

# Guided Acquisition

Advanced Automated Microscopy



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

[microscopy@zeiss.com](mailto:microscopy@zeiss.com)  
[www.zeiss.com/microscopy](http://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	ZEN Prerequisites	9
2.2	Acquire Sample Data with Overview Scan	10
2.3	Create Image Analysis Pipeline	12
2.4	Define Overview Experiment	14
2.5	Define Detailed Scan Experiment	15
3	Automation via OAD	16
3.1	Prerequisites	16
3.2	Pure Scripting or User Dialog	16
3.3	User Dialog Requirements	16
3.4	Create the User Dialog	18
3.5	Overview Scan Experiment	20
3.6	Find the interesting Objects	21
3.7	Prepare Detail Scan	23
3.8	Modify the Tile Dimensions	24
3.9	Run the Detailed Scan	25
4	ZEN Blue Software Module for Guided Acquisition	26
5	Testrun with FluoCells Slide	27
6	Test Run with Brain Slide	29
7	Mitosis Detection with Camera and LSM	31
8	Appendix	33
8.1	Complete Workflow Diagram	33
8.2	Python Script: Guided_Acquisition_shortUI.py	35
9	ToDo's and Limitations	48
10	Disclaimer	48

## 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.

Depending on the application the image analysis step can be done by using

- ZEN Image Analysis incl. Machine-Learning methods (Guided Acquisition module)
- Running an APEER module inside a docker container (script version)
- Calling an external application (script version)

### INFO:

The option Running an APEER module required to have Docker installed on the system, an APEER account and obviously a module that created a table containing the correct stage coordinated for the object of interest. For more info on APEER please visit: <http://www.apeer.com>

## 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 or ...
3. ... create an APEER module
4. Define an experiment which does the Overview Scan.
5. Define an experiment which does the Detailed Scan.
6. 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.

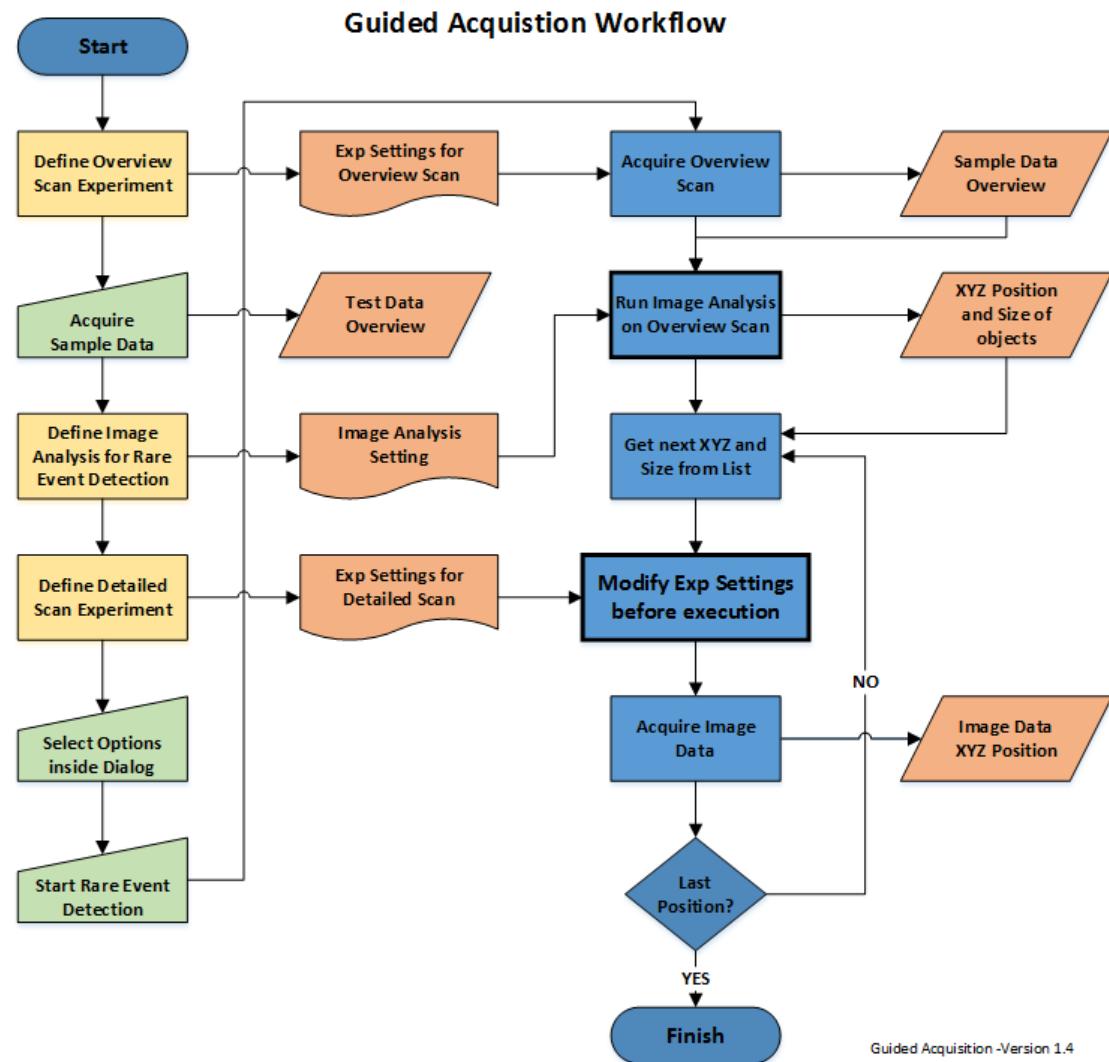


Figure 1: General workflow for Guided Acquisition.

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

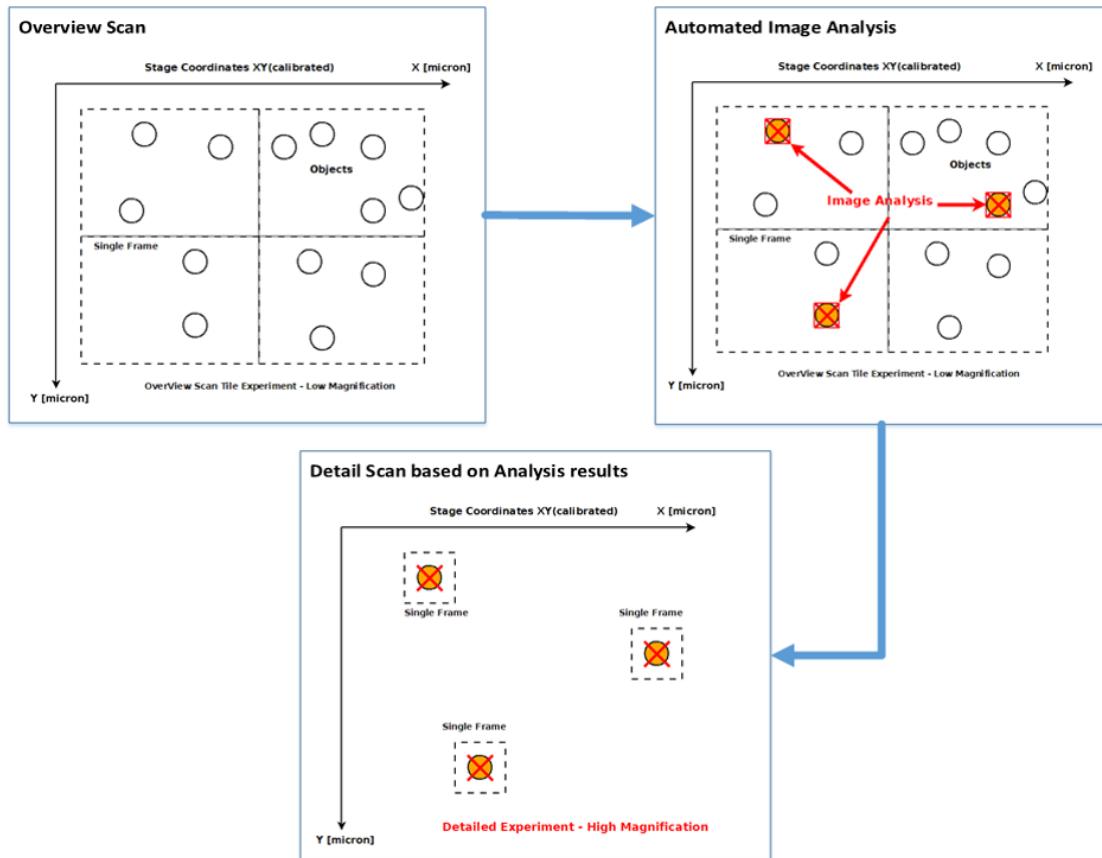


Figure 2: Larger area with some candidates for a "interesting" objects.

### 1.3 General Workflow - Complete Overview

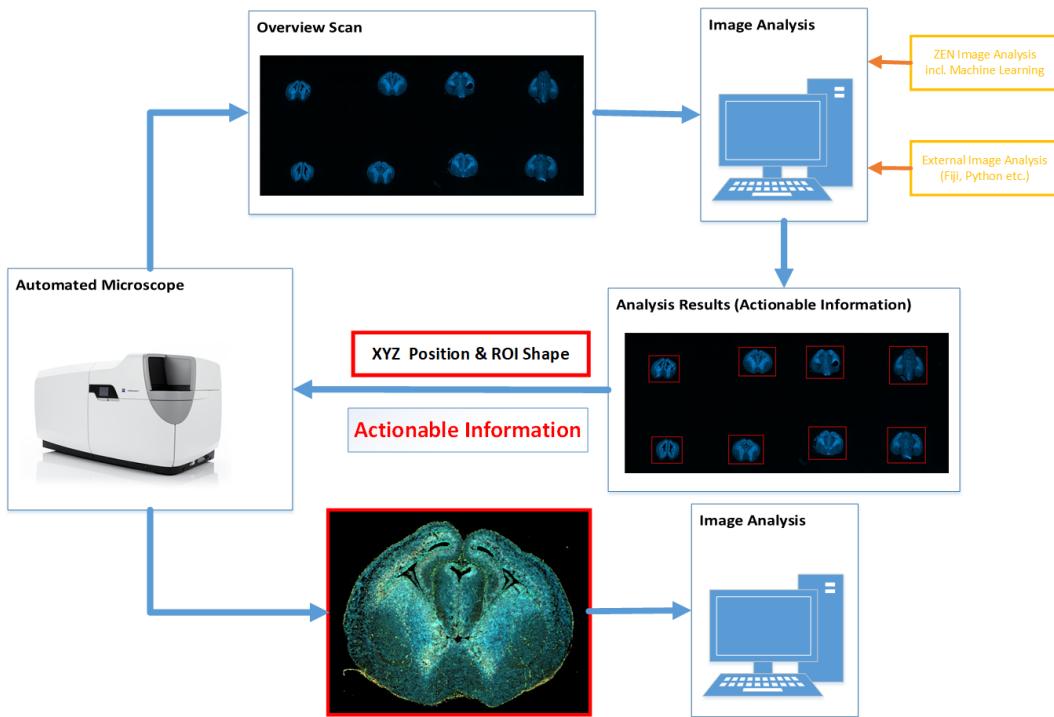


Figure 3: Workflow - Actionable Information 1.

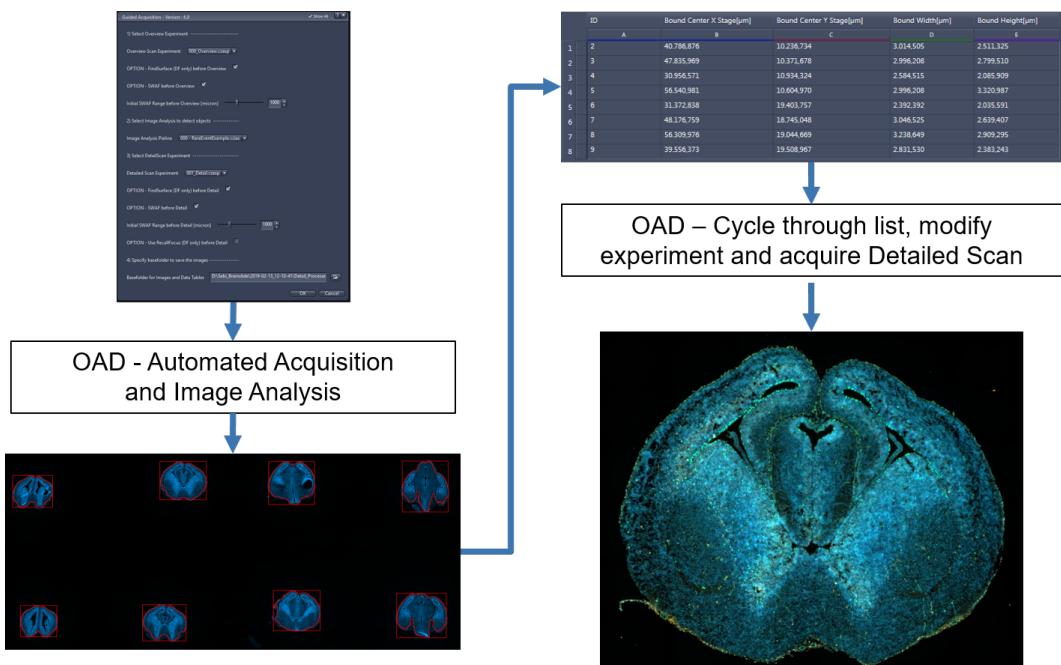


Figure 4: Workflow - Actionable Information 2

## 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 ZEN Prerequisites

For automated workflows like Guided Acquisition is it mandatory to switch off any modal windows, that would require the user to press a button during the preparation of an experiment run. An example is the reminder message box during the initialization of the Definite Focus.

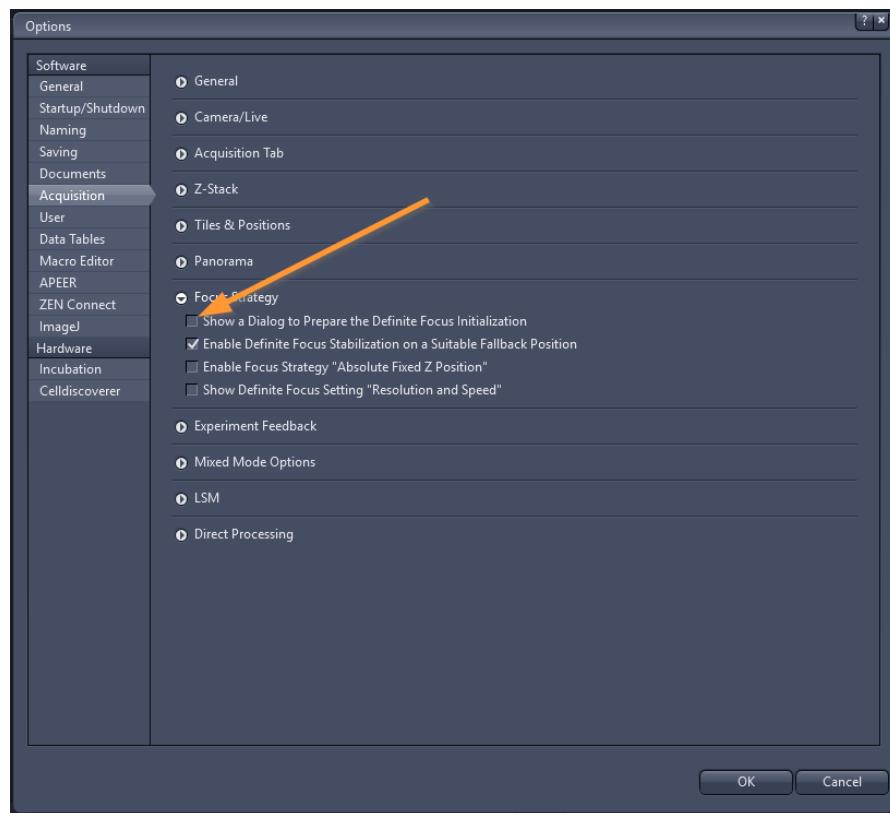


Figure 5: Prerequisites - Disable Message Box

#### INFO:

The option Show a Dialog to Prepare the Definite Focus Initialization should be disabled. The only exception would be using Focus Strategies using multiple z-Offsets that require user interaction in setting them up.

## 2.2 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 ZEN Analysis Wizard.
2. Program your own image analysis pipeline using an OAD macro (this is not covered by this tutorial).
3. Use an APEER module to detect the objects of interest and create the required table
4. 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 and allows to use trained machine-learning models to segment the image as well.

**INFO:**

AN example for an APEER module suitable for the Guided Acquisition workflow can be found here: <https://github.com/zeiss-microscopy/OAD/tree/master/Apeer/modules/segmentobjects>.

## 2.3 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.

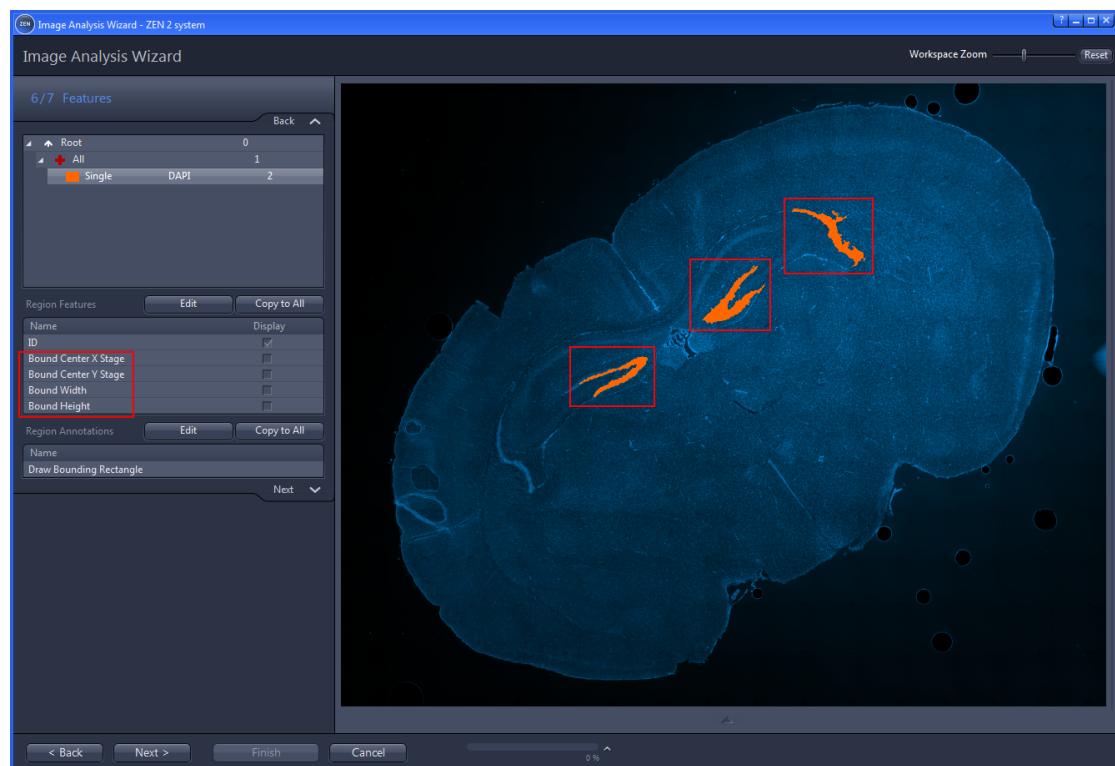


Figure 6: Select the required positional features inside the wizard.

Currently one needs to select the following parameters when creating rectangular TileRegions bases on the Image Anaylsis:

- 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 in [micron]. 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.

When choosing the option to create Polygon TileRegions BoundWidth and Bound Height are not required but it is recommended to include them anyway

## 2.4 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:

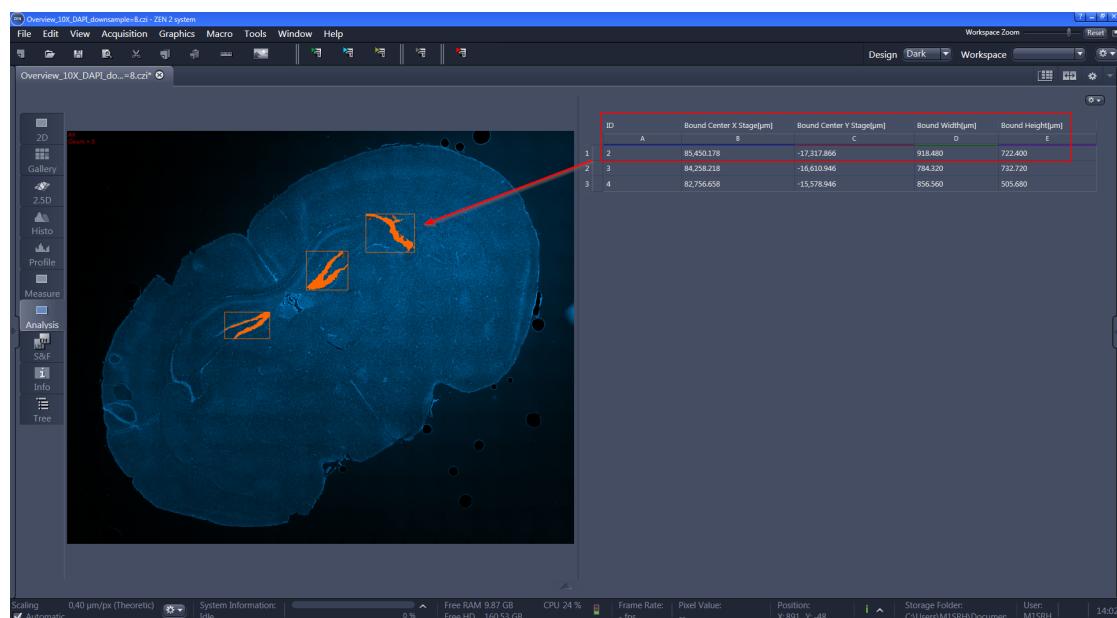


Figure 7: Result of the image analysis on the overview scan image.

## 2.5 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.

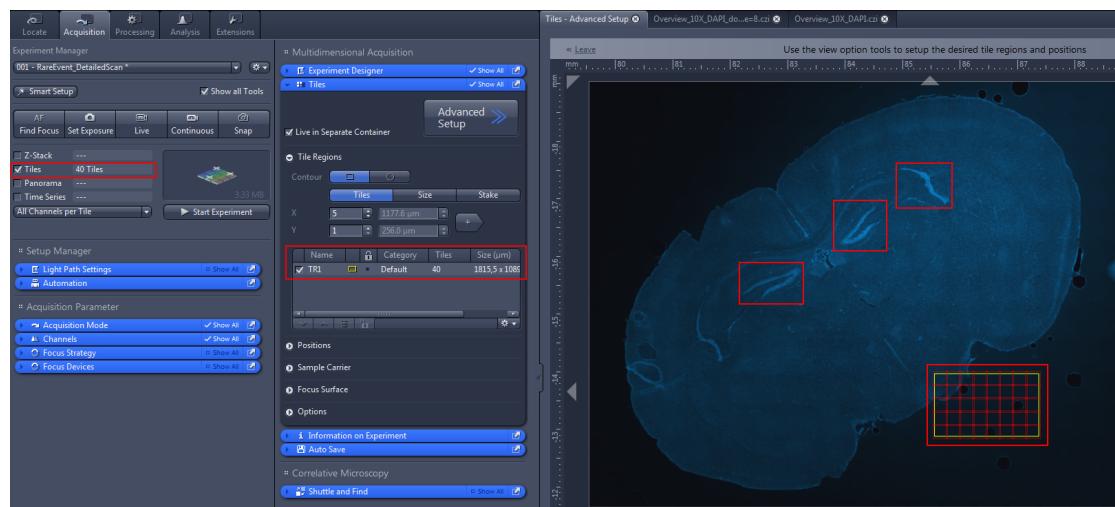


Figure 8: Setup of a simple detailed scan experiment.

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 user interface of the dialog will following functionality:

1. Select the overview scan experiment.
2. Option to use Definite Focus (DF2) before overview scan
3. Option to use SWAF before Overview
4. Define range for SWAF
5. Define ZEN Image Analysis Settings to be used
6. Option to Polygon TileRegions instead of BoundingBox
7. Option to use APEER module instead of ZEN Image Analysis
8. Define the APEER module setting to be used
9. Select the detail scan experiment

10. Option to use Definite Focus before overview scan
11. Option to use SWAF before overview scan
12. Define range for SWAF
13. Option to use Recall Focus using (DF2)before detail scan
14. Select a folder to store the image data and analysis results

### 3.4 Create the User Dialog

The relevant part of the script is shown below. 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.

```
540 # Initialize Dialog
541 GuidedAcqDialog = Zenwindow()
542 GuidedAcqDialog.Initialize("Guided Acquisition - Version : " + str(version))
543 # add components to dialog
544 GuidedAcqDialog.AddLabel("----- Select Overview Experiment -----")
545 GuidedAcqDialog.AddDropDown('overview_exp', 'Overview Scan Experiment', expfiles_short, 0)
546 GuidedAcqDialog.AddCheckbox('fs_before_overview', 'OPTION - FindSurface (DF only) before Overview', False)
547 GuidedAcqDialog.AddCheckbox('SWAF_before_overview', 'OPTION - SWAF before Overview', False)
548 GuidedAcqDialog.AddIntegerRange('SWAF_ov_initial_range', 'Initial SWAF Range before Overview [micron]', 200, 50, 3000)
549 GuidedAcqDialog.AddLabel("----- Select Image Analysis to detect objects -----")
550 GuidedAcqDialog.AddDropDown('ip_pipe', 'Image Analysis Pipeline', ipfiles_short, 0)
551 GuidedAcqDialog.AddCheckbox('use_poly', 'Use Polygons from ZEN IA instead of BBox', True)
552 GuidedAcqDialog.AddCheckbox('use_apeer_for_IA', 'Use APEER module run run IA instead', False)
553 GuidedAcqDialog.AddDropDown('ap_pipe', 'APEER Module Settings', apfiles_short, 0)
554 GuidedAcqDialog.AddLabel("----- Select DetailScan Experiment -----")
555 GuidedAcqDialog.AddDropDown('detailed_exp', 'Detailed Scan Experiment', expfiles_short, 1)
556 GuidedAcqDialog.AddCheckbox('fs_before_detail', 'OPTION - FindSurface (DF only) before Detail', False)
557 GuidedAcqDialog.AddCheckbox('SWAF_before_detail', 'OPTION - SWAF before Detail', False)
558 GuidedAcqDialog.AddIntegerRange('SWAF_detail_initial_range', 'Initial SWAF Range before Detail [micron]', 100, 10, 1000)
559 GuidedAcqDialog.AddCheckbox('recallfocus_beforeDT', 'OPTION - Use RecallFocus (DF only) before Detail', False)
560 GuidedAcqDialog.AddLabel("----- Specify basefolder to save the images -----")
561 GuidedAcqDialog.AddFolderBrowser('outfolder', 'Basefolder for Images and Data Tables', imgfolder)
562
563 # show the window
564 result = GuidedAcqDialog.Show()
565 if result.HasCanceled:
566     message = 'Macro was canceled by user.'
567     print(message)
568     raise SystemExit
569
570 # get the values and store them
571 OverViewExpName = str(result.GetValue('overview_exp'))
572 ImageAS = str(result.GetValue('ip_pipe'))
573 ApeerMS = str(result.GetValue('ap_pipe'))
574 DetailExpName = str(result.GetValue('detailed_exp'))
575 OutputFolder = str(result.GetValue('outfolder'))
576 fs_beforeOV = result.GetValue('fs_before_overview')
577 SWAF_beforeOV = result.GetValue('SWAF_before_overview')
578 SWAF_beforeOV_range = result.GetValue('SWAF_ov_initial_range')
579 fs_beforeDT = result.GetValue('fs_before_detail')
580 SWAF_beforeDT = result.GetValue('SWAF_before_detail')
581 SWAF_beforeDT_range = result.GetValue('SWAF_detail_initial_range')
582 RecallFocus = result.GetValue('recallfocus_beforeDT')
583 use_apeer = result.GetValue('use_apeer_for_IA')
584 use_polygon = result.GetValue('use_poly')
```

The resulting dialog window is shown below.

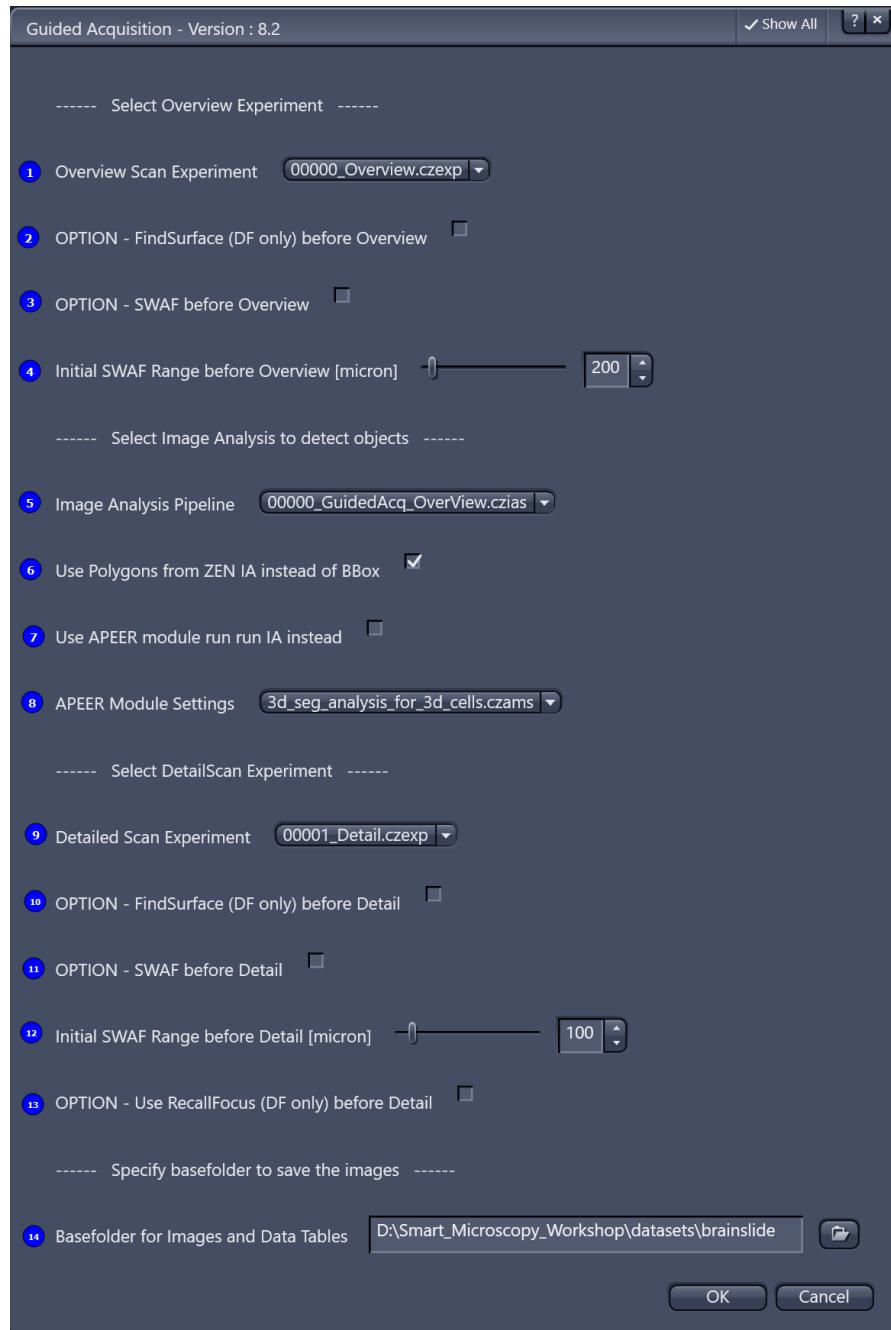


Figure 9: Simple User Dialog to run the Guided Acquisition WorkFlow (Remark: Version in screenshot may differ from the actual latest version)

### 3.5 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.

```
611 ##### START OVERVIEW SCAN EXPERIMENT #####
612
613 if fs_beforeOV:
614     # initial focussing via FindSurface to assure a good starting position
615     Zen.Acquisition.FindSurface()
616     print('Z-Position after FindSurface: ', Zen.Devices.Focus.ActualPosition)
617
618 if SWAF_beforeOV:
619     zSWAF = runSWAF_special(OVScan_reloaded,
620                             delay=hwdelay,
621                             searchStrategy='Full',
622                             sampling=ZenSoftwareAutofocusSampling.Coarse,
623                             relativeRangeIsAutomatic=False,
624                             relativeRangeSize=SWAF_beforeOV_range,
625                             timeout=1)
626
627 # get the resulting z-position
628 znew = Zen.Devices.Focus.ActualPosition
629
630 # adapt the Overview Scan Tile Experiment with new Z-Position
631 if OVScanIsTileExp:
632     OVScan_reloaded.ModifyTileRegionsZ(blockindex, znew)
633     print('Adapted Z-Position of Tile OverView. New Z = ', '%.2f' % znew)
634
635 # execute the experiment
636 print('\nRunning OverviewScan Experiment.\n')
637 output_OVScan = Zen.Acquisition.Execute(OVScan_reloaded)
638 # For testing purposes - Load overview scan image automatically instead of executing the "real" experiment
639 #ovtestimage = r'D:\GA_test_SRH\ov_ia.czi'
640 #output_OVScan = Zen.Application.LoadImage(ovtestimage, False)
641
642 # the the stage top-left and the scaling of the overview image
643 stageTL = output_OVScan.GetPositionLeftTop()
644 ovscaling = output_OVScan.Scaling
645
646 # show the overview scan inside the document area
647 Zen.Application.Documents.Add(output_OVScan)
648 ovdoc = Zen.Application.Documents.GetByName(output_OVScan.Name)
649
650 # save the overview scan image inside the select folder
651 savepath_ovscan = Path.Combine(OutputFolder, ovscan_name)
652 output_OVScan.Save(savepath_ovscan)
653
654 # get the actual focus value for the overscan independent from the z-value
655 # before the start of the overview scan using the image metadata
656 zvalue_ovscan = output_OVScan.Metadata.FocusPositionMicron
657
658 ##### END OVERVIEW SCAN EXPERIMENT #####
659
```

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.

### 3.6 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.

```
660 # run normal ZEN image analysis
661 if not use_apeer:
662
663     # Load analysis setting created by the wizard or an separate macro
664     ias = ZenImageAnalysisSetting()
665
666     # for simulation use: 000 - RareEventExample.czias
667     ias.Load(ImageAS)
668
669     # Analyse the image
670     Zen.Analyzing.Analyze(output_OVScan, ias)
671
672     if use_polygon:
673
674         # get classes and derive regions names from that
675         iasclasses = getClassnames(ias)
676
677         # get the individual image analysis regions from 1st IA class (!!!)
678         # ATTENTION: the first single objects class will be used. It has ID = 2
679         regions = Zen.Analyzing.GetRegions(output_OVScan, iasclasses[2])
680         print('RegionClassNames: ', iasclasses)
681         print('Analysis found ' + str(regions.Count) + 'regions!')
682
683         # create dictionary for polygon regions
684         polyregions = {}
685
686         # loop over all regions and get the points of polygon (outline of the object)
687         for i in range(0, regions.Count):
688             points = regions[i].GetPolygon()
689
690             # convert points to stage coordinates
691             polyregions[i] = create_exppolygon(points, stageTL, ovscaling)
692
693         # Create Zen table with results for all detected objects (parent class)
694         AllObj = Zen.Analyzing.CreateRegionsTable(output_OVScan)
695
696         # Create Zen table with results for each single object
697         SingleObj = Zen.Analyzing.CreateRegionTable(output_OVScan)
698
699         # check for existence of required column names inside table
700         soi = SingleObj.GetColumnInfoFromImageAnalysis(True)
701
702         # 1st item is a bool indicating if all required columns could be found
703         columnsOK = soi.AreRequiredColumnsAvailable
704
705         if not columnsOK:
706             print('Execution stopped. Required Columns are missing.')
707             raise Exception('Execution stopped. Required Columns are missing.')
708
709         # show and save data tables to the specified folder
710         Zen.Application.Documents.Add(AllObj)
711         Zen.Application.Documents.Add(SingleObj)
712         AllObj.Save(Path.Combine(OutputFolder, 'OverviewTable.csv'))
713         SingleObj.Save(Path.Combine(OutputFolder, 'SingleObjectsTable.csv'))
714
715 # use APEER module to detect the objects
716 if use_apeer:
717
718     # read it from settings file
719     ams = ZenApeer.Onsite.ModuleSetting()
720
721     # loaf the APEER module setting b - remove *.czams file extension first
722     ams.Load(Path.GetFileNameWithoutExtension(ApeerMS))
723
724     print('----- Apeer Module Setting -----')
725     print('Module Name : ', ams.ModuleName)
726     print('Module Version : ', ams.ModuleVersion)
727     print('Module Parameters : ', ams.Parameters)
728
729     # get module and check
730     mymodule, version_found = get_module(ams.ModuleName, module_version=ams.ModuleVersion)
731
732     # exit if the check failed
733     if mymodule is None or not version_found:
```

```

734     print('Module check failed. Exiting.')
735     raise SystemExit
736
737     # get the module parameters for the specified module
738     module_params = ZenApeer.Onsite.GetSampleModuleParameters(ams.ModuleName, ams.ModuleVersion)
739
740     # get the module input programmatically
741     module_inputs = get_module_inputs(module_params)
742
743     # create the required dictionary with the correct key and value
744     input_image = {module_inputs[0]: savepath_ovscan}
745
746     # create the path to save the results
747     #savepath_apeer = Path.Combine(OutputFolder, 'apeer_results')
748     savepath_apeer = OutputFolder
749
750     # create the output directory if not existing already
751     if not Directory.Exists(savepath_apeer):
752         Directory.CreateDirectory(savepath_apeer)
753
754     # run the local APEER module with using keywords
755     try:
756         runoutputs, status, log = ZenApeer.Onsite.RunModule(moduleName=ams.ModuleName,
757                                                       moduleVersion=ams.ModuleVersion,
758                                                       inputs=input_image,
759                                                       parameters=ams.Parameters,
760                                                       storagePath=savepath_apeer)
761
762         for op in runoutputs.GetEnumerator():
763             print('---- Outputs ----')
764             print(op.Key, ' : ', op.Value)
765
766     except ApplicationException as e:
767         print('Module Run failed.', e.Message)
768         raise SystemExit
769
770     # get results storage locations
771     # IMPORTANT: To be used inside Guided Acquisition
772     # the APEER Module is expected to have
773     # at least those to outputs with exactly those names
774
775     if not runoutputs.ContainsKey('segmented_image'):
776         print('No output : segmented_image')
777         raise SystemExit
778
779     if not runoutputs.ContainsKey('objects_table'):
780         print('No output : objects_table')
781         raise SystemExit
782
783     # load the segmented image and make sure the pyramid is calculated
784     segmented_image = Zen.Application.LoadImage(runoutputs['segmented_image'], False)
785     Zen.Processing.Utilities.GenerateImagePyramid(segmentedImage, ZenBackgroundMode.Black)
786     Zen.Application.Documents.Add(segmentedImage)
787
788     # auto-display min-max
789     ids = segmented_image.DisplaySetting.GetAllChannelIds()
790     for id in ids:
791         segmented_image.DisplaySetting.SetParameter(id, 'IsAutoApplyEnabled', True)
792
793     # initialize ZenTable object and load CSV file
794     SingleObj = ZenTable()
795     SingleObj.Load(runoutputs['objects_table'])
796     Zen.Application.Documents.Add(SingleObj)
797
798     # get all columns as dict with columnIDs
799     colID = get_columns(SingleObj)
800
801     # define the required column names here
802     col2check = ('bbox_center_stageX', 'bbox_center_stageY', 'bbox_width_scaled', 'bbox_height_scaled')
803
804     # check if all columns exist
805     if all(key in colID for key in col2check):
806         print('All required columns found.')
807     else:
808         print('Not All required columns found. Exiting.')
809         raise SystemExit
810

```

As shown in figure 6, 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.

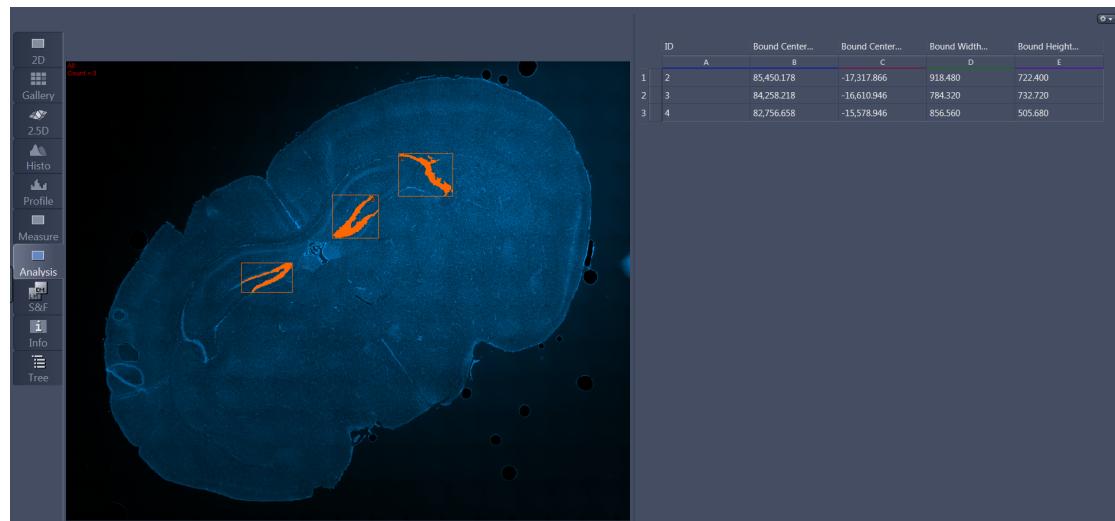


Figure 10: The results of the overview scan and the image analysis.

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.

### 3.7 Prepare Detail Scan

Before the actual loop over all objects can start one has to do the following steps:

1. Move stage to the 1st XY position from the object table
2. Move z-Drive to the actual focus position of the overview scan

### 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 detail scan experiment at the new position.
- Adapt TileRegion using a BoundingBox or ...
- ... modify TileRegion using the Polygon Points.

```
884 if not use_apeer:
885
886     # check for the column 'ID' which is required
887     ID_exists, column_ID = checktableentry(SingleObj, entry2check='ID')
888
889     # execute detailed experiment at the position of every detected object
890     for obj in range(SingleObj.RowCount):
891
892         # get the object information from the position table
893         POI_ID = SingleObj.GetValue(obj, column_ID)
894
895         # get XY-stage position from table
896         xpos = SingleObj.GetValue(obj, soi.CenterXColumnIndex)
897         ypos = SingleObj.GetValue(obj, soi.CenterYColumnIndex)
898
899         # move to the current position
900         Zen.Devices.Stage.MoveTo(xpos + dx_detector, ypos + dy_detector)
901         print('Moving Stage to Object ID:', POI_ID, ' at :', '%.2f' % xpos, '%.2f' % ypos)
902
903         # try to apply RecallFocus (DF only) when this option is used
904         if userecallfocus:
905             zpos = apply_recall_focus()
906
907         # if DetailScan is a Tile Experiment
908         if DetailIsTileExp:
909
910             print('Detailed Experiment contains TileRegions.')
911             # Modify tile center position - get bounding rectangle width and height in microns
912             bcwidth = SingleObj.GetValue(obj, soi.WidthColumnIndex)
913             bcheight = SingleObj.GetValue(obj, soi.HeightColumnIndex)
914
915             if use_polygon:
916                 # modify the experiment with the respective polygon region
917                 DetailScan_reloaded = modify_tileexperiment_polygon(DetailScan_reloaded, polyregions[obj],
918                                         blockindex=0,
919                                         zpos=zpos)
920             else:
921                 # modify the experiment
922                 DetailScan_reloaded = modify_tileexperiment_rect(DetailScan_reloaded,
923                                         blockindex=blockindex,
924                                         bcwidth=bcwidth,
925                                         bcheight=bcheight,
926                                         xpos=xpos,
927                                         ypos=ypos,
928                                         zpos=zpos)
929
930         if not DetailIsTileExp:
931             print('Detailed Experiment does not contain TileRegions. Nothing to modify.')
```

### 3.9 Run the Detailed Scan

The last steps, which need to be executed 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.

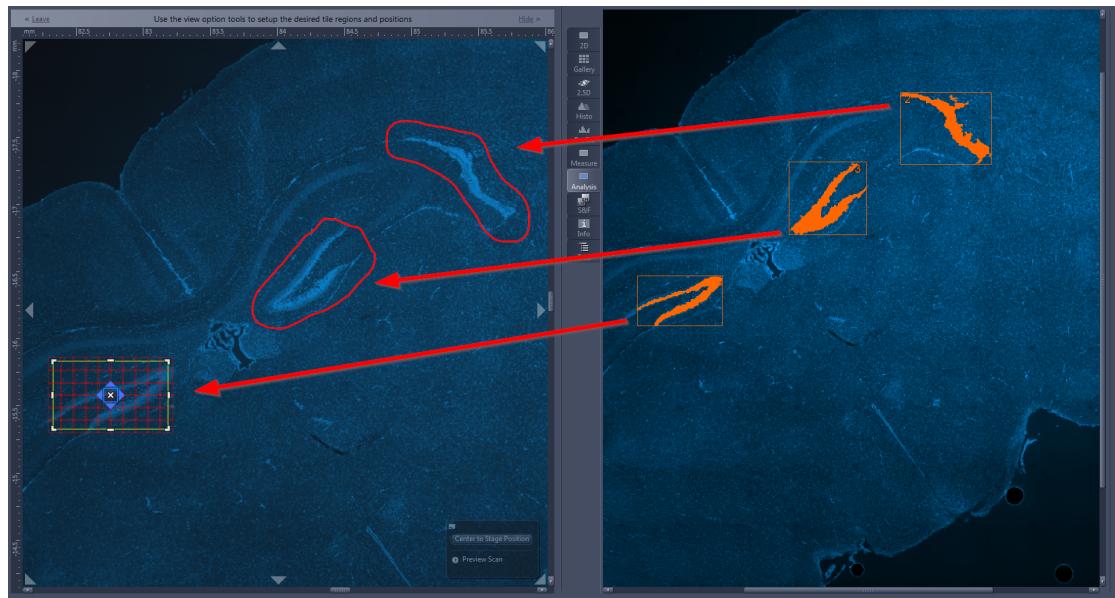


Figure 11: The results of the detailed scan - tile regions adapts to objects.

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.

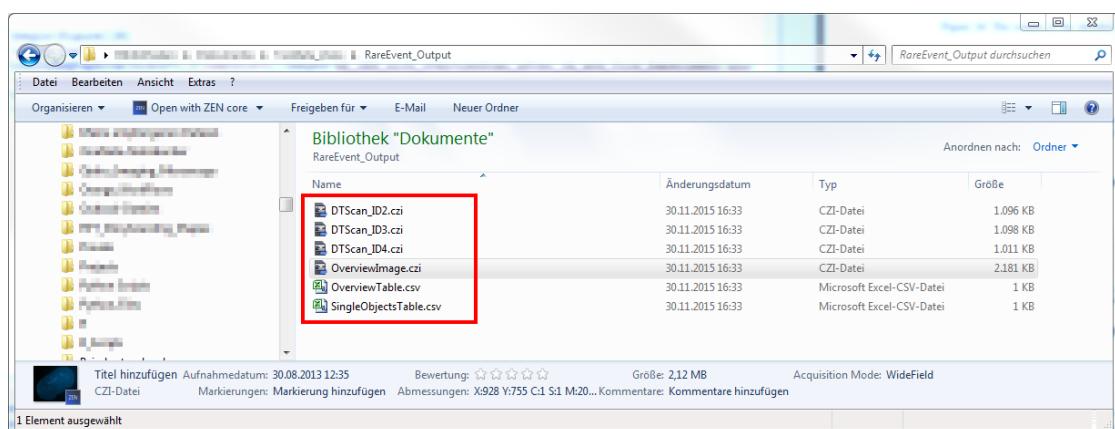


Figure 12: The results of the detailed scan stored inside a folder.

## 4 ZEN Blue Software Module for Guided Acquisition

Starting with ZEN Blue 3.1 there will be the option to purchase dedicated software module for Guided Acquisition, which has a streamlined UI interface and other additional functionality, like save settings and pre-processing functions etc.

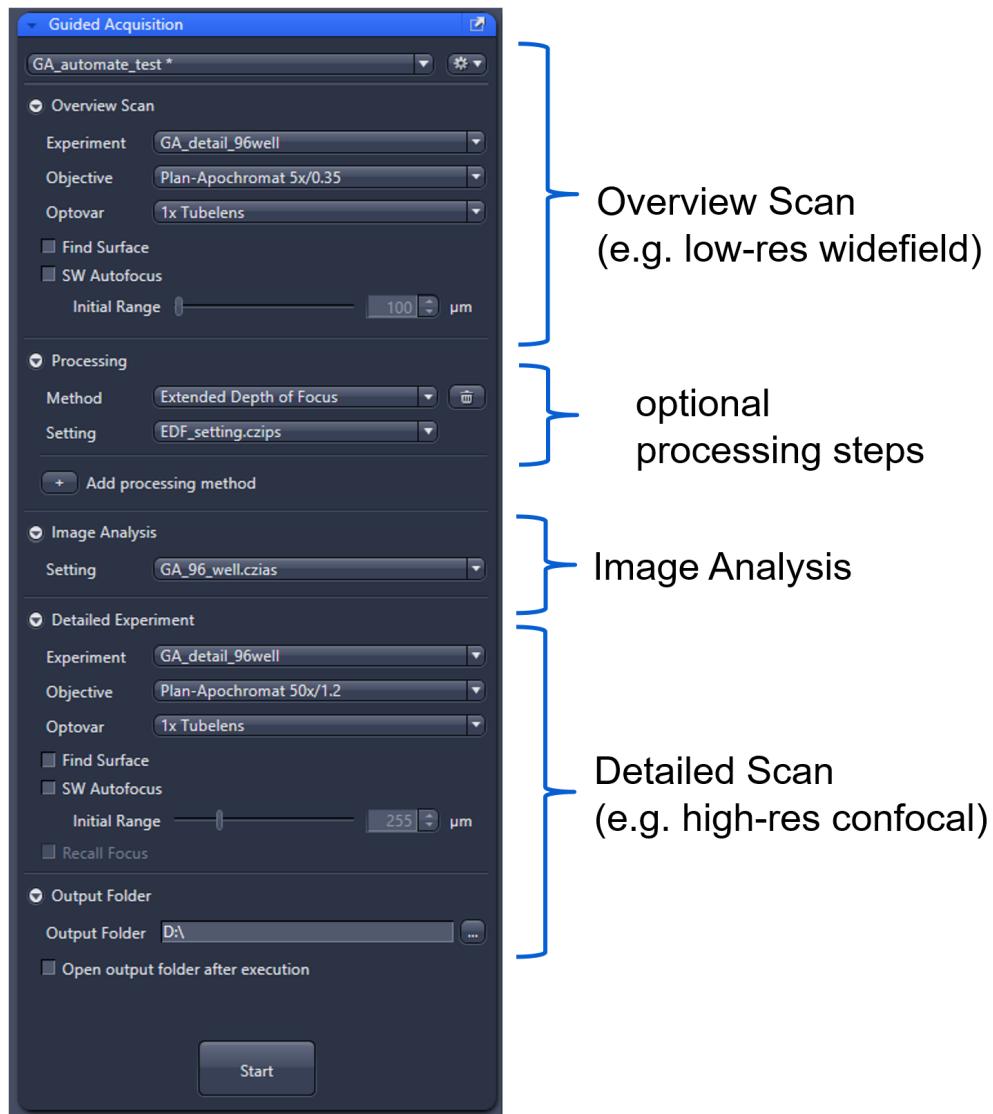


Figure 13: Software Module - Guided Acquisition (ZEN Blue 3.1 or better)

## 5 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.

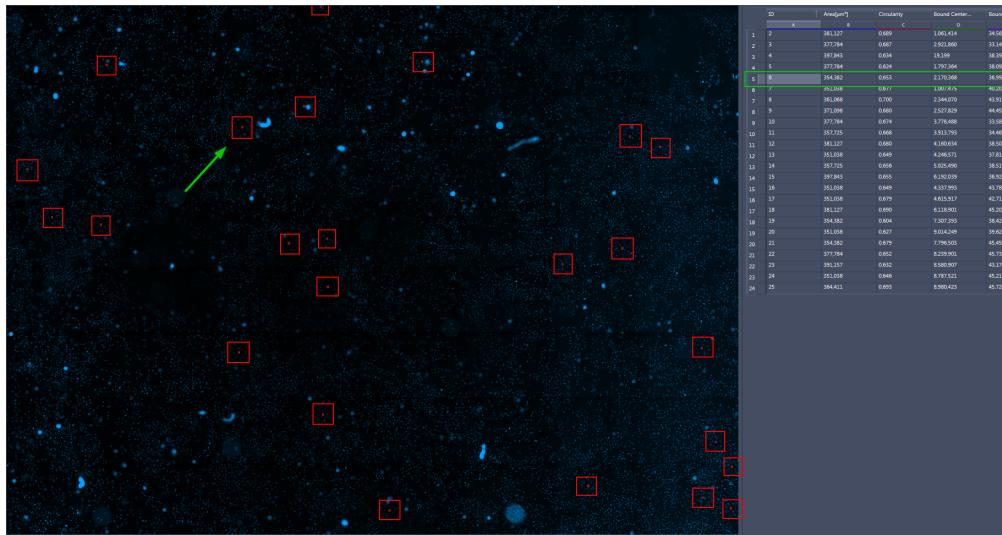


Figure 14: The results of the overview scan and image analysis.

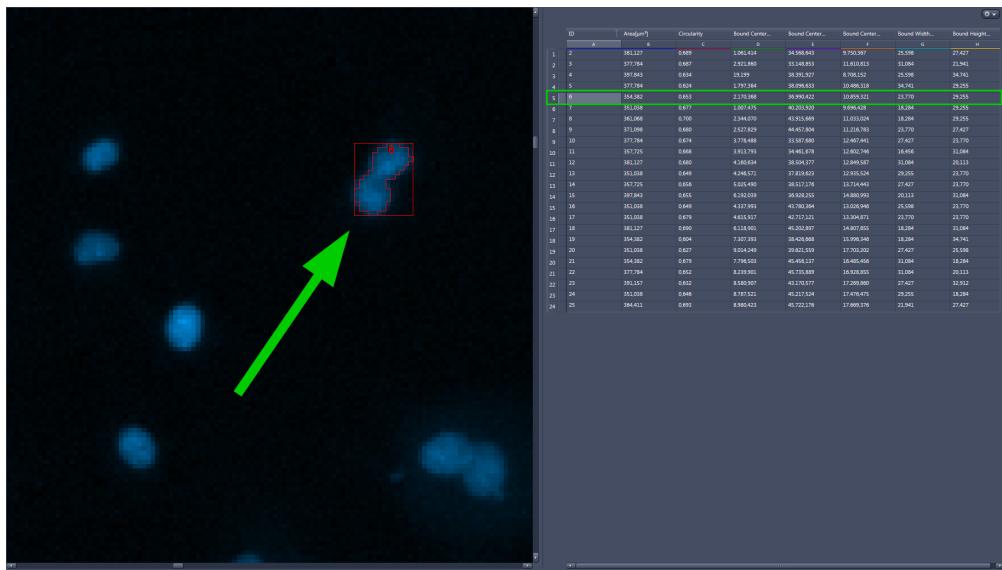


Figure 15: Detailed view on a detected object.

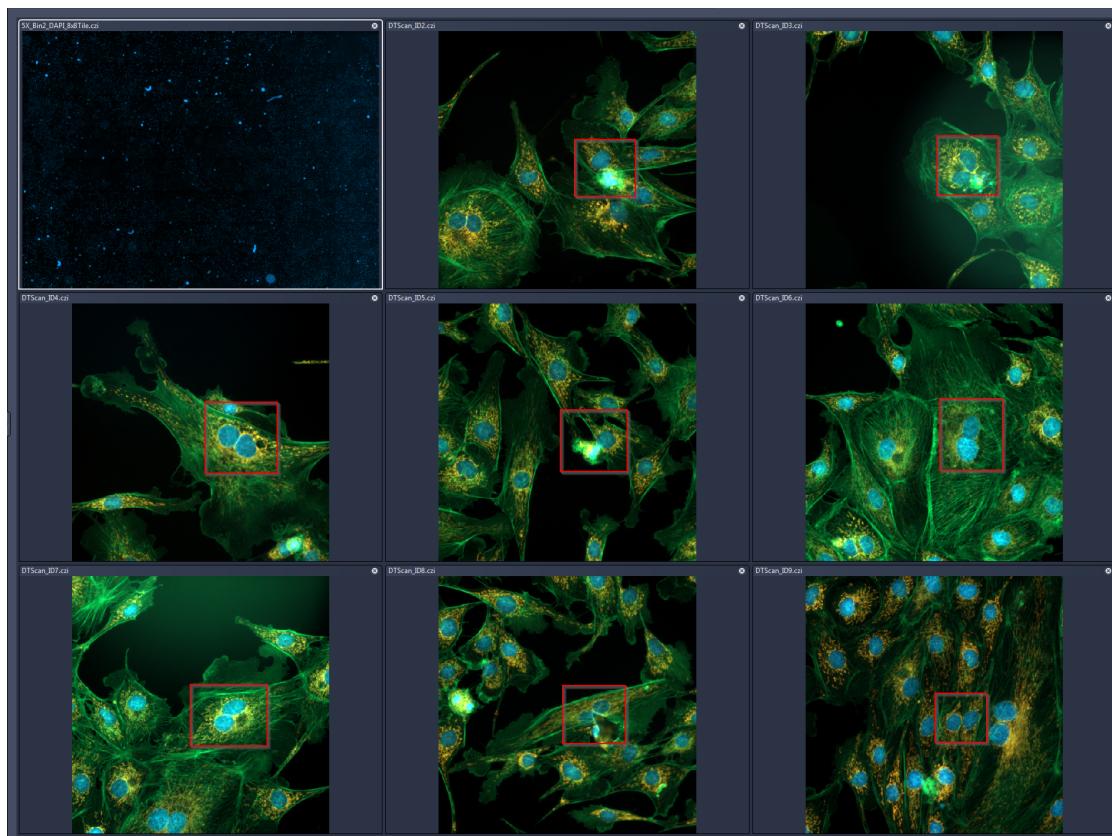


Figure 16: Detailed images from some of the detected objects.

The image analysis yielded 24 positive detection 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.

## 6 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.

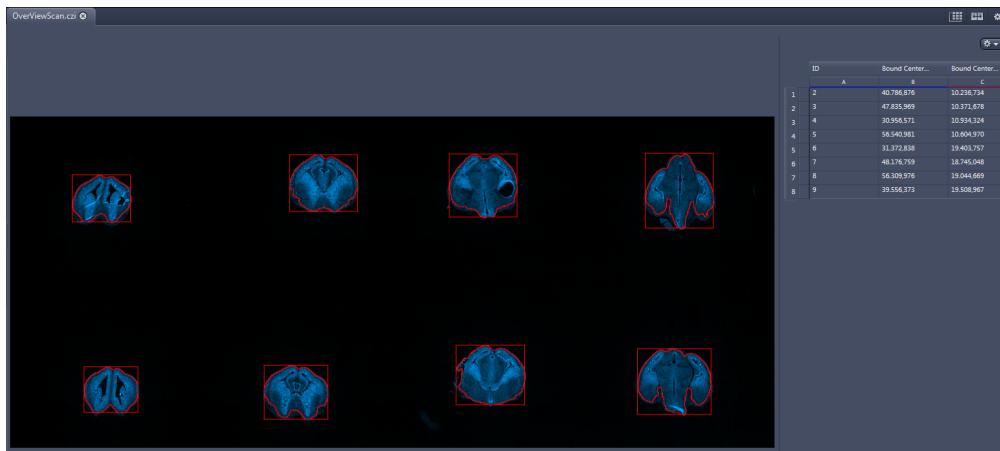


Figure 17: The results of the overview scan and image analysis.

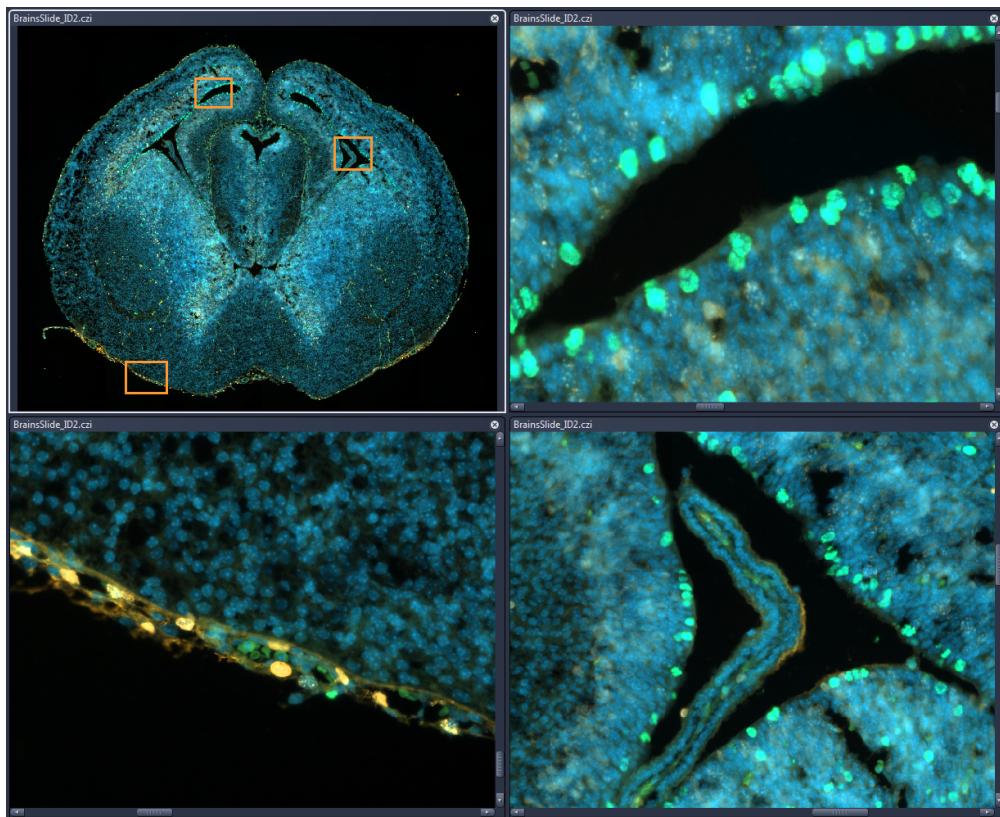


Figure 18: A more detailed image of the first specimen with three zoomed-in regions.

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.

## 7 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.

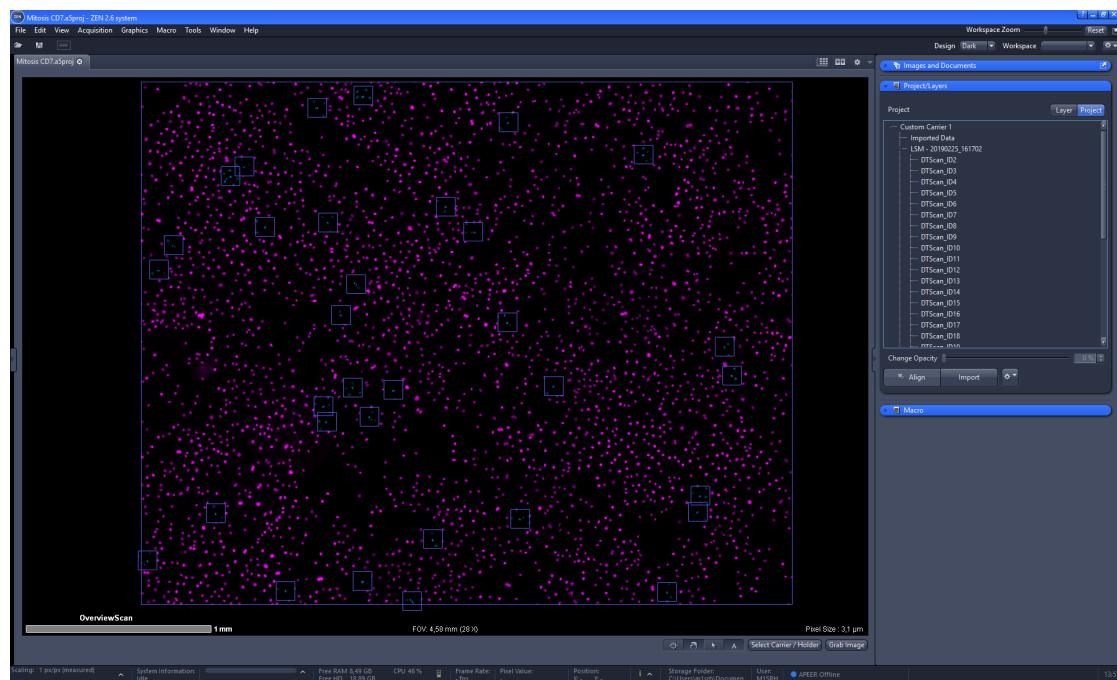


Figure 19: The results of the overview scan and image analysis inside the CWS.

## Advanced Automated Microscopy: Guided Acquisition

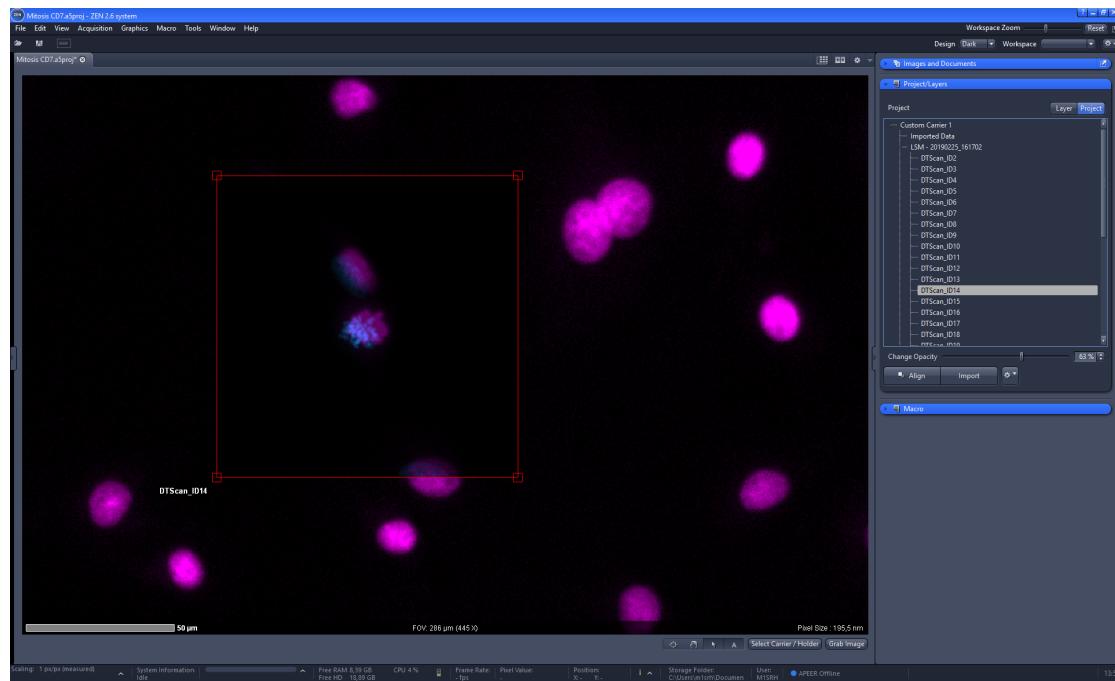


Figure 20: Overlay between positive detection inside overview scan and detailed scan using the Correlative Workspace (CWS).

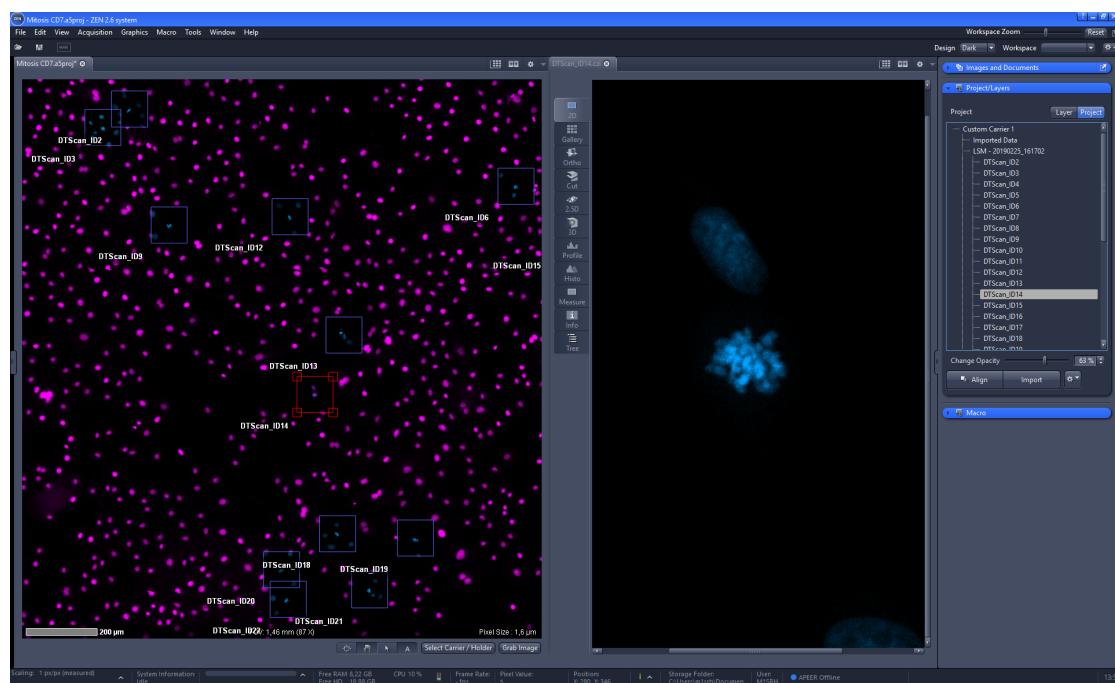


Figure 21: Side-by-Side view of the CWS and the normal view inside ZEN Blue.

8 Appendix

## 8.1 Complete Workflow Diagram

This is the complete workflow diagram for Guided Acquisition.

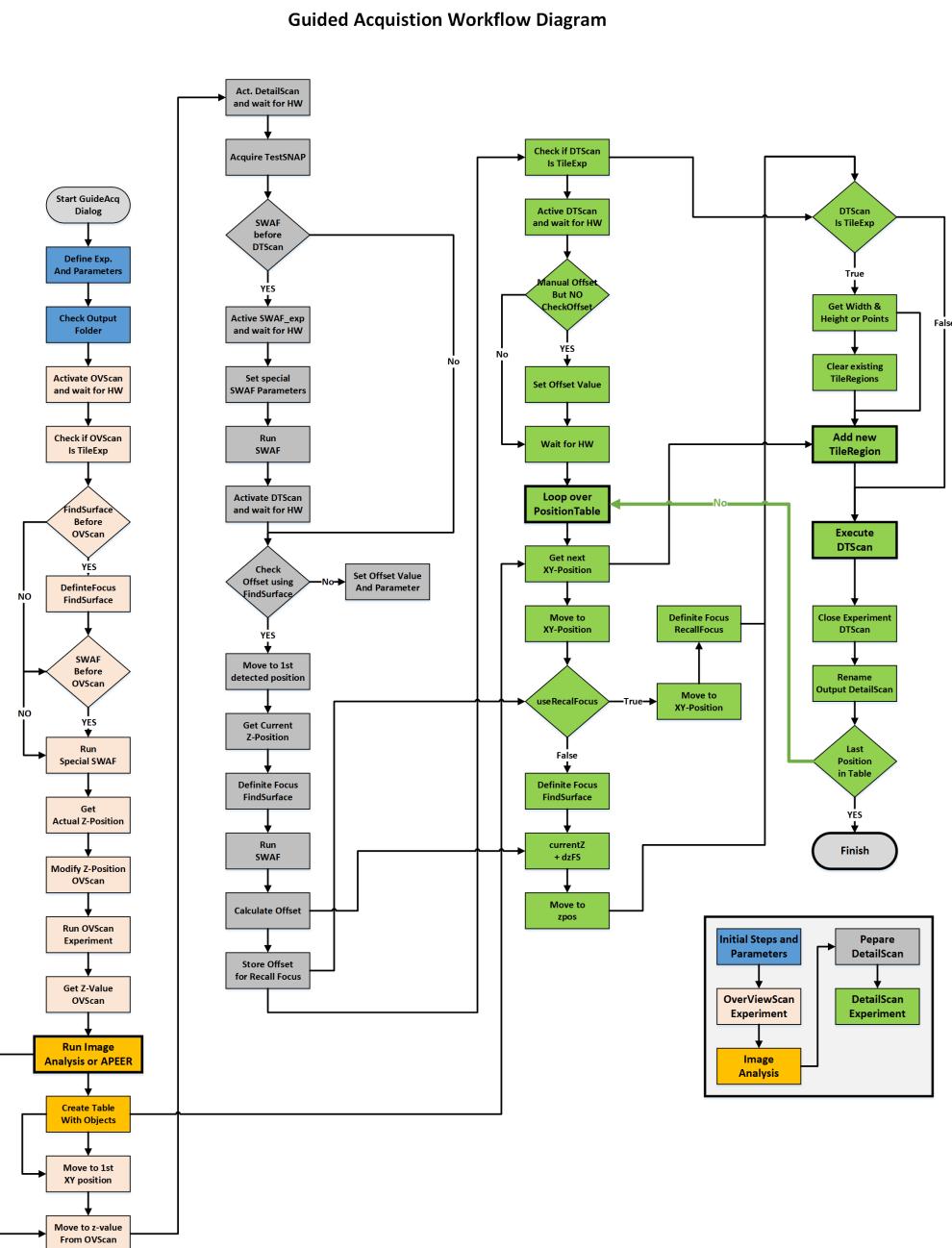


Figure 22: Complete workflow diagram for Guided Acquisition

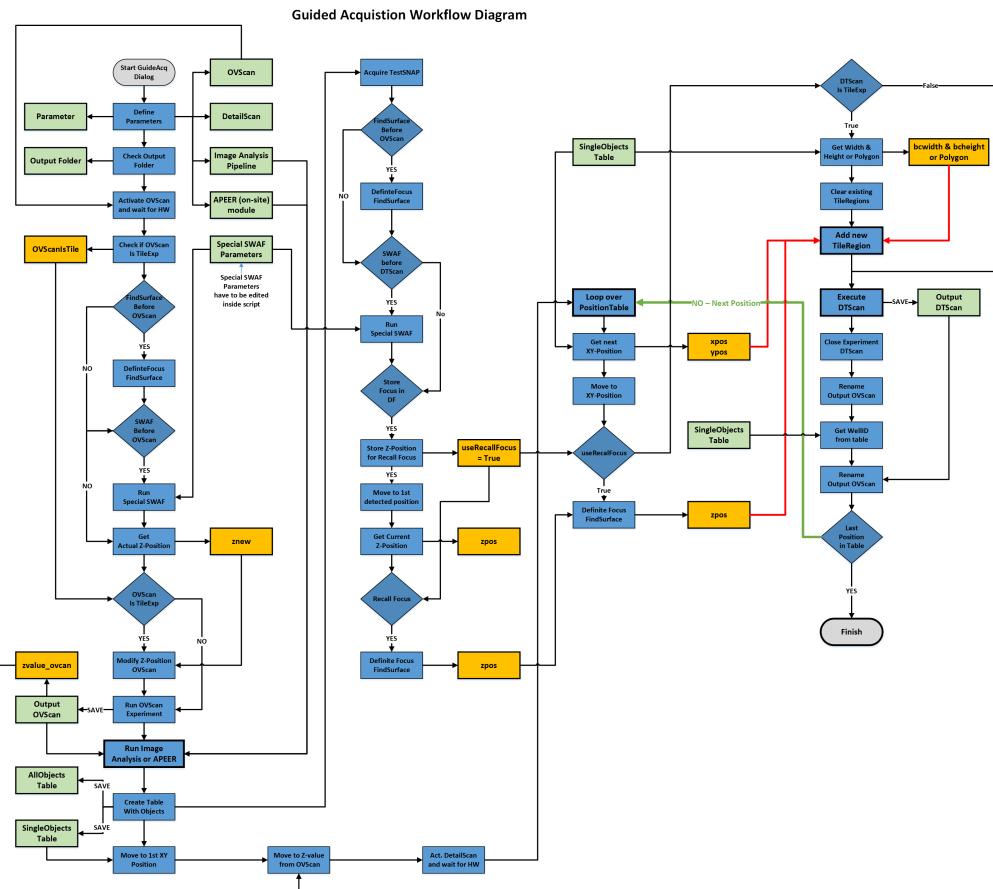


Figure 23: Complete workflow diagram for Guided Acquisition

## 8.2 Python Script: Guided\_Acquisition\_shortUI.py

This is the complete ZEN Blue python script. It can work on any motorized microscope stand using 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   : 8.3
4 # Author    : czsrh, czmla, czkel
5 # Date     : 21.07.2021
6 # Institution : Carl Zeiss Microscopy GmbH
7 #
8 # Optimized for the use with Celldiscoverer 7 and DF2, but
9 # applicable for all motorized stands running in ZEN Blue.
10 # Please adapt focussing commands, especially FindSurface
11 # when using with other stands.
12 #
13 # 1) - Select Overview Scan Experiment
14 # 2) - Select appropriate Image Analysis Pipeline or APEER module setting
15 #       to detect objects of interest
16 # 3) - Select Detailed Scan Experiment
17 # 4) - Specify the output folder for the image and data tables
18 #
19 # Tested with ZEN blue 3.4.
20 # Should work also with later version with some limitations
21 #
22 # Disclaimer: This tool is purely experimental. Feel free to
23 # use it at your own risk. Especially be aware of the fact
24 # that automated stage movements might damage hardware if
25 # one starts an experiment and the system is not setup
26 # and calibrated properly. Check everything in simulation mode first!
27 #
28 # Copyright(c) 2021 Carl Zeiss AG, Germany. All Rights Reserved.
29 #
30 # Permission is granted to use, modify and distribute this code,
31 # as long as this copyright notice remains part of the code.
32 #####
33
34 import time
35 from datetime import datetime
36 import errno
37 from System import Array
38 from System import ApplicationException
39 from System import TimeoutException
40 from System.IO import File, Directory, Path
41 from System.Collections.Generic import List
42 import sys
43
44 # version number for dialog window
45 version = 8.3
46 # file name for overview scan
47 ovscan_name = 'OverviewScan.czi'
48 """
49 Additional XY offset for possible 2nd port relative to the 1st port
50 Example: 1st port Camera and 2nd port LSM
51 Can be set to Zero, when system is correctly calibrated
52 !!! Only use when overview and detailed scan are using different detector !!!
53 """
54 dx_detector = 0.0
55 dy_detector = 0.0
56
57 # experiment blockindex
58 blockindex = 0
59 # delay for specific hardware movements in [seconds]
60 hwdelay = 1
61 # postprocessing switch
62 do_postprocess = False
63
64 #####
65
66
67 def find_module(name):
68     """Finds an APEER module given its Name.
69
70     :param name: The name of the module to look for
71     :type name: str
72 
```

```

73     :return: The first module with the specified name or None, if no such module exists.
74     :rtype: ApeerModule
75     """
76
77     modules = ZenApeer.Onsite.ListLocalModules()
78
79     return next((m for m in modules if m.ModuleName == name), None)
80
81
82 def get_module_inputs(params):
83     """Get the module inputs.
84
85     :param params: Apeer module parameters
86     :type params: ApeerModuleDescription
87     :return: List with input parameters
88     :rtype: list
89     """
90
91     module_inputs = []
92     for ip in params.Inputs:
93         module_inputs.append(ip.Key)
94
95     return module_inputs
96
97
98 def get_columns(table):
99     """Get columns names and their IDs as a dictionary.
100
101    :param table: Input table
102    :type table: ZenTable
103    :return: Dictionary with columns names and their respective IDs
104    :rtype: dict
105    """
106
107    col_dict = {}
108    colid = -1
109    # loop over all columns
110    for col in range(0, table.ColumnCount):
111        colid += 1
112        # get the caption and store in dictionary with ID as value
113        colcaption = table.Columns[col].Caption
114        col_dict[colcaption] = colid
115
116    return col_dict
117
118 def get_module(module_name, module_version=0):
119     """Get an APEER module and check if the desired version is available.
120
121     :param module_name: APEER module name
122     :type module_name: str
123     :param module_version: Version number of the APEER module, defaults to 0
124     :type module_version: int, optional
125     :return: mymodule, version_found
126     :rtype: ApeerModule, bool
127     """
128
129     version_found = False
130
131     # try to find the desired module
132     mymodule = find_module(module_name)
133
134     if mymodule is not None:
135         print 'Module : ' + module_name + ' found.'
136     elif mymodule is None:
137         print 'Module : ' + module_name + ' not found.'
138
139     if str(module_version) in mymodule.AvailableVersions:
140         print 'Module : ' + module_name + ' Version ' + str(module_version) + ' found.'
141         version_found = True
142
143     elif not str(module_version) in mymodule.AvailableVersions:
144         print 'Module : ' + module_name + ' Version ' + str(module_version) + ' not found.'
145
146     return mymodule, version_found
147
148
149 def getshortfiles(filelist):
150     """Create list with shortended filenames
151
152     :param filelist: List with files with complete path names
153     :type filelist: list
154     :return: List with filenames only
155     :rtype: list
156     """
157
158     files_short = []
159     for short in filelist:
160         files_short.append(Path.GetFileName(short))
161
162     return files_short
163
164
165 def checktableentry(datatable, entry2check='ImageSceneContainerName'):
166

```

```

162     """Check ZEN table for an existing column by name.
163
164     :param datatable: ZEE Table with data.
165     :type datatable: ZenTable
166     :param entry2check: Column to look for, defaults to 'ImageSceneContainerName'
167     :type entry2check: str, optional
168     :return: entry_exists, column
169     :rtype: bool, integer
170     """
171
172     num_col = datatable.ColumnCount
173     entry_exists = False
174     column = None
175
176     for c in range(0, num_col):
177         # get the current column name
178         colname = datatable.Columns[c].ColumnName
179         if colname == entry2check:
180             column = c
181             entry_exists = True
182             break
183
184     return entry_exists, column
185
186
187 def dircheck(basefolder):
188     """Check if a directory or folder already exists.
189     Create the folder if it does not exist
190
191     :param basefolder: folder to check
192     :type basefolder: str
193     :return: new_directory
194     :rtype: str
195     """
196
197     # check if the destination basefolder exists
198     base_exists = Directory.Exists(basefolder)
199
200     if base_exists:
201         print('Selected Directory Exists: ', base_exists)
202         # specify the desired output format for the folder, e.g. 2017-08-08_17-47-41
203         format = '%Y-%m-%d_%H-%M-%S'
204
205         # create the new directory
206         newdir = createfolder(basefolder, formatstring=format)
207         print('Created new directory: ', newdir)
208
209     if not base_exists:
210         Directory.CreateDirectory(basefolder)
211         newdir = basefolder
212
213     return newdir
214
215
216 def createfolder(basedir, formatstring='%Y-%m-%d_%H-%M-%S'):
217     """Creates a new folder inside an existing folder using a specific format
218
219     :param basedir: Folder inside which the new directory should be created
220     :type basedir: str
221     :param formatstring: String format for the new folder, defaults to '%Y-%m-%d_%H-%M-%S'
222     :type formatstring: str, optional
223     :return: Newly created directory
224     :rtype: str
225     """
226
227     # construct new directory name based on date and time
228     newdir = Path.Combine(basedir, datetime.now().strftime(formatstring))
229     # check if the new directory (for whatever reasons) already exists
230     try:
231         newdir_exists = Directory.Exists(newdir)
232         if not newdir_exists:
233             # create new directory if it does not exist
234             Directory.CreateDirectory(newdir)
235         if newdir_exists:
236             # raise error if it really already exists
237             raise SystemExit
238     except OSError as e:
239         if e.errno != errno.EEXIST:
240             newdir = None
241             raise # This was not a "directory exist" error..
242
243     return newdir
244
245
246 def cloneexp(expname, prefix='GA_', save=True, reloadexp=True):
247     """Clone an existing ZenExperiment.
248
249     :param expname: Name of the ZenExperiment
250     :type expname: ZenExperiment

```

```

251     :param prefix: Prefix for the experiment name, defaults to 'GA_'
252     :type prefix: str, optional
253     :param save: Option to save the cloned experiment, defaults to True
254     :type save: bool, optional
255     :param reloadexp: option to reload the cloned experiment afterwards, defaults to True
256     :type reloadexp: bool, optional
257     :return: exp_reload
258     :rtype: ZenExperiment or None
259     """
260
261     exp = Zen.Acquisition.Experiments.GetByName(expname)
262     exp_newname = prefix + expname
263
264     # save experiment
265     if save:
266         exp.SaveAs(exp_newname, False)
267         print('Saved Temporary Experiment as : ', exp_newname)
268         # close the original experiment object
269         exp.Close()
270         time.sleep(1)
271
272     # reload experiment
273     if reloadexp:
274         exp_reload = Zen.Acquisition.Experiments.GetByName(exp_newname)
275     elif not reloadexp:
276         exp_reload = None
277
278     return exp_reload
279
280
281 def runSWAF_special(SWAF_exp,
282                      delay=5,
283                      searchStrategy='Full',
284                      sampling=ZenSoftwareAutofocusSampling.Coarse,
285                      relativeRangeIsAutomatic=False,
286                      relativeRangeSize=500,
287                      timeout=0):
288     """Execute SWAF for a ZenExperiment with custom settings.
289
290     :param SWAF_exp: ZenExperiment containing the SWAF settings
291     :type SWAF_exp: ZenExperiment
292     :param delay: Hardware delay to wait until "everything" has moved into place, defaults to 5
293     :type delay: int, optional
294     :param searchStrategy: Search Strategy for the SWAF, defaults to 'Full'
295     :type searchStrategy: str, optional
296     :param sampling: Sampling for the SWAF, defaults to ZenSoftwareAutofocusSampling.Coarse
297     :type sampling: ZenSoftwareAutofocusSampling, optional
298     :param relativeRangeIsAutomatic: Set SWAF range automatically, defaults to False
299     :type relativeRangeIsAutomatic: bool, optional
300     :param relativeRangeSize: Relative range of SWAF [micron], defaults to 500
301     :type relativeRangeSize: int, optional
302     :param timeout: Timeout for SWAF [s], defaults to 0
303     :type timeout: int, optional
304     :return: zSWAF
305     :rtype: float
306     """
307
308     # get current z-Position
309     zSWAF = Zen.Devices.Focus.ActualPosition
310     print('Z-Position before special SWAF :', zSWAF)
311
312     # set DetailScan active and wait for moving hardware due to settings
313     SWAF_exp.SetActive()
314     time.sleep(delay)
315
316     # set SWAF parameters
317     SWAF_exp.SetAutofocusParameters(searchStrategy=searchStrategy,
318                                     sampling=sampling,
319                                     relativeRangeIsAutomatic=relativeRangeIsAutomatic,
320                                     relativeRangeSize=relativeRangeSize)
321     try:
322         print('Running special SWAF ...')
323         zSWAF = Zen.Acquisition.FindAutofocus(SWAF_exp, timeoutSeconds=timeout)
324     except ApplicationException as e:
325         print('Application Exception : ', e.Message)
326     except TimeoutException as e:
327         print(e.Message)
328
329     print('Z-Position after initial SWAF [micron]: ', zSWAF)
330
331     return zSWAF
332
333
334 def run_postprocessing(image, parameters={}, func='topography'):
335     """Example template to add post-processing functions.
336
337     :param image: Image to be processed
338     :type image: ZenImage
339     :param parameters: Dictionary with processing parameters for function, defaults to {}
340

```

```

340     :type parameters: dict, optional
341     :param func: Function name, defaults to 'topography'
342     :type func: str, optional
343     :return: image or processed image
344     :rtype: ZenImage
345     """
346
347     if func == 'topography':
348
349         noise_low = parameters['noise_low']
350         noise_high = parameters['noise_high']
351         outputfolder = parameters['outputfolder']
352         ext = parameters['extension']
353
354         # converting to topo, with defined FZ noisecut
355         # in filter settings: 0-255 means no filter, 1-254 means cut one gray scale from top and from bottom
356         imgtop = Zen.Processing.Transformation.Topography.CreateTopography(image, noise_low, noise_high)
357
358         # saving file to the directory
359         topo_filepath = Path.Combine(outputfolder, Path.GetFileNameWithoutExtension(image.FileName) + ext)
360         Zen.Processing.Utilities.ExportHeightmapFromTopography(imgtop, topo_filepath)
361         print('Exported to : ', topo_filepath)
362         imgtop.Close()
363
364     return image
365
366
367 def apply_recall_focus():
368     """Recall the stored Z-Value for Definite Focus 2 (DF2).
369
370     :return: [description]
371     :rtype: [type]
372     """
373
374     # get the z-position
375     zpos = Zen.Devices.Focus.ActualPosition
376
377     try:
378         # apply RecallFocus for the current position
379         Zen.Acquisition.RecallFocus()
380         zpos = Zen.Devices.Focus.ActualPosition
381         print('Recall Focus (Definite Focus) applied.')
382         print('Updated Z-Position: ', zpos)
383     except ApplicationException as e:
384         print('Application Exception : ', e.Message)
385         print('Recalling Focus (Definite Focus 2) failed.')
386
387     print('New Z-Position before Detail Experiment will start:', zpos)
388
389     return zpos
390
391
392 def modify_tileexperiment_rect(tileexp, blockindex=0,
393                                bcwidth=1.0,
394                                bcheight=1.0,
395                                xpos=1.0,
396                                ypos=1.0,
397                                zpos=1.0):
398     """Modify an existing ZenExperiment with an existing Tileregion by adding
399     an new rectangular Tileregion.
400
401     :param tileexp: ZenExperiment to be modified with a rectangular TileRegion.
402     :type tileexp: ZenExperiment
403     :param blockindex: Experiment Block index, defaults to 0
404     :type blockindex: int, optional
405     :param bcwidth: Width of the new TileRegion [micron], defaults to 1.0
406     :type bcwidth: float, optional
407     :param bcheight: Height of the new Tileregion [micron], defaults to 1.0
408     :type bcheight: float, optional
409     :param xpos: StageX of the new Tileregion center [micron], defaults to 1.0
410     :type xpos: float, optional
411     :param ypos: StageY of the new TileRegion center [micron], defaults to 1.0
412     :type ypos: float, optional
413     :param zpos: StageZ of the new TileRegion [micron], defaults to 1.0
414     :type zpos: float, optional
415     :return: tileexp
416     :rtype: ZenExperiment
417     """
418
419     print('Width and Height : ', '%.2f' % bcwidth, '%.2f' % bcheight)
420     print('Modifying Tile Properties XYZ Position and width and height.')
421
422     # Modify the XYZ position and size of the TileRegion on-the-fly
423     print('Starting Z-Position for current Object: ', '%.2f' % zpos)
424     print('New Tile Properties: ', '%.2f' % xpos, '%.2f' % ypos, '%.2f' % zpos, '%.2f' % bcwidth, '%.2f' % bcheight)
425     tileexp.ClearTileRegionsAndPositions(blockindex)
426     try:
427         tileexp.AddRectangleTileRegion(blockindex, xpos, ypos, bcwidth, bcheight, zpos)
428     except ApplicationException as e:

```

```

429         print('Application Exception : ', e.Message)
430
431     return tileexp
432
433
434     def modify_tileexperiment_polygon(tileexp, polyregion,
435                                         blockindex=0,
436                                         zpos=0.0):
437         """Modify an existing ZenExperiment with an existing Tileregion by adding
438         an new Polygon Tileregion.
439
440         :param tileexp: ZenExperiment to be modified with a Polygon TileRegion.
441         :type tileexp: ZenExperiment
442         :param polyregion: List with ZenPoints describing the polygon in absolute stageXY coordinates [micron]
443         :type bwidth: IList[ZenPoints]
444         :param blockindex: Experiment Block index, defaults to 0
445         :type blockindex: int, optional
446         :param zpos: StageZ of the new TileRegion [micron], defaults to 1.0
447         :type zpos: float, optional
448         :return: tileexp
449         :rtype: ZenExperiment
450
451     """
452
453     print('Modifying Tile Properties using Polygon for ZEN Image Analysis.')
454     print('Starting Z-Position for current Object: ', '%.2f' % zpos)
455     tileexp.ClearTileRegionsAndPositions(blockindex)
456     try:
457         tileexp.AddPolygonTileRegion(blockindex, polyregion, zpos)
458     except ApplicationException as e:
459         print('Application Exception : ', e.Message)
460
461     return tileexp
462
463
464     def create_exppolygon(points, stageTL, scaling):
465         """Convert List of pixel-based points from Image Analysis region into
466         List[ZenPoints] in absolute stageXY coordinates [micron].
467
468         :param points: List of Points [pixel]
469         :type points: List
470         :param stageTL: Image Stage Top-Left [micron]
471         :type stageTL:
472         :param scaling: [description]
473         :type scaling: Point
474         :return: zplist
475         :rtype: List[ZenPoints]
476
477
478         # create new list of ZenPoints
479         pl = []
480         zplist= List[ZenPoint]()
481
482         # iterate over all points inside list
483         for p in range(len(points)):
484             # use StageTopLeft and scaling to convert points to stageXY
485             newx = stageTL.X + round(points[p].X * scaling.X, 1)
486             newy = stageTL.Y + round(points[p].Y * scaling.Y, 1)
487             pl.append(ZenPoint(newx, newy))
488
489         # create IList[ZenPoint]
490         zplist.AddRange(pl)
491
492     return zplist
493
494
495     def getclassnames(ias):
496         """Get ID and name of classes from Image Analysis Setting
497
498         :param ias: Setting to extract the ID and names from
499         :type ias: ZenImageAnalysisSetting
500         :return: iacllasses
501         :rtype: dict
502
503
504         # create empty dictionary to conatin the ID and Name
505         iacllasses = {}
506         classnames = ias.GetRegionClassNames()
507
508         for id in range(0, len(classnames)):
509             try:
510                 # class for single objects
511                 cl = ias.GetRegionClass(id)
512             except:
513                 # class for all objects
514                 cl = ias.GetRegionsClass(id)
515
516             iacllasses[cl.ID] = cl.Name
517             print('ID - RegionClassName: ', cl.ID, cl.Name)

```

```

518     return iaclasses
519
520 #####
521 # clear console output
522 Zen.Application.MacroEditor.ClearMessages()
523
524 # check the location of experiment setups and image analysis settings are stored
525 docfolder = Zen.Application.Environment.GetFolderPath(ZenSpecialFolder.UserDocuments)
526 imgfolder = Zen.Application.Environment.GetFolderPath(ZenSpecialFolder.ImageAutoSave)
527 imgfolder = r'd:\Output\Guided_Acquisition'
528 format = '%Y-%m-%d_%H-%M-%S'
529
530 # get list with all existing experiments and image analysis setup and a short version of that list
531 expfiles = Directory.GetFiles(Path.Combine(docfolder, 'Experiment Setups'), '*.czexp')
532 ipfiles = Directory.GetFiles(Path.Combine(docfolder, 'Image Analysis Settings'), '*.czias')
533 apfiles = Directory.GetFiles(Path.Combine(docfolder, 'APEER Module Settings'), '*.czams')
534 expfiles_short = getshortfiles(expfiles)
535 ipfiles_short = getshortfiles(ipfiles)
536 apfiles_short = getshortfiles(apfiles)
537
538 # Initialize Dialog
539 GuidedAcqDialog = ZenWindow()
540 GuidedAcqDialog.Initialize("Guided Acquisition - Version : " + str(version))
541 # add components to dialog
542 GuidedAcqDialog.AddLabel('----- Select Overview Experiment -----')
543 GuidedAcqDialog.AddDropDown('overview_exp', 'Overview Scan Experiment', expfiles_short, 0)
544 GuidedAcqDialog.AddCheckbox('fs_before_overview', 'OPTION - Findsurface (DF only) before Overview', False)
545 GuidedAcqDialog.AddCheckbox('SWAF_before_overview', 'OPTION - SWAF before Overview', False)
546 GuidedAcqDialog.AddIntegerRange('SWAF_ov_initial_range', 'Initial SWAF Range before Overview [micron]', 200, 50, 3000)
547 GuidedAcqDialog.AddLabel('----- Select Image Analysis to detect objects -----')
548 GuidedAcqDialog.AddDropDown('ip_pipe', 'Image Analysis Pipeline', ipfiles_short, 0)
549 GuidedAcqDialog.AddCheckbox('use_poly', 'Use Polygons from ZEA IA instead of BBox', True)
550 GuidedAcqDialog.AddCheckbox('use_apeer_for_IA', 'Use APEER module run run IA instead', False)
551 GuidedAcqDialog.AddDropDown('ap_pipe', 'APEER Module Settings', apfiles_short, 0)
552 GuidedAcqDialog.AddLabel('----- Select DetailScan Experiment -----')
553 GuidedAcqDialog.AddDropDown('detailed_exp', 'Detailed Scan Experiment', expfiles_short, 1)
554 GuidedAcqDialog.AddCheckbox('fs_before_detail', 'OPTION - Findsurface (DF only) before Detail', False)
555 GuidedAcqDialog.AddCheckbox('SWAF_before_detail', 'OPTION - SWAF before Detail', False)
556 GuidedAcqDialog.AddIntegerRange('SWAF_detail_initial_range', 'Initial SWAF Range before Detail [micron]', 100, 10, 1000)
557 GuidedAcqDialog.AddCheckbox('recallfocus_beforeDT', 'OPTION - Use RecallFocus (DF only) before Detail', False)
558 GuidedAcqDialog.AddLabel('----- Specify basefolder to save the images -----')
559 GuidedAcqDialog.AddFolderBrowser('outfolder', 'Basefolder for Images and Data Tables', imgfolder)
560
561 # show the window
562 result = GuidedAcqDialog.Show()
563 if result.HasCanceled:
564     message = 'Macro was canceled by user.'
565     print(message)
566     raise SystemExit
567
568 # get the values and store them
569 OverViewExpName = str(result.GetValue('overview_exp'))
570 ImageAS = str(result.GetValue('ip_pipe'))
571 ApeerMS = str(result.GetValue('ap_pipe'))
572 DetailExpName = str(result.GetValue('detailed_exp'))
573 OutputFolder = str(result.GetValue('outfolder'))
574 fs_beforeOV = result.GetValue('fs_before_overview')
575 SWAF_beforeOV = result.GetValue('SWAF_before_overview')
576 SWAF_beforeOV_range = result.GetValue('SWAF_ov_initial_range')
577 fs_beforeDT = result.GetValue('fs_before_detail')
578 SWAF_beforeDT = result.GetValue('SWAF_before_detail')
579 SWAF_beforeDT_range = result.GetValue('SWAF_detail_initial_range')
580 RecallFocus = result.GetValue('recallfocus_beforeDT')
581 use_apeer = result.GetValue('use_apeer_for_IA')
582 use_polygon = result.GetValue('use_poly')
583
584
585 # print values
586 print('Overview Scan Experiment : ', OverViewExpName)
587 if not use_apeer:
588     print('Image Analysis Pipeline : ', ImageAS)
589     print('Use IA Polygon instead of BBox : ', use_polygon)
590 if use_apeer:
591     print('Use Apeer Module Setting : ', ApeerMS)
592     print('Detailed Scan Experiment : ', DetailExpName)
593     print('Output Folder for Data : ', OutputFolder)
594     print('\n')
595
596 # check directory
597 OutputFolder = dircheck(OutputFolder)
598
599 # create a duplicate of the OVScan experiment to work with
600 OVScan_reloaded = cloneexp(OverViewExpName)
601
602 # active the temporary experiment to trigger its validation
603 OVScan_reloaded.SetActive()
604 time.sleep(hwdelay)
605

```

```

607 # check if the experiment contains tile regions
608 OVScanIsTileExp = OVScan_reloaded.IsTilesExperiment(blockindex)
609
610 ###### START OVERVIEW SCAN EXPERIMENT #####
611
612 if fs_beforeOV:
613     # initial focussing via FindSurface to assure a good starting position
614     Zen.Acquisition.FindSurface()
615     print('Z-Position after FindSurface: ', Zen.Devices.Focus.ActualPosition)
616
617 if SWAF_beforeOV:
618     zSWAF = runSWAF_special(OVScan_reloaded,
619                             delay=hwdelay,
620                             searchStrategy='Full',
621                             sampling=ZenSoftwareAutofocusSampling.Coarse,
622                             relativeRangeIsAutomatic=False,
623                             relativeRangeSize=SWAF_beforeOV_range,
624                             timeout=1)
625
626 # get the resulting z-position
627 znew = Zen.Devices.Focus.ActualPosition
628
629 # adapt the Overview Scan Tile Experiment with new Z-Position
630 if OVScanIsTileExp:
631     OVScan_reloaded.ModifyTileRegionsZ(blockindex, znew)
632     print('Adapted Z-Position of Tile OverView. New Z = ', '%.2f' % znew)
633
634 # execute the experiment
635 print('\nRunning OverviewScan Experiment.\n')
636 output_OVScan = Zen.Acquisition.Execute(OVScan_reloaded)
637 # For testing purposes - Load overview scan image automatically instead of executing the "real" experiment
638 #ovtestimage = r'D:\GA_test_SRH\ov_ia.czii'
639 #output_OVScan = Zen.Application.LoadImage(ovtestimage, False)
640
641 # the stage top-left and the scaling of the overview image
642 stageTL = output_OVScan.GetPositionLeftTop()
643 ovscaling = output_OVScan.Scaling
644
645 # show the overview scan inside the document area
646 Zen.Application.Documents.Add(output_OVScan)
647 ovdoc = Zen.Application.Documents.GetByName(output_OVScan.Name)
648
649 # save the overview scan image inside the select folder
650 savepath_ovscan = Path.Combine(OutputFolder, ovscan_name)
651 output_OVScan.Save(savepath_ovscan)
652
653 # get the actual focus value for the overscan independent from the z-value
654 # before the start of the overview scan using the image metadata
655 #zvalue_ovscan = output_OVScan.Metadata.FocusPositionMicron
656
657 ##### END OVERVIEW SCAN EXPERIMENT #####
658
659 # run normal ZEN image analysis
660 if not use_appear:
661
662     # Load analysis setting created by the wizard or an separate macro
663     ias = ZenImageAnalysisSetting()
664
665     # for simulation use: 000 - RareEventExample.czias
666     ias.Load(ImageAS)
667
668     # Analyse the image
669     Zen.Analyzing.Analyze(output_OVScan, ias)
670
671     if use_polygon:
672
673         # get classes and derive regions names from that
674         iasclasses = getclassnames(ias)
675
676         # get the individual image analysis regions from 1st IA class (!!!)
677         # ATTENTION: the first single objects class will be used. It has ID = 2
678         regions = Zen.Analyzing.GetRegions(output_OVScan, iasclasses[2])
679         print('RegionClassNames: ', iasclasses)
680         print('Analysis found ' + str(regions.Count) + 'regions!')
681
682         # create dictionary for polygon regions
683         polyregions = {}
684
685         # loop over all regions and get the points of polygon (outline of the object)
686         for i in range(0, regions.Count):
687             points = regions[i].GetPolygon()
688
689             # convert points to stage coordinates
690             polyregions[i] = create_exppolygon(points, stageTL, ovscaling)
691
692         # Create Zen table with results for all detected objects (parent class)
693         AllObj = Zen.Analyzing.CreateRegionsTable(output_OVScan)
694
695

```

```

696     # Create Zen table with results for each single object
697     SingleObj = Zen.Analyzing.CreateRegionTable(output_OVScan)
698
699     # check for existence of required column names inside table
700     soi = SingleObj.GetBoundsColumnInfoFromImageAnalysis(True)
701
702     # 1st item is a bool indicating if all required columns could be found
703     columnsOK = soi.AreRequiredColumnsAvailable
704
705     if not columnsOK:
706         print('Execution stopped. Required Columns are missing.')
707         raise Exception('Execution stopped. Required Columns are missing.')
708
709     # show and save data tables to the specified folder
710     Zen.Application.Documents.Add(AllObj)
711     Zen.Application.Documents.Add(SingleObj)
712     AllObj.Save(Path.Combine(OutputFolder, 'OverviewTable.csv'))
713     SingleObj.Save(Path.Combine(OutputFolder, 'SingleObjectsTable.csv'))
714
715 # use APEER module to detect the objects
716 if use_apeer:
717
718     # read it from settings file
719     ams = ZenApeer.Onsite.ModuleSetting()
720
721     # loaf the APEER module setting b - remove *.czams file extension first
722     ams.Load(Path.GetFileNameWithoutExtension(ApeerMS))
723
724     print('----- Apeer Module Setting -----')
725     print('Module Name : ', ams.ModuleName)
726     print('Module Version : ', ams.ModuleVersion)
727     print('Module Parameters : ', ams.Parameters)
728
729     # get module and check
730     mymodule, version_found = get_module(ams.ModuleName, module_version=ams.ModuleVersion)
731
732     # exit if the check failed
733     if mymodule is None or not version_found:
734         print('Module check failed. Exiting.')
735         raise SystemExit
736
737     # get the module parameters for the specified module
738     module_params = ZenApeer.Onsite.GetSampleModuleParameters(ams.ModuleName, ams.ModuleVersion)
739
740     # get the module input programmatically
741     module_inputs = get_module_inputs(module_params)
742
743     # create the required dictionary with the correct key and value
744     input_image = {module_inputs[0]: savepath_ovscan}
745
746     # create the path to save the results
747     #savepath_apeer = Path.Combine(OutputFolder, 'apeer_results')
748     savepath_apeer = OutputFolder
749
750     # create the output directory if not existing already
751     if not Directory.Exists(savepath_apeer):
752         Directory.CreateDirectory(savepath_apeer)
753
754     # run the local APEER module with using keywords
755     try:
756         runoutputs, status, log = ZenApeer.Onsite.RunModule(moduleName=ams.ModuleName,
757                                                       moduleVersion=ams.ModuleVersion,
758                                                       inputs=input_image,
759                                                       parameters=ams.Parameters,
760                                                       storagePath=savepath_apeer)
761
762         for op in runoutputs.GetEnumerator():
763             print('----- Outputs -----')
764             print(op.Key, ' : ', op.Value)
765
766     except ApplicationException as e:
767         print('Module Run failed.', e.Message)
768         raise SystemExit
769
770     # get results storage locations
771     # IMPORTANT: To be used inside Guided Acquisition
772     # the APEER Module is expected to have
773     # at least those to outputs with exactly those names
774
775     if not runoutputs.ContainsKey('segmented_image'):
776         print('No output : segmented_image')
777         raise SystemExit
778
779     if not runoutputs.ContainsKey('objects_table'):
780         print('No output : objects_table')
781         raise SystemExit
782
783     # load the segmented image and make sure the pyramid is calculated
784     segmented_image = Zen.Application.LoadImage(runoutputs['segmented_image'], False)

```

```

785     Zen.Processing.Utilities.GenerateImagePyramid(segmented_image, ZenBackgroundMode.Black)
786     Zen.Application.Documents.Add(segmented_image)
787
788     # auto-display min-max
789     ids = segmented_image.DisplaySetting.GetAllChannelIds()
790     for id in ids:
791         segmented_image.DisplaySetting.SetParameter(id, 'IsAutoApplyEnabled', True)
792
793     # initialize ZenTable object and load CSV file
794     SingleObj = ZenTable()
795     SingleObj.Load(runoutputs['objects_table'])
796     Zen.Application.Documents.Add(SingleObj)
797
798     # get all columns as dict with columnIDs
799     colID = get_columns(SingleObj)
800
801     # define the required column names here
802     col2check = ('bbox_center_stageX', 'bbox_center_stageY', 'bbox_width_scaled', 'bbox_height_scaled')
803
804     # check if all columns exist
805     if all(key in colID for key in col2check):
806         print('All required columns found.')
807     else:
808         print('Not All required columns found. Exiting.')
809         raise SystemExit
810
811     # check the number of detected objects = rows inside image analysis table
812     num_POI = SingleObj.RowCount
813
814 ##### Prepare DetailScan #####
815
816 print('Starting DetailScan ...')
817
818 if not use_apeer:
819     xpos_1st = SingleObj.GetValue(0, soi.CenterXColumnIndex)
820     ypos_1st = SingleObj.GetValue(0, soi.CenterYColumnIndex)
821
822 if use_apeer:
823     xpos_1st = SingleObj.GetValue(0, colID['bbox_center_stageX'])
824     ypos_1st = SingleObj.GetValue(0, colID['bbox_center_stageY'])
825
826 # move to 1st detected object to be at a XY-position that makes sense
827 Zen.Devices.Stage.MoveTo(xpos_1st + dx_detector, ypos_1st + dy_detector)
828
829 # and move the the z-values from the overview scan image
830 Zen.Devices.Focus.MoveTo(zvalue_ovscan)
831
832 # create an duplicate of the DetailScan experiment to work with
833 DetailScan_reloaded = cloneexp(DetailExpName)
834
835 # active the temporary experiment to trigger its validation
836 DetailScan_reloaded.SetActive()
837 time.sleep(hwdelay)
838 # check if the experiment contains tile regions
839 DetailIsTileExp = DetailScan_reloaded.IsTilesExperiment(blockindex)
840
841 # test snap to change to the valid settings, e.g. the objective from the DetailScan
842 testsnap = Zen.Acquisition.AcquireImage(DetailScan_reloaded)
843 print('Acquire Test Snap using setting from DetailScan')
844 testsnap.Close()
845 # wait for moving hardware due to settings
846 time.sleep(hwdelay)
847
848 if fs_beforeDT:
849     try:
850         # initial focussing via FindSurface to assure a good starting position
851         Zen.Acquisition.FindSurface()
852         print('Z-Position after FindSurface: ', Zen.Devices.Focus.ActualPosition)
853     except ApplicationException as e:
854         print('Application Exception : ', e.Message)
855         print('FindSurface (Definite Focus) failed.')
856
857 if SWAF_beforeDT:
858     zSWAF = runSWAF_special(DetailScan_reloaded,
859                             delay=hwdelay,
860                             searchStrategy='Full',
861                             sampling=ZenSoftwareAutofocusSampling.Coarse,
862                             relativeRangeIsAutomatic=False,
863                             relativeRangeSize=SWAF_beforeDT_range,
864                             timeout=0)
865
866 userecallfocus = False
867
868 if RecallFocus:
869     try:
870         # store current focus position inside DF to use it with RecallFocus
871         Zen.Acquisition.StoreFocus()
872         userecallfocus = True
873     except ApplicationException as e:

```

```

874     print('Application Exception : ', e.Message)
875     print('StoreFocus (Definite Focus) failed.')
876     userrecallfocus = False
877
878 ##### START DETAILED SCAN EXPERIMENT #####
879
880 # again get the resulting focus position
881 zpos = Zen.Devices.Focus.ActualPosition
882
883 if not use_apeer:
884
885     # check for the column 'ID' which is required
886     ID_exists, column_ID = checktableentry(SingleObj, entry2check='ID')
887
888     # execute detailed experiment at the position of every detected object
889     for obj in range(SingleObj.RowCount):
890
891         # get the object information from the position table
892         POI_ID = SingleObj.GetValue(obj, column_ID)
893
894         # get XY-stage position from table
895         xpos = SingleObj.GetValue(obj, soi.CenterXColumnIndex)
896         ypos = SingleObj.GetValue(obj, soi.CenterYColumnIndex)
897
898         # move to the current position
899         Zen.Devices.Stage.MoveTo(xpos + dx_detector, ypos + dy_detector)
900         print('Moving Stage to Object ID: ', POI_ID, ' at : ', '%.2f' % xpos, '%.2f' % ypos)
901
902         # try to apply RecallFocus (DF only) when this option is used
903         if userrecallfocus:
904             zpos = apply_recall_focus()
905
906         # if DetailScan is a Tile Experiment
907         if DetailIsTileExp:
908
909             print('Detailed Experiment contains TileRegions.')
910             # Modify tile center position - get bounding rectangle width and height in microns
911             bcwidth = SingleObj.GetValue(obj, soi.WidthColumnIndex)
912             bcheight = SingleObj.GetValue(obj, soi.HeightColumnIndex)
913
914             if use_polygon:
915                 # modify the experiment with the respective polygon region
916                 DetailScan_reloaded = modify_tileexperiment_polygon(DetailScan_reloaded, polyregions[obj],
917                                                               blockindex=0,
918                                                               zpos=zpos)
919             else:
920                 # modify the experiment
921                 DetailScan_reloaded = modify_tileexperiment_rect(DetailScan_reloaded,
922                                                               blockindex=blockindex,
923                                                               bcwidth=bcwidth,
924                                                               bcheight=bcheight,
925                                                               xpos=xpos,
926                                                               ypos=ypos,
927                                                               zpos=zpos)
928
929         if not DetailIsTileExp:
930             print('Detailed Experiment does not contain TileRegions. Nothing to modify.')
931
932         # execute the DetailScan experiment
933         print('Running Detail Scan Experiment at new XYZ position.')
934         try:
935             output_detailscan = Zen.Acquisition.Execute(DetailScan_reloaded)
936         except ApplicationException as e:
937             print('Application Exception : ', e.Message)
938
939         # get the image data name
940         dtscan_name = output_detailscan.Name
941
942         """
943         Modification for multiscene images: container name needs to be part
944         of the filename of the detailed Scan.
945         First check if Column "Image Scene Container Name" or "WellId" is available
946         and then add Container Name to filename.
947         """
948
949         wellid_exist, column_wellid = checktableentry(SingleObj, entry2check='ImageSceneContainerName')
950
951         # in case Image Scene Container Name is not defined
952         if not wellid_exist:
953             message = 'Missing column ImageSceneContainerName in Image Analysis Results.\nPlease Select Features inside the'
954             ' Image Analysis when needed.'
955             print(message)
956             container_name = 'empty'
957
958         # save the image data to the selected folder and close the image
959         output_detailscan.Save(Path.Combine(OutputFolder, output_detailscan.Name))
960         output_detailscan.Close()
961

```

```

962     # rename the CZI regarding to the object ID - Attention - IDs start with 2 !!!
963     if not wellid_exist:
964         # no wellID found inside table
965         newname_dtscan = 'DTScan_ID_' + str(POI_ID) + '.czi'
966     if wellid_exist:
967         # wellID was found inside table
968         well_id = SingleObj.GetValue(obj, column_wellid)
969         newname_dtscan = 'DTScan_Well_' + str(well_id) + '_ID_' + str(POI_ID) + '.czi'
970
971     print('Renaming File: ' + dtscan_name + ' to: ' + newname_dtscan + '\n')
972     File.Move(Path.Combine(OutputFolder, dtscan_name), Path.Combine(OutputFolder, newname_dtscan))
973
974 ##### OPTIONAL POSTPROCESSING #####
975
976 if do_postprocess:
977
978     # do the postprocessing
979     image2process = Zen.Application.LoadImage(newname_dtscan, False)
980
981     # define the parameters for processing: Topography Export
982     parameters = {}
983     parameters['noise_low'] = 1
984     parameters['noise_high'] = 254
985     parameters['outputfolder'] = OutputFolder
986     parameters['extension'] = '.sur'
987
988     # run the processing and export and close the image
989     image2process = run_postprocessing(image2process, parameters=parameters, func='topography')
990     image2process.Close()
991
992 if use_apeer:
993
994     # execute detailed experiment at the position of every detected object
995     for obj in range(SingleObj.RowCount):
996
997         # get the object information from the position table
998         POI_ID = obj
999
1000        # get XY-stage position from table
1001        xpos = SingleObj.GetValue(obj, colID['bbox_center_stageX'])
1002        ypos = SingleObj.GetValue(obj, colID['bbox_center_stageY'])
1003
1004        # move to the current position
1005        Zen.Devices.Stage.MoveTo(xpos + dx_detector, ypos + dy_detector)
1006        print('Moving Stage to Object:', POI_ID + 1, ' at : ', '%.2f' % xpos, '%.2f' % ypos)
1007
1008        # try to apply RecallFocus (DF only) when this option is used
1009        if userecallfocus:
1010            zpos = apply_recall_focus()
1011
1012        # if DetailScan is a Tile Experiment
1013        if DetailIsTileExp:
1014
1015
1016            # Modify tile center position - get bounding rectangle width and height [microns]
1017            print('Detailed Experiment contains TileRegions.')
1018            bewidth = SingleObj.GetValue(obj, colID['bbox_width_scaled'])
1019            bcheight = SingleObj.GetValue(obj, colID['bbox_height_scaled'])
1020
1021            # modify the ZenExperiment for the DetailScan
1022            DetailScan_reloaded = modify_tileexperiment_rect(DetailScan_reloaded,
1023                                                    blockindex=blockindex,
1024                                                    bewidth=bewidth,
1025                                                    bcheight=bcheight,
1026                                                    xpos=xpos,
1027                                                    ypos=ypos,
1028                                                    zpos=zpos)
1029
1030        if not DetailIsTileExp:
1031            print('Detailed Experiment does not contain TileRegions. Nothing to modify.')
1032
1033        # execute the DetailScan experiment
1034        print('Running Detail Scan Experiment at new XYZ position.')
1035        try:
1036            output_detailscan = Zen.Acquisition.Execute(DetailScan_reloaded)
1037        except ApplicationException as e:
1038            print('Application Exception : ', e.Message)
1039
1040        # get the image data name
1041        dtscan_name = output_detailscan.Name
1042
1043        """
1044        Modification for multiscene images: container name needs to be part
1045        of the filename of the detailed Scan.
1046        First check if Column "Image Scene Container Name" or "WellId" is available
1047        and then add Container Name to filename.
1048        """
1049
1050        if 'WellId' in colID:

```

```

1051     print('WellId column found.')
1052     wellid_exist = True
1053     column_wellid = colID['WellId']
1054 else:
1055     print('WellId column not found.')
1056     wellid_exist = False
1057     column_wellid = None
1058
1059 # in case Image Scene Container Name is not defined
1060 if not wellid_exist:
1061     message = 'Missing column WellId in DataTable.\nPlease modify your image analysis inside the APEER module or
1062     => manually add column.'
1063     print(message)
1064     container_name = 'empty'
1065
1066 # save the image data to the selected folder and close the image
1067 output_detailscan.Save(Path.Combine(OutputFolder, output_detailscan.Name))
1068 output_detailscan.Close()
1069
1070 # rename the CZI regarding to the object ID - Attention - IDs start with 2 !!!
1071 if not wellid_exist:
1072     # no wellID found inside table
1073     newname_dtscan = 'DTScan_ID_' + str(POI_ID) + '.czi'
1074 if wellid_exist:
1075     # wellID was found inside table
1076     well_id = SingleObj.GetValue(obj, column_wellid)
1077     newname_dtscan = 'DTScan_Well_' + str(well_id) + '_ID_' + str(POI_ID) + '.czi'
1078
1079 print('Renaming File: ' + dtscan_name + ' to: ' + newname_dtscan + '\n')
1080 File.Move(Path.Combine(OutputFolder, dtscan_name), Path.Combine(OutputFolder, newname_dtscan))
1081 ##### OPTIONAL POSTPROCESSING #####
1082
1083 if do_postprocess:
1084
1085     # do the postprocessing
1086     image2process = Zen.Application.LoadImage(newname_dtscan, False)
1087
1088     # define the parameters for processing: Topography Export
1089     parameters = {}
1090     parameters['noise_low'] = 1
1091     parameters['noise_high'] = 254
1092     parameters['outputfolder'] = OutputFolder
1093     parameters['extension'] = '.sur'
1094
1095     # run the processing and export and close the image
1096     image2process = run_postprocessing(image2process, parameters=parameters, func='topography')
1097     image2process.Close()
1098
1099 ###### END DETAILED SCAN EXPERIMENT #####
1100
1101 # restore the original OVScan experiment
1102 OVScan_orig = Zen.Acquisition.Experiments.GetByName(OverViewExpName)
1103 OVScan_orig.SetActive()
1104
1105 # delete the temporary experiments when all loops are finished
1106 Zen.Acquisition.Experiments.Delete(OVScan_reloaded)
1107 Zen.Acquisition.Experiments.Delete(DetailScan_reloaded)
1108
1109 # show the overview scan document again at the end
1110 Zen.Application.Documents.ActiveDocument = ovdoc
1111
1112 print('All Positions done. Guided Acquisition Workflow finished.')

```

## 9 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.

## 10 Disclaimer

### DISCLAIMER:

The script and application are free and can be used by everybody on their 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.