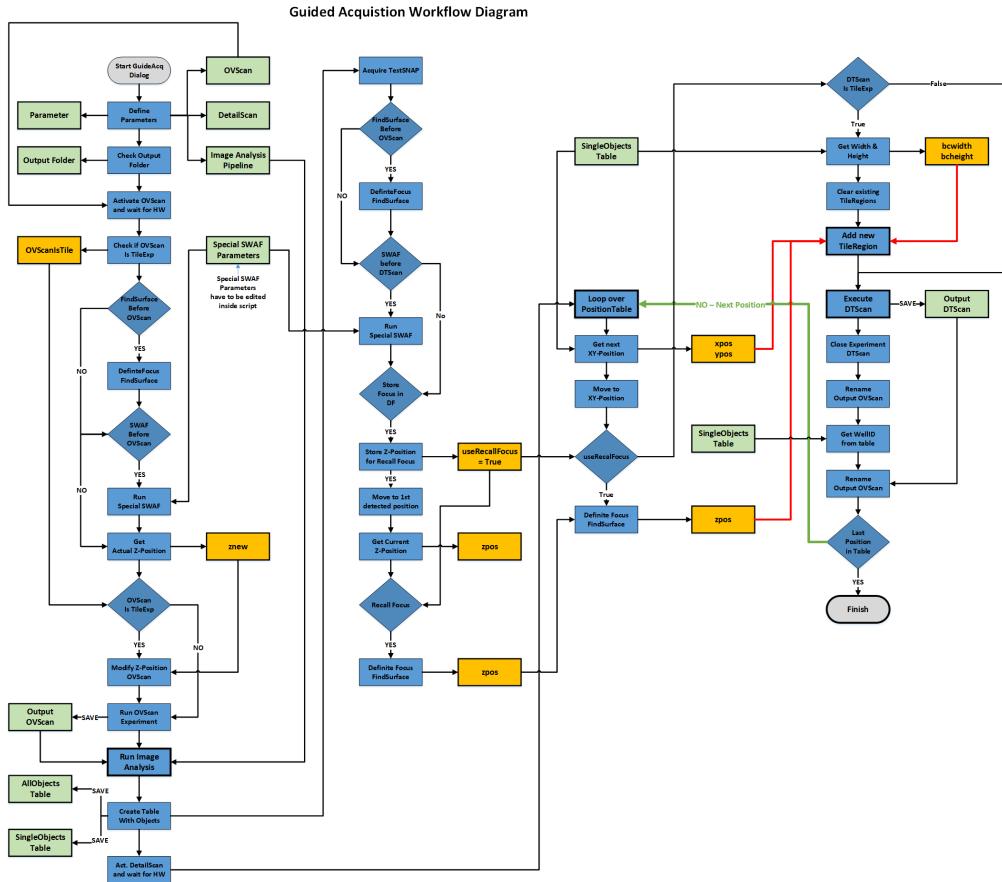


Figure 20: Complete workflow diagram for Guided Acquisition

7.2 Python Script: Guided_Acquisition_shortUI.py

This is the complete ZEN Blue python script used to realize the workflow described above. In principle it can work on any motorized microscope stand running under ZEN blue (with some modifications), but it was tested and optimized mainly for the Celldiscoverer 7 and for the AxioObserver 7. Both of those stands have the Definite Focus 2 as an additional hardware option, which is strongly advised for such highly automated workflows, where fast and efficient focus options, such as Find Surface are really a big plus.

```
1 #####  
2 # File      : Guided_Acquisition_shortUI.py  
3 # Version   : 7.0  
4 # Author    : czsrb, czmla  
5 # Date      : 12.04.2019  
6 # Institution : Carl Zeiss Microscopy GmbH  
7 #  
8 # !!! Requires with ZEN >=2.6 HF3 - Use at your own Risk !!!  
9 #  
10 # Optimized for the use with Celldiscoverer 7 and DF2, but  
11 # applicable for all motorized stands running in ZEN Blue.  
12 # Please adapt focussing commands, especially FindSurface  
13 # when using with other stands.  
14 #  
15 # 1) - Select Overview Scan Experiment  
16 # 2) - Select appropriate Image Analysis Pipeline  
17 # 3) - Select Detailed Scan Experiment  
18 # 4) - Specify the output folder for the image and data tables  
19 #  
20 #  
21 # Copyright(c) 2019 Carl Zeiss AG, Germany. All Rights Reserved.  
22 #  
23 # Permission is granted to use, modify and distribute this code,  
24 # as long as this copyright notice remains part of the code.  
25 #####  
26  
27 import time  
28 from datetime import datetime  
29 import errno  
30 from System import Array  
31 from System import ApplicationException  
32 from System import TimeoutException  
33 from System.IO import File, Directory, Path  
34 import sys  
35  
36  
37 # version number for dialog window  
38 version = 7.0  
39 # file name for overview scan  
ovscan_name = 'OverviewScan.czi'  
40  
41 #####  
42 # Additional XY offset for possible 2nd port relative to the 1st port  
43 # Example: 1st port Camera and 2nd port LSM  
44 # Can be set to Zero, when system is correctly calibrated  
45 #!!! Only use when OverView and DetailedScan are using different detector !!!  
46 #####  
47 dx_detector = 0.0  
dy_detector = 0.0  
50  
51 # experiment blockindex  
blockindex = 0  
53 # delay for specific hardware movements in [seconds]  
54 hwdelay = 1  
55 # postprocessing switch  
do_postprocess = False  
57  
58  
59 def dircheck(basefolder):  
60  
61     # check if the destination basefolder exists  
62     base_exists = Directory.Exists(basefolder)  
63  
64     if base_exists:  
65         print('Selected Directory Exists: ', base_exists)  
66         # specify the desired output format for the folder, e.g. 2017-08-08_17-47-41  
67         format = '%Y-%m-%d_%H-%M-%S'  
68         # create the new directory  
69         newdir = createfolder(basefolder, formatstring=format)  
70         print('Created new directory: ', newdir)  
71     if not base_exists:
```

```

72     Directory.CreateDirectory(basefolder)
73     newdir = basefolder
74
75     return newdir
76
77
78 def createfolder(basedir, formatstring='%(Y-%m-%d_%H-%M-%S)':
79     # construct new directory name based on date and time
80     newdir = Path.Combine(basedir, datetime.now().strftime(formatstring))
81     # check if the new directory (for whatever reasons) already exists
82     try:
83         newdir_exists = Directory.Exists(newdir)
84         if not newdir_exists:
85             # create new directory if it does not exist
86             Directory.CreateDirectory(newdir)
87         if newdir_exists:
88             # raise error if it really already exists
89             raise SystemExit
90     except OSError as e:
91         if e.errno != errno.EEXIST:
92             newdir = None
93             raise # This was not a "directory exist" error..
94
95     return newdir
96
97
98 def getshortfiles(filelist):
99     files_short = []
100    for short in filelist:
101        files_short.append(Path.GetFileName(short))
102
103    return files_short
104
105
106 def cloneexp(expname, prefix='GA_', save=True, reloadexp=True):
107
108     exp = Zen.Acquisition.Experiments.GetByName(expname)
109     exp_newname = prefix + expname
110
111     # save experiment
112     if save:
113         exp.SaveAs(exp_newname, False)
114         print('Saved Temporary Experiment as : ', exp_newname)
115         # close the original experiment object
116         exp.Close()
117         time.sleep(1)
118
119     # reload experiment
120     if reloadexp:
121         exp_reload = Zen.Acquisition.Experiments.GetByName(exp_newname)
122     elif not reloadexp:
123         exp_reload = None
124
125     return exp_reload
126
127
128 def runSWAF_special(SWAF_exp,
129                     delay=5,
130                     searchStrategy='Full',
131                     sampling=ZenSoftwareAutofocusSampling.Coarse,
132                     relativeRangeIsAutomatic=False,
133                     relativeRangeSize=500,
134                     timeout=0):
135
136     # get current z-Position
137     zSWAF = Zen.Devices.Focus.ActualPosition
138     print('Z-Position before special SWAF : ', zSWAF)
139
140     # set DetailScan active and wait for moving hardware due to settings
141     SWAF_exp.SetActive()
142     time.sleep(delay)
143
144     # set SWAF parameters
145     SWAF_exp.SetAutofocusParameters(searchStrategy=searchStrategy,
146                                     sampling=sampling,
147                                     relativeRangeIsAutomatic=relativeRangeIsAutomatic,
148                                     relativeRangeSize=relativeRangeSize)
149     try:
150         print('Running special SWAF ...')
151         zSWAF = Zen.Acquisition.FindAutofocus(SWAF_exp, timeoutSeconds=timeout)
152     except ApplicationException as e:
153         print('Application Exception : ', e.Message)
154     except TimeoutException as e:
155         print(e.Message)
156
157     print('Z-Position after initial SWAF : ', zSWAF)
158
159     return zSWAF
160

```

```

161 def checktableentry(datatable, entry2check='ImageSceneContainerName'):
162
163     num_col = datatable.ColumnCount
164     entry_exists = False
165
166     for c in range(0, num_col):
167         # get the current column name
168         colname = datatable.Columns[c].ColumnName
169         if colname == entry2check:
170             column = c
171             entry_exists = True
172             break
173
174     return entry_exists, column
175
176
177 def run_postprocessing(image, parameters={}, func='topography'):
178
179     if func == 'topography':
180
181         noise_low = parameters['noise_low']
182         noise_high = parameters['noise_high']
183         outputfolder = parameters['outputfolder']
184         ext = parameters['extension']
185
186         # converting to topo, with defined FZ noisecut
187         # in filter settings: 0-255 means no filter, 1-254 means cut one gray scale from top and from bottom
188         imgtop = Zen.Processing.Transformation.Topography.CreateTopography(image, noise_low, noise_high)
189         # saving file to the directory
190         topo_filepath = Path.Combine(outputfolder, Path.GetFileNameWithoutExtension(image.FileName) + ext)
191         Zen.Processing.Utilities.ExportHeightmapFromTopography(imgtop, topo_filepath)
192         print('Exported to : ', topo_filepath)
193         imgtop.Close()
194
195         image.Close()
196
197 #####
198
199 # clear console output
200 Zen.Application.MacroEditor.ClearMessages()
201
202 # check the location of experiment setups and image analysis settings are stored
203 docfolder = Zen.Application.Environment.GetFolderPath(ZenSpecialFolder.UserDocuments)
204 imgfolder = Zen.Application.Environment.GetFolderPath(ZenSpecialFolder.ImageAutoSave)
205 imgfolder = Path.Combine(imgfolder, 'Guided_Acquisition')
206 # imgfolder = r'c:\Output\Guided_Acquisition'
207 format = '%Y-%m-%d_%H-%M-%S'
208
209 # get list with all existing experiments and image analysis setup and a short version of that list
210 expfiles = Directory.GetFiles(Path.Combine(docfolder, 'Experiment Setups'), '*.czexp')
211 ipfiles = Directory.GetFiles(Path.Combine(docfolder, 'Image Analysis Settings'), '*.czias')
212 expfiles_short = getshortfiles(expfiles)
213 ipfiles_short = getshortfiles(ipfiles)
214
215 # Initialize Dialog
216 GuidedAcqDialog = ZenWindow()
217 GuidedAcqDialog.Initialize('Guided Acquisition - Version : ' + str(version))
218 # add components to dialog
219 GuidedAcqDialog.AddLabel('1) Select Overview Experiment -----')
220 GuidedAcqDialog.AddDropDown('overview_exp', 'Overview Scan Experiment', expfiles_short, 0)
221 GuidedAcqDialog.AddCheckbox('fs_before_overview', 'OPTION - FindSurface (DF only) before Overview', False)
222 GuidedAcqDialog.AddCheckbox('SWAF_before_overview', 'OPTION - SWAF before Overview', False)
223 GuidedAcqDialog.AddIntegerRange('SWAF_ov_initial_range', 'Initial SWAF Range before Overview [micron]', 200, 50, 3000)
224 GuidedAcqDialog.AddLabel('2) Select Image Analysis to detect objects -----')
225 GuidedAcqDialog.AddDropDown('ip_pipe', 'Image Analysis Pipeline', ipfiles_short, 0)
226 GuidedAcqDialog.AddLabel('3) Select DetailScan Experiment -----')
227 GuidedAcqDialog.AddDropDown('detailed_exp', 'Detailed Scan Experiment', expfiles_short, 1)
228 GuidedAcqDialog.AddCheckbox('fs_before_detail', 'OPTION - FindSurface (DF only) before Detail', False)
229 GuidedAcqDialog.AddCheckbox('SWAF_before_detail', 'OPTION - SWAF before Detail', False)
230 GuidedAcqDialog.AddIntegerRange('SWAF_detail_initial_range', 'Initial SWAF Range before Detail [micron]', 100, 10, 1000)
231 GuidedAcqDialog.AddLabel('4) Specify basefolder to save the images -----')
232 GuidedAcqDialog.AddLabel('recallfocus_beforeDT', 'OPTION - Use RecallFocus (DF only) before Detail', False)
233 GuidedAcqDialog.AddLabel('outfolder', 'Basefolder for Images and Data Tables', imgfolder)
234
235 # show the window
236 result = GuidedAcqDialog.Show()
237 if result.HasCanceled:
238     message = 'Macro was canceled by user.'
239     print(message)
240     raise SystemExit
241
242 # get the values and store them
243 OverViewExpName = str(result.GetValue('overview_exp'))
244 ImageAS = str(result.GetValue('ip_pipe'))
245 DetailExpName = str(result.GetValue('detailed_exp'))
246 OutputFolder = str(result.GetValue('outfolder'))
247 fs_beforeOV = result.GetValue('fs_before_overview')
248 SWAF_beforeOV = result.GetValue('SWAF_before_overview')

```

```

250 SWAF_beforeOV_range = result.GetValue('SWAF_ov_initial_range')
251 fs_beforeDT = result.GetValue('fs_before_detail')
252 SWAF_beforeDT = result.GetValue('SWAF_before_detail')
253 SWAF_beforeDT_range = result.GetValue('SWAF_detail_initial_range')
254 RecallFocus = result.GetValue('recallfocus_beforeDT')
255
256
257 # print values
258 print('Overview Scan Experiment : ' + OverViewExpName)
259 print('Image Analysis Pipeline : ' + ImageAS)
260 print('Detailed Scan Experiment : ' + DetailExpName)
261 print('Output Folder for Data : ' + OutputFolder)
262 print('\n')
263
264 # check directory
265 OutputFolder = dircheck(OutputFolder)
266
267 # create a duplicate of the OVScan experiment to work with
268 OVScan_reloaded = cloneexp(OverViewExpName)
269
270 # active the temporary experiment to trigger its validation
271 OVScan_reloaded.SetActive()
272 time.sleep(hwdelay)
273 # check if the experiment contains tile regions
274 OVScanIsTileExp = OVScan_reloaded.IsTilesExperiment(blockindex)
275
276 ##### START OVERVIEW SCAN EXPERIMENT #####
277
278 if fs_beforeOV:
279     # initial focussing via FindSurface to assure a good starting position
280     Zen.Acquisition.FindSurface()
281     print('Z-Position after FindSurface: ', Zen.Devices.Focus.ActualPosition)
282
283 if SWAF_beforeOV:
284     zSWAF = runSWAF_special(OVScan_reloaded,
285                             delay=hwdelay,
286                             searchStrategy='Full',
287                             sampling=ZenSoftwareAutofocusSampling.Coarse,
288                             relativeRangeIsAutomatic=False,
289                             relativeRangeSize=SWAF_beforeOV_range,
290                             timeout=1)
291
292 # get the resulting z-position
293 znew = Zen.Devices.Focus.ActualPosition
294
295 # adapt the Overview Scan Tile Experiment with new Z-Position
296 if OVScanIsTileExp:
297     OVScan_reloaded.ModifyTileRegionsZ(blockindex, znew)
298     print('Adapted Z-Position of Tile OverView. New Z = ', '%.2f' % znew)
299
300 # execute the experiment
301 print('\nRunning OverviewScan Experiment.\n')
302 output_OVScan = Zen.Acquisition.Execute(OVScan_reloaded)
303 # For testing purposes - Load overview scan image automatically instead of executing the "real" experiment
304 # output_OVScan = Zen.Application.LoadImage(r'c:\Temp\input\OverViewScan_8Brains.czi', False)
305
306 # show the overview scan inside the document area
307 Zen.Application.Documents.Add(output_OVScan)
308 ovdoc = Zen.Application.Documents.GetByName(output_OVScan.Name)
309
310 # save the overview scan image inside the select folder
311 output_OVScan.Save(Path.Combine(OutputFolder, ovscan_name))
312
313 ##### END OVERVIEW SCAN EXPERIMENT #####
314
315 # Load analysis setting created by the wizard or an separate macro
316 ias = ZenImageAnalysisSetting()
317
318 # for simulation use: 000 - RareEventExample.czias
319 ias.Load(ImageAS)
320
321 # Analyse the image
322 Zen.Analyzing.Analyze(output_OVScan, ias)
323
324 # Create Zen table with results for all detected objects (parent class)
325 AllObj = Zen.Analyzing.CreateRegionsTable(output_OVScan)
326
327 # Create Zen table with results for each single object
328 SingleObj = Zen.Analyzing.CreateRegionTable(output_OVScan)
329
330 # check for existence of required column names inside table
331 soi = SingleObj.GetBoundsColumnInfoFromImageAnalysis(True)
332
333 # 1st item is a bool indicating if all required columns could be found
334 columnsOK = soi.AreRequiredColumnsAvailable
335
336 if not columnsOK:
337     print('Execution stopped. Required Columns are missing.')
338     raise Exception('Execution stopped. Required Columns are missing.')

```

```

339
340 # show and save data tables to the specified folder
341 Zen.Application.Documents.Add(AllObj)
342 Zen.Application.Documents.Add(SingleObj)
343 AllObj.Save(Path.Combine(OutputFolder, 'OverviewTable.csv'))
344 SingleObj.Save(Path.Combine(OutputFolder, 'SingleObjectsTable.csv'))
345
346 # check the number of detected objects = rows inside image analysis table
347 num_POI = SingleObj.RowCount
348
349 ##### Prepare DetailScan #####
350
351 print('Starting DetailScan ...')
352
353 # create an duplicate of the DetailScan experiment to work with
354 DetailScan_reloaded = cloneexp(DetailExpName)
355
356 # active the temporary experiment to trigger its validation
357 DetailScan_reloaded.SetActive()
358 time.sleep(hwdelay)
359 # check if the experiment contains tile regions
360 DetailIsTileExp = DetailScan_reloaded.IsTilesExperiment(blockindex)
361
362 # test snap to change to the valid settings, e.g. the objective from the DetailScan
363 testsnap = Zen.Acquisition.AcquireImage(DetailScan_reloaded)
364 print('Acquire Test Snap using setting from DetailScan')
365 testsnap.Close()
366 # wait for moving hardware due to settings
367 time.sleep(hwdelay)
368
369 # move to 1st detected object
370 xpos_1st = SingleObj.GetValue(0, soi.CenterXColumnIndex)
371 ypos_1st = SingleObj.GetValue(0, soi.CenterYColumnIndex)
372 Zen.Devices.Stage.MoveTo(xpos_1st + dx_detector, ypos_1st + dy_detector)
373
374 if fs_beforeDT:
375     try:
376         # initial focussing via FindSurface to assure a good starting position
377         Zen.Acquisition.FindSurface()
378         print('Z-Position after FindSurface: ', Zen.Devices.Focus.ActualPosition)
379     except ApplicationException as e:
380         print('Application Exception : ', e.Message)
381         print('FindSurface (Definite Focus) failed.')
382
383 if SWAF_beforeDT:
384     zSWAF = runSWAF_special(DetailScan_reloaded,
385                             delay=hwdelay,
386                             searchStrategy='Full',
387                             sampling=ZenSoftwareAutofocusSampling.Coarse,
388                             relativeRangeIsAutomatic=False,
389                             relativeRangeSize=SWAF_beforeDT_range,
390                             timeout=0)
391
392 userecallfocus = False
393
394 if RecallFocus:
395     try:
396         # store current focus position inside DF to use it with RecallFocus
397         Zen.Acquisition.StoreFocus()
398         userecallfocus = True
399     except ApplicationException as e:
400         print('Application Exception : ', e.Message)
401         print('StoreFocus (Definite Focus) failed.')
402         userecallfocus = False
403
404
405 ##### START DETAILED SCAN EXPERIMENT #####
406
407 # get the actual Focus position
408 zpos = Zen.Devices.Focus.ActualPosition
409
410 # check for the column 'ID' which is required
411 ID_exits, column_ID = checktableentry(SingleObj,
412                                         entry2check='ID')
413
414
415 # execute detailed experiment at the position of every detected object
416 for i in range(0, num_POI, 1):
417
418     # get the object information from the position table
419     # get the ID of the object - IDs start with 2 !!!
420     #POI_ID = SingleObj.GetValue(i, 0)
421     POI_ID = SingleObj.GetValue(i, column_ID)
422
423     # get XY-stage position from table
424     xpos = SingleObj.GetValue(i, soi.CenterXColumnIndex)
425     ypos = SingleObj.GetValue(i, soi.CenterYColumnIndex)
426
427     # move to the current position

```

```

428 Zen.Devices.Stage.MoveTo(xpos + dx_detector, ypos + dy_detector)
429 print('Moving Stage to Object ID:', POI_ID, ' at :', '%.2f' % xpos, '%.2f' % ypos)
430
431 # try to apply RecallFocus (DF only) when this option is used
432 if userecallfocus:
433     try:
434         # apply RecallFocus for the current position
435         Zen.Acquisition.RecallFocus()
436         zpos = Zen.Devices.Focus.ActualPosition
437         print('Recall Focus (Definite Focus) applied.')
438         print('Updatd Z-Position: ', zpos)
439     except ApplicationException as e:
440         print('Application Exception : ', e.Message)
441         print('RecallFocus (Definite Focus) failed.')
442
443     print('New Z-Position before Detail Experiment will start:', zpos)
444
445 # if DetailScan is a Tile Experiment
446 if DetailIsTileExp:
447
448     print('Detailed Experiment contains TileRegions.')
449     # Modify tile center position - get bounding rectangle width and height in microns
450     bcwidth = SingleObj.GetValue(i, soi.WidthColumnIndex)
451     bcheight = SingleObj.GetValue(i, soi.HeightColumnIndex)
452     print('Width and Height : ', '%.2f' % bcwidth, '%.2f' % bcheight)
453     print('Modifying Tile Properties XYZ Position and width and height.')
454
455     # Modify the XYZ position and size of the TileRegion on-the-fly
456     print('Starting Z-Position for current Object: ', '%.2f' % zpos)
457     print('New Tile Properties: ', '%.2f' % xpos, '%.2f' % ypos, '%.2f' % zpos, '%.2f' % bcwidth, '%.2f' % bcheight)
458     DetailScan_reloaded.ClearTileRegionsAndPositions(blockindex)
459     try:
460         DetailScan_reloaded.AddRectangleTileRegion(blockindex, xpos, ypos, bcwidth, bcheight, zpos)
461     except ApplicationException as e:
462         print('Application Exception : ', e.Message)
463
464 if not DetailIsTileExp:
465     print('Detailed Experiment does not contains TileRegions. Nothing to modify.')
466
467 # execute the DetailScan experiment
468 print('Running Detail Scan Experiment at new XYZ position.')
469 try:
470     output_detailscan = Zen.Acquisition.Execute(DetailScan_reloaded)
471 except ApplicationException as e:
472     print('Application Exception : ', e.Message)
473
474 # get the image data name
475 dtscan_name = output_detailscan.Name
476
477 """
478 Modification for multiscene images: container name needs to be part
479 of the filename of the detailed Scan.
480 First check if Column "Image Scene Container Name" is available
481 and then add Container Name to filename.
482 """
483
484 wellid_exist, column_wellid = checktableentry(SingleObj,
485                                             entry2check='ImageSceneContainerName')
486
487 # in case Image Scene Container Name is not defined
488 if not wellid_exist:
489     message = 'Missing column ImageSceneContainerName in Image Analysis Results.\nPlease Select Features inside the Image'
490     message += 'Analysis when needed.'
491     print(message)
492     container_name = 'empty'
493
494 # save the image data to the selected folder and close the image
495 output_detailscan.Save(Path.Combine(OutputFolder, output_detailscan.Name))
496 output_detailscan.Close()
497
498 # rename the CZI regarding to the object ID - Attention - IDs start with 2 !!!
499 if not wellid_exist:
500     # no wellID found inside table
501     newname_dtscan = 'DTScan_ID' + str(POI_ID) + '.czi'
502 if wellid_exist:
503     # wellID was found inside table
504     well_id = SingleObj.GetValue(i, column_wellid)
505     newname_dtscan = 'DTScan_Well_' + str(wellid) + '_ID_' + str(POI_ID) + '.czi'
506
507 print('Renaming File: ' + dtscan_name + ' to: ' + newname_dtscan + '\n')
508 File.Move(Path.Combine(OutputFolder, dtscan_name), Path.Combine(OutputFolder, newname_dtscan))
509 ##### OPTIONAL POSTPROCESSING #####
510
511 if do_postprocess:
512
513     # do the postprocessing
514     image2process = Zen.Application.LoadImage(newname_dtscan, False)

```

```
516 # define the parameters for processing: Topography Export
517 parameters = {}
518 parameters['noise_low'] = 1
519 parameters['noise_high'] = 254
520 parameters['outputfolder'] = OutputFolder
521 parameters['extension'] = '.sur'
522
523 # run the processing and export and close the image
524 run_postprocessing(image2process, func='topography', parameters)
525 image2process.Close()
526
527 ##### END DETAILED SCAN EXPERIMENT #####
528
529 # restore the original OVScan experiment
530 OVScan_orig = Zen.Acquisition.Experiments.GetByName(OverViewExpName)
531 OVScan_orig.SetActive()
532
533 # delete the temporary experiments when all loops are finished
534 Zen.Acquisition.Experiments.Delete(OVScan_reloaded)
535 Zen.Acquisition.Experiments.Delete(DetailScan_reloaded)
536
537 # show the overview scan document again at the end
538 Zen.Application.Documents.ActiveDocument = ovdoc
539 print('All Positions done. Guided Acquisition Workflow finished.')
```

8 ToDo's and Limitations

The current version of this tool has still limitations and room for improvement.

- No yet built-in check to avoid double detailed scans for objects close to each other within one field of view.
- Currently only modification of rectangular tile regions are supported.

9 Disclaimer

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

Especially be aware of the fact that automated stage movements might damage hardware if the system is not setup properly and not correctly calibrated.

Please check everything in simulation mode first!

Carl Zeiss Microscopy GmbH's ZEN software allows connection to 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 and using this script confirm that you understood the involved risks you agree to this disclaimer.