Contents

1 Introduction **2**
1.1 Basic Workflow Definition . . . 2
1.2 General Workflow Definition . . . 3
1.3 General Workflow - Complete Overview . . . 5

2 Workflow - Single Steps **6**
2.1 Acquire Sample Data with Overview Scan . . . 6
2.2 Create Image Analysis Pipeline . . . 7
2.3 Define Overview Experiment . . . 9
2.4 Define Detailed Scan Experiment . . . 10

3 Automation via OAD **11**
3.1 Pure Scripting or User Dialog . . . 11
3.2 User Dialog Requirements . . . 11
3.3 Prerequisites . . . 12
3.4 Create the User Dialog . . . 13
3.5 Overview Scan Experiment . . . 15
3.6 Find the interesting Objects . . . 16
3.7 Modify the Tile Dimensions . . . 18
3.8 Run the Detailed Scan . . . 19

4 Testrun with FluoCells Slide **21**

5 Test Run with Brain Slide **23**

6 Appendix: Python Scripts **25**
6.1 Guided_Acquisition.czmac . . . 25
6.2 Custom DLLs . . . 30

7 ToDo's and Limitations **30**

8 Disclaimer **30**

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 (or a long period of time).
- 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.

- An object that meets specific criteria regarding its parameters (size, brightness, shape, intensity, ...)
- A specific change of a parameter during an experiment (e.g. a cell gets really "bright" upon ...)

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.

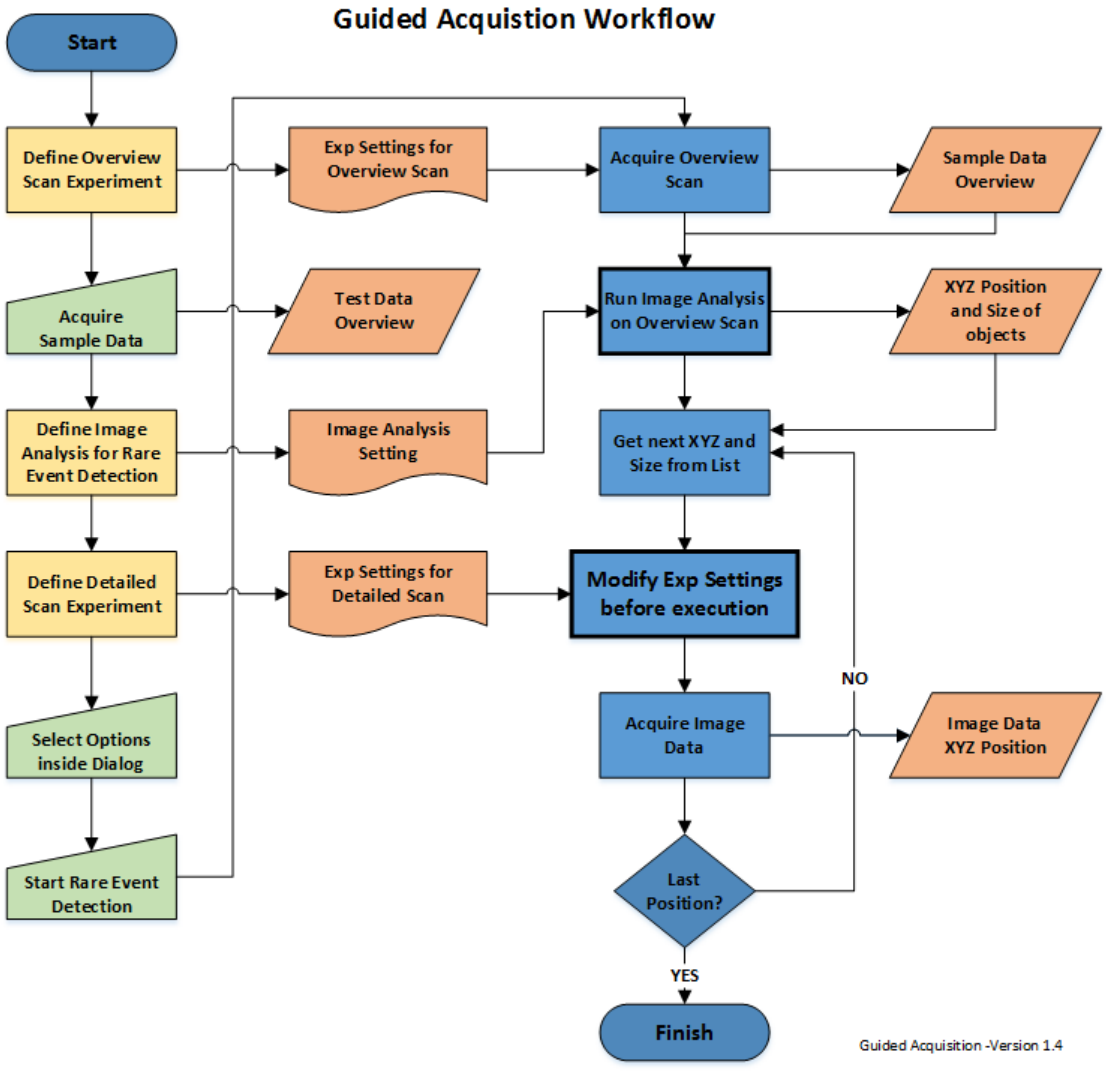

Figure 1: General workflow for Guided Acquisition.

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

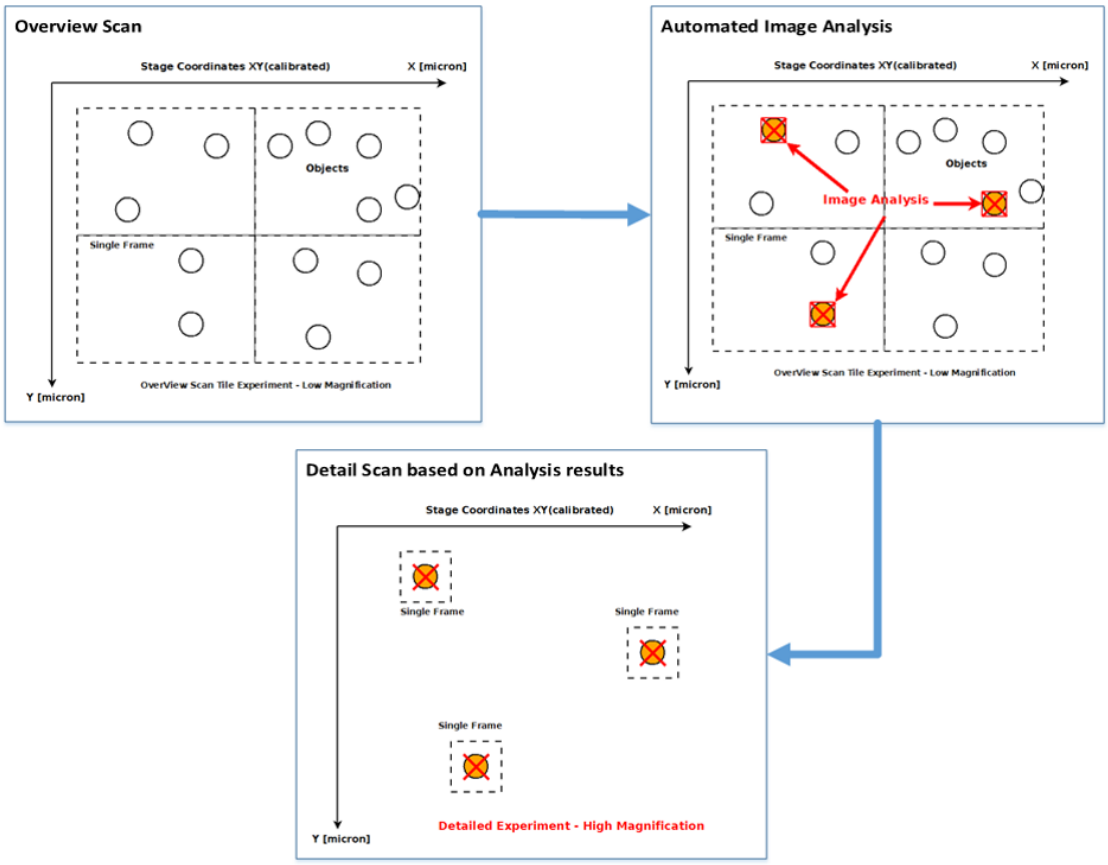

Figure 2: Larger area with some candidates for a "interesting" objects.

1.3 General Workflow - Complete Overview

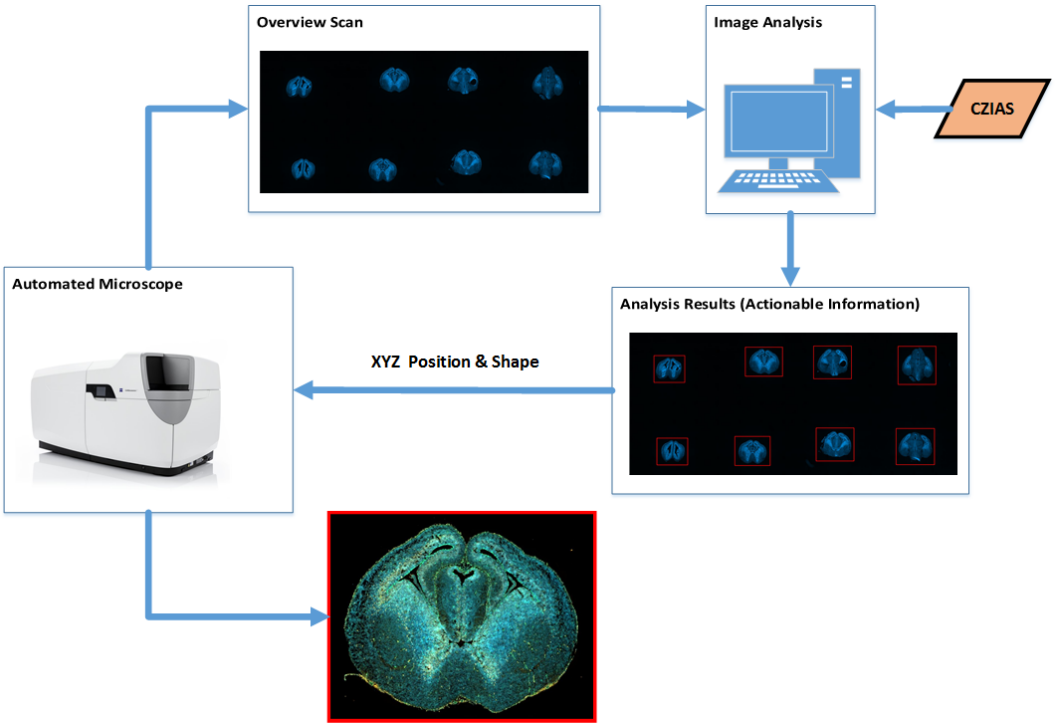

Figure 3: Workflow - Actionable Information 1.

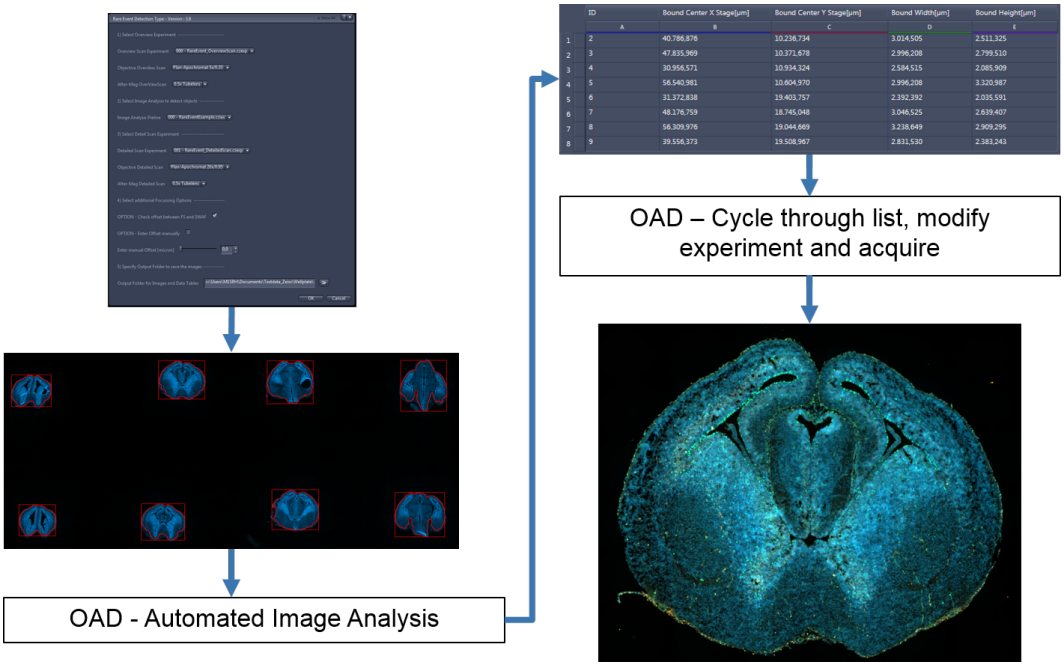

Figure 4: Workflow - Actionable Information 2

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).

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.

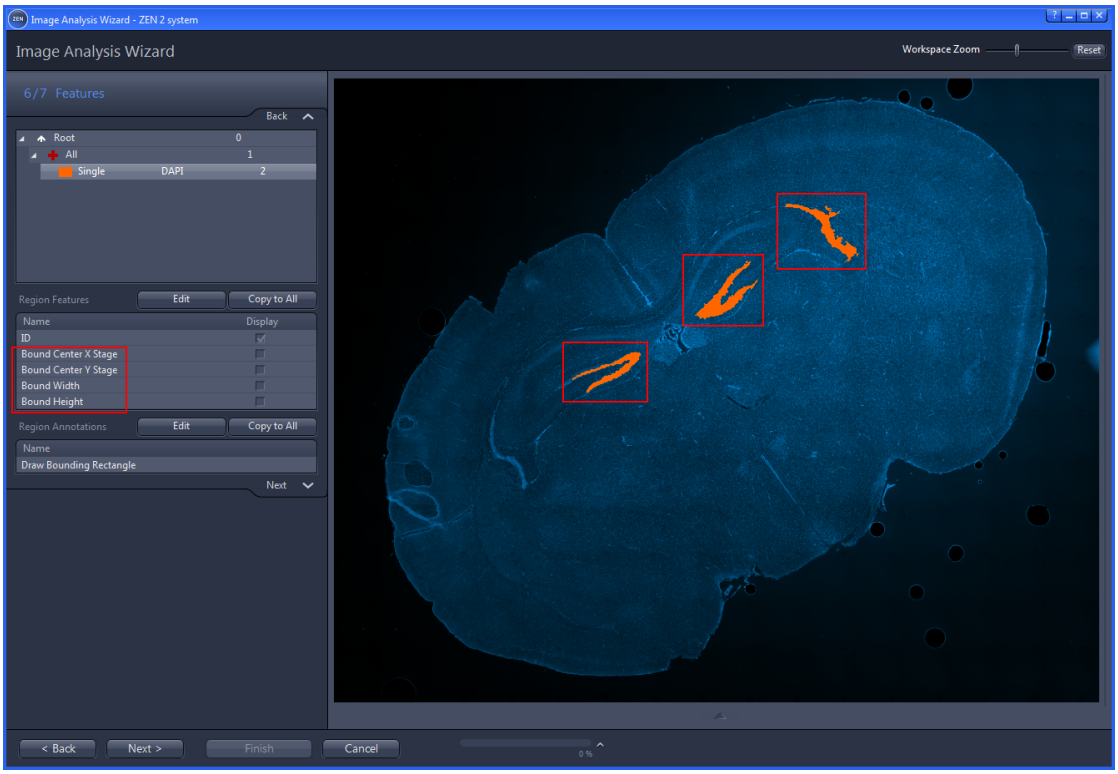

Figure 5: Select the required positional features inside the wizard.

Currently one needs to select the following parameters:

- Object ID
- Bound Center Stage X and Y
- Bound Width and Height

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.

Depending on your application, it might be useful to measure additional parameters. Feel free to add whatever might be useful.

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:

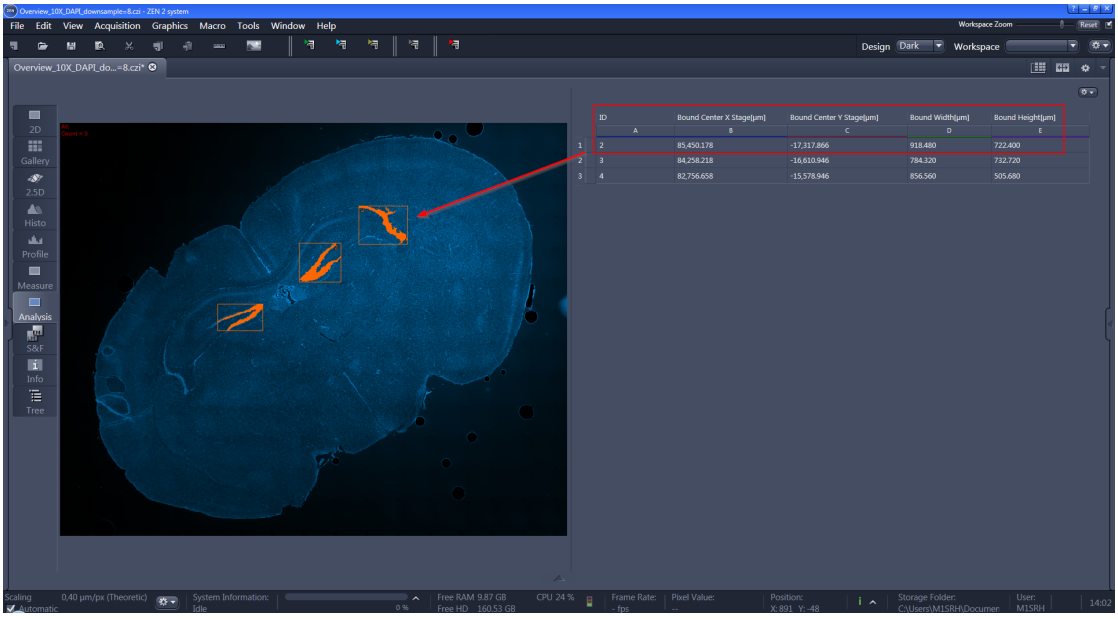

Figure 6: Result of the image analysis on the overview scan image.

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 automated almost any kind of workflow at a given position.

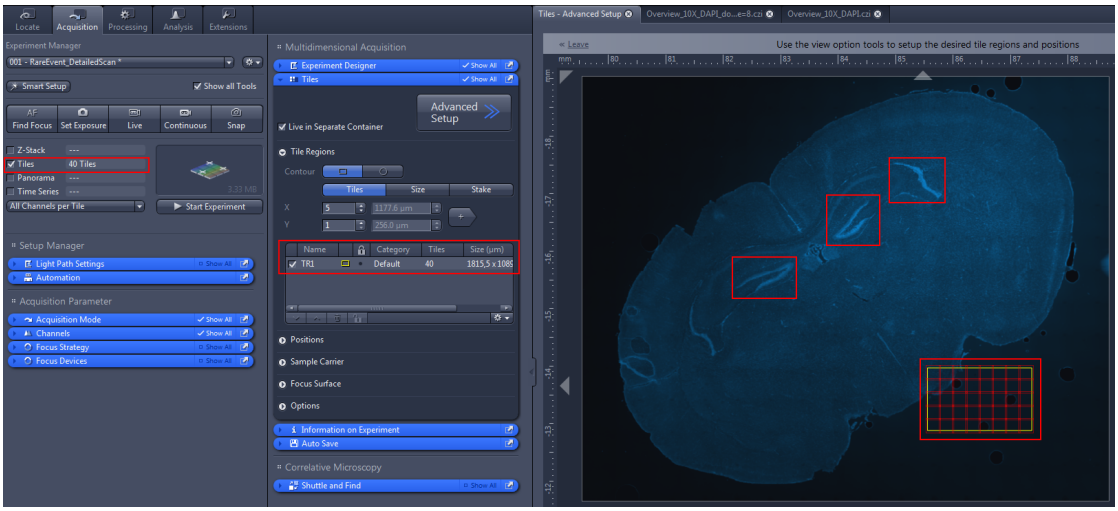

Figure 7: Setup of a simple detailed scan experiment.

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 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.

3.2 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.**

3.3 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 a python function is defined here for later usage. It will check, if a chosen directory is really empty before the actual workflow starts.

```
21  # import custom DLLs required for Rare Event Detection
22  import clr
23  clr.AddReference('RETools.dll')
24  clr.AddReference('TileTools.dll')
25  import RETools
26  import Tiles
27
28  from System.IO import File, Directory, Path
29  import sys
30
31  # optional debugging output
32  verbose = True
33  # version number for dialog window
34  version = 4.1
35  # file name for overview scan
36  ovscan_name = 'OverviewScan.czi'
37  # additional XY offest for possible LSM port relative to the Camera port (zero)
38  dx_LSM = 0.0
39  dy_LSM = 0.0
40
41  def dircheck(folder2check, createdir=False):
42
43      # check if the destination folder is empty
44      direxists = Directory.Exists(folder2check)
45      if direxists:
46          isEmpty = Directory.GetFiles(folder2check).Length
47          # check if the directory is empty in case it already exits
48          if isEmpty != 0:
49              print 'Directory already contains files! Confirm Selection or Create new Directory.'
50              SelectFolderDialog = ZenWindow()
51              SelectFolderDialog.Initialize('New Folder Selection? Directory Contains Files!',650,100,True,True)
52              # add components to dialog
53              SelectFolderDialog.AddFolderBrowser('fd', 'Confirm Folder or Select new Directory', folder2check, '0','0')
54              sf = SelectFolderDialog.Show()
55              if sf.HasCanceled:
56                  message = 'Macro was canceled by user.'
57                  print message
58                  raise SystemExit
59
60              folder2check = str(sf.GetValue('fd'))
61              print 'Creating Output Folder: ', folder2check
62
63      # create directory when option is set
64      if createdir:
65          print 'Creating Output Folder: ', folder2check
66          Directory.CreateDirectory(folder2check)
67
68      return folder2check
69
70  # clear console output
71  Zen.Application.MacroEditor.ClearMessages()
72
73  # check the location of folder where experiment setups and image analysis settings are stored
74  docfolder =  Zen.Application.Environment.GetFolderPath(ZenSpecialFolder.UserDocuments)
75  #imgfolder =  Zen.Application.Environment.GetFolderPath(ZenSpecialFolder.ImageAutoSave)
76  imgfolder = r'D:\RareEvent_FL-Slide1\Output_RareEvent'
77
78  # get list with all existing experiments and image analysis setup
79  expfiles = RETools.RareEventTools.GetExperimentSetups(docfolder)
80  ipfiles = RETools.RareEventTools.GetImageAnalysisSetups(docfolder)
81
82  # collect information about the configured objectives and optovars
83  objectives = RETools.RareEventTools.CreateObjectiveInfo()
84  optovars = RETools.RareEventTools.CreateOptovarInfo()
85
86  # Activate Zen.Application
87  RareEventDialog = ZenWindow()
88  RareEventDialog.Initialize('Rare Event Detection - Version : ' + str(version),650,750,True,True)
```

3.4 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.

```
90  # collect information about the configured objectives and optovars
91  objectives = RETools.RareEventTools.CreateObjectiveInfo()
92  optovars = RETools.RareEventTools.CreateOptovarInfo()
93  if verbose:
94      print('Objectives:')
95      print('Number: ', objectives.Count)
96      print('Names: ', objectives.Names)
97      print('Positions: ', objectives.Positions)
98      print('-----------------------------------------')
99      print('Optovars:')
100     print('Number: ', optovars.Count)
101     print('Names: ', optovars.Names)
102     print('Positions: ', optovars.Positions)
103     print('\n')
104
105 # Activate Zen.Application
106 RareEventDialog = ZenWindow()
107 RareEventDialog.Initialize('Guided Acquisition - Version : ' + str(version), 650, 750, True, True)
108 # add components to dialog
109 RareEventDialog.AddLabel('1) Select Overview Experiment  ------------------------------')
110 RareEventDialog.AddDropDown('overview_exp', 'Overview Scan Experiment', expfiles, 0)
111 RareEventDialog.AddDropDown('objectiveOV', 'Objective OvervIew Scan', objectives.Names, 1)
112 RareEventDialog.AddDropDown('optovarOV', 'After-Mag OverViewScan', optovars.Names, 2)
113 RareEventDialog.AddCheckbox('SWAF_before_overview', 'OPTION - (Find Surface) & SWAF before Overview', True)
114 RareEventDialog.AddLabel('2) Select Image Analysis to detect objects  -----------------')
115 RareEventDialog.AddDropDown('ip_pipe', 'Image Analysis Pieline', ipfiles, 0)
116 RareEventDialog.AddLabel('3) Select Detail Scan Experiment  --------------------------')
117 RareEventDialog.AddDropDown('detailed_exp', 'Detailed Scan Experiment', expfiles, 1)
118 RareEventDialog.AddDropDown('objectiveDT', 'Objective Detailed Scan', objectives.Names, 2)
119 RareEventDialog.AddDropDown('optovarDT', 'After-Mag Detailed Scan', optovars.Names, 2)
120 RareEventDialog.AddCheckbox('checkoffset', 'OPTION - Check offset between FS and SWAF', False)
121 RareEventDialog.AddCheckbox('manualoffset', 'OPTION - Enter Offset manually', False)
122 RareEventDialog.AddDoubleRange('offsetvalue', 'Enter manual Offset [micron]', 0, 0, 100)
123 RareEventDialog.AddLabel('4) Specify Output Folder to save the images -----------------')
124 RareEventDialog.AddFolderBrowser('outfolder','Output Folder for Images and Data Tables', imgfolder)
125
126 # show the window
127 result = RareEventDialog.Show()
128 if result.HasCanceled:
129     message = 'Macro was canceled by user.'
130     print message
131     raise SystemExit
132
133 # get the values and store them
134 OverViewExpName = str(result.GetValue('overview_exp'))
135 ImageAS = str(result.GetValue('ip_pipe'))
136 DetailExpName = str(result.GetValue('detailed_exp'))
137 OutputFolder = str(result.GetValue('outfolder'))
138 SWAF_beforeOV = result.GetValue('SWAF_before_overview')
139 CheckOffset = result.GetValue('checkoffset')
140 ManualOffset = result.GetValue('manualoffset')
141 OffsetValue = result.GetValue('offsetvalue')
142 objOV = result.GetValue('objectiveOV')
143 optoOV = result.GetValue('optovarOV')
144 objDT = result.GetValue('objectiveDT')
145 optoDT = result.GetValue('optovarDT')
146
147 # print values - this is optional
148 print 'Overview Scan Experiment : ' + OverViewExpName
149 print 'Combination OverView Scan: ', objOV + ' + ' + optoOV
150 print 'Image Analysis Pipeline : ' + ImageAS
151 print 'Detailed Scan Experiment : ' + DetailExpName
152 print 'Combination Detailed Scan: ', objDT + ' + ' + optoDT
153 print 'Output Folder for Data : ' + OutputFolder, '\n'
154
155 # check directory
156 OutputFolder = dircheck(OutputFolder)
```

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.

3.5 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.

```
175 ############ START OVERVIEW SCAN EXPERIMENT #################
176
177 # use the correct objective and optovar for the overview scan
178 RETools.RareEventTools.SetObjOptovarbyName(objOV, optoOV, False, False)
179
180 if SWAF_beforeOV==True:
181     try:
182         # initial focussing via FindSurface to assure a good starting position
183         # requires DF2 for Observer or Celldiscoverer 7 --> otherwise comment line !!!
184         Zen.Acquisition.FindSurface()
185     except:
186         print 'Was not able to run Find Surface.'
187
188     try:
189         # run the SWAF using the settings from the OVScan --> check SWAF  for overview experiment !!!
190         Zen.Acquisition.FindAutofocus(OVScan)
191     except:
192         print 'Was not able to run SWAF using the seetings: ', OverViewExpName
193
194 # get the resulting z-position
195 znew = Zen.Devices.Focus.ActualPosition
196
197 # try to adapt the Tile Experiment with new Z-Position
198 try:
199     Tiles.TileTools.ModifyTileRegionsZonly(OVScan, znew)
200     print 'Adapted Z-Position of Tile OverView. New Z = ',znew
201 except:
202     print 'Was not able to adapt Z-Position of Overview Scan Experiment.'
203
204 # execute the experiment
205 print 'Running OverviewScan Experiment.\n'
206 output_OVScan = Zen.Acquisition.Execute(OVScan)
207 OVScan.Close()
208 # For testing purposes - Load overview scan image automatically instead of executing the "real" experiment
209 #output_OVScan =
     ↪    Zen.Application.LoadImage(r'c:\Users\M1SRH\Documents\Testdata_Zeiss\RareEvent_Test_Wizard\OverViewScan_Test_raw.czi',
     ↪    False
210
211 # show the overview scan inside the document area
212 Zen.Application.Documents.Add(output_OVScan)
213 ovdoc = Zen.Application.Documents.GetByName(output_OVScan.Name)
214
215 # save the overview scan image inside the select folder
216 output_OVScan.Save(OutputFolder + '\\' + ovscan_name)
217
218 ############ 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 software autofocus.
3. Store the resulting new Z-position.
4. Modify the tile overview experiment using the 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.

```
220 # Load analysis setting created by the wizard or an separate macro
221 ias = ZenImageAnalysisSetting()
222 # for simulation use: 000 - RareEventExample.czias
223 ias.Load(ImageAS)
224 # Analyse the image
225 Zen.Analyzing.Analyze(output_OVScan,ias)
226 # Create Zen table with results for all detected objects (parent class)
227 AllObj = Zen.Analyzing.CreateRegionsTable(output_OVScan)
228 # Create Zen table with results for each single object
229 SingleObj = Zen.Analyzing.CreateRegionTable(output_OVScan)
230
231 # check for existence of required column names
232 DetailScan = Zen.Acquisition.Experiments.GetByName(DetailExpName)
233 DetailScan.SetActive()
234 DetailIsTileExp = Tiles.TileTools.IsTilesExperiment(DetailScan)
235 out =  RETools.RareEventTools.CheckColumns(SingleObj, DetailIsTileExp)
236 DetailScan.Close()
237
238 # 1st item is a bool indicating if all required columns could be found
239 columnsOK = out.Item1
240
241 if columnsOK == False:
242     print 'Execution stopped. Required Columns are missing.'
243     raise Exception('Execution stopped. Required Columns are missing.')
244
245 # 2nd item is a dictionary containing the tile properties
246 # colID: used keys = 'BCcolx' / 'BCcoly' / 'BCWidthcolx' / 'BCHeightcoly'
247 colID = out.Item2
248
249 # show and save data tables to the specified folder
250 Zen.Application.Documents.Add(AllObj)
251 Zen.Application.Documents.Add(SingleObj)
252 AllObj.Save(OutputFolder + '\\OverviewTable.csv')
253 SingleObj.Save(OutputFolder + '\\SingleObjectsTable.csv')
254
255 # check the number of detected objects = rows inside image analysis table
256 num_POI = SingleObj.RowCount
257
258 # switch to new magnification and optovar before the detailed scan starts
259 RETools.RareEventTools.SetObjOptovarbyName(objDT, optoDT, False, False)
260
261 # determine offset between FindSurface and objects found by SWAF --> required DF2
262 if CheckOffset==True:
263     # move to the first valid XY position to detemine the offset
264     xpos_1st   = SingleObj.GetValue(0, colID['BCcolx']) # get x stage position from list
265     ypos_1st   = SingleObj.GetValue(0, colID['BCcoly']) # get y stage position from list
266     Zen.Devices.Stage.MoveTo(xpos_1st + dx_LSM, ypos_1st + dy_LSM)
267
268     # get the actual offset using FindSurface followed by SWAF --> requires DF2
269     try:
270         dzFS = RETools.RareEventTools.getOffsetFindSurfaceSWAF(DetailExpName, False)
271         print 'Automatic Offset dzFS measured [micron]: ', round(dzFS, 2)
272         # store this new value inside the DF in order to use RecallFocus later on
273         print 'Store Z-Value Offset for RecallFocus.\n '
274         # store offset inside DF to use it with Recall Focus using DF2
275         Zen.Acquisition.StoreFocus()
276         useRecallFocus = True
277     except:
278         print 'Automatic Offset Check failed. Setting Offset dzFS = 0.'
279         dzFS = 0.0
280         useRecallFocus = False
281
282 elif CheckOffset==False:
283     dzFS = 0.0
284     useRecallFocus = False
285
286 if ManualOffset==True and CheckOffset==False:
287     # use the manually entered offset only when dz was not measured automatically
288     dzFS = OffsetValue
289     print ' Manual Offset [micron]: ', round(dzFS, 2)
```

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.

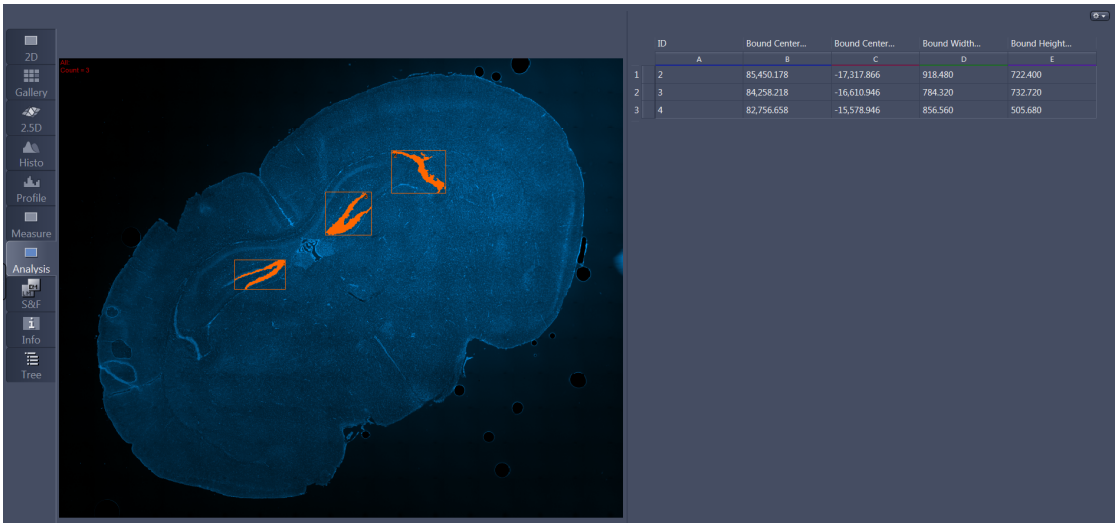

Figure 9: The results of the overview scan and the image analysis.

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.

3.7 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*.

```
294 ############# START DETAILED SCAN EXPERIMENT #############
295
296 # execute detailed experiment at the position of every detected object
297 for i in range(0, num_POI, 1):
298
299     # get the object information from the position table
300     POI_ID = SingleObj.GetValue(i, 0) # get the ID of the object - IDs start with 2 !!!
301     xpos = SingleObj.GetValue(i, colID['BCcolx']) # get X-stage position from table
302     ypos = SingleObj.GetValue(i, colID['BCcoly']) # get Y-stage position from table
303
304     # move to the current position
305     Zen.Devices.Stage.MoveTo(xpos + dx_LSM, ypos + dy_LSM) # comment this, if one uses a simulated experiment
306     print 'Moving Stage to Object ID:', POI_ID, ' at :', round(xpos, 2), round(ypos, 2)
307
308     if useRecallFocus == False:
309         # Initial FindSurface before the Detail Scan starts
310         try:
311             Zen.Acquisition.FindSurface()
312         except:
313             'Was not able to run Find Surface.'
314         # calculate new focus position plus offset and move z-drive
315         zpos = Zen.Devices.Focus.ActualPosition + dzFS
316         print 'Move to Z-Position: ', round(zpos, 2)
317         Zen.Devices.Focus.MoveTo(zpos)
318
319     elif useRecallFocus == True:
320         # alternative solution - use RecallFocus
321         try:
322             Zen.Acquisition.RecallFocus()
323         except:
324             print 'Was not able to run Recall Focus.'
325         zpos = Zen.Devices.Focus.ActualPosition
326         print 'New z-position after Recall Focus: ', zpos
327
328     # load the predefined detailed scan experiment
329     DetailScan = Zen.Acquisition.Experiments.GetByName(DetailExpName)
330
331     # only modify the Tile Properties if required IAS features BoundWidth and BoundHeight were found
332     # if experiment is a Tile Experiment
333     if DetailIsTileExp == True:
334
335         # Modify tile center position - get bounding rectangle width & height in microns
336         bcwidth =  SingleObj.GetValue(i, colID['BCWidthcolx'])
337         bcheight = SingleObj.GetValue(i, colID['BCHeightcoly'])
338         print 'Width and Height : ', str(round(bcwidth, 2)), str(round(bcheight, 2))
339         print 'Modifying Tile Properties XYZ Position and width & height.'
340         # Modify the XYZ position of the tile region on-the-fly
341         Tiles.TileTools.ModifyTileRegionsXYZ(DetailScan, xpos, ypos, zpos)
342         # Modify ConturSize for the tile according to the size of the bounding rectangle
343         Tiles.TileTools.ModifyTileRegionsSize(DetailScan, bcwidth, bcheight)
344         print 'New Tile Properties: ', round(xpos, 2), round(ypos, 2), round(zpos, 2), round(bcwidth, 2), round(bcheight, 2)
```

3.8 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.

```
346     # execute the experiment
347     print 'Running Detail Scan Experiment at new XYZ position.'
348
349     # run the Detail Scan
350     output_detailscan = Zen.Acquisition.Execute(DetailScan)
351     DetailScan.Close()
352
353     # get the image data name
354     dtscan_name = output_detailscan.Name
355     # save the image data to the selected folder and close the image
356     output_detailscan.Save(OutputFolder + '\\' + output_detailscan.Name)
357     output_detailscan.Close()
358     # rename the CZI regarding to the object ID - Attention - IDs start with 2 !!!
359     newname_dtscan = 'DTScan_ID' + str(POI_ID) + '.czi'
360     if verbose:
361         print 'Renaming File: ' + dtscan_name + ' to: ' + newname_dtscan + '\n'
362     File.Move(OutputFolder + '\\' + dtscan_name, OutputFolder + '\\' + newname_dtscan)
363
364 ############# END DETAILED SCAN EXPERIMENT #############
365
366 # show the overview scan document again at the end
367 Zen.Application.Documents.ActiveDocument = ovdoc
368 print 'All Positions done. RareEvent Detection finished.'
```

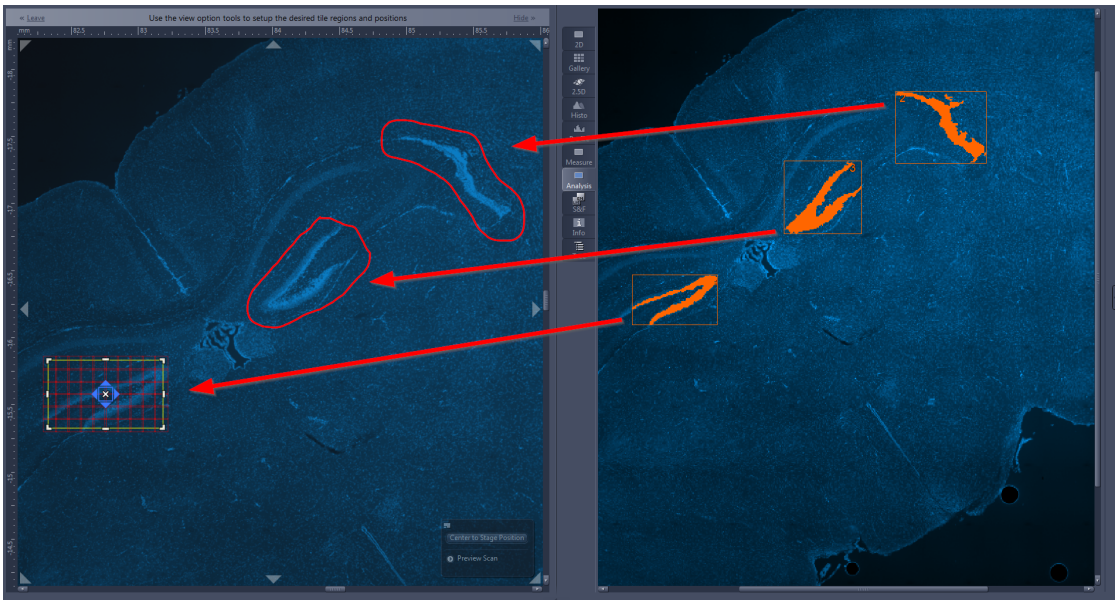

Figure 10: The results of the detailed scan - tile regions adapts to objects.

If the optional output(verbose = True) option was used, one can see check what is going on inside the IDE Watch viewblock of the OAD Macro editor in ZEN Blue.

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.

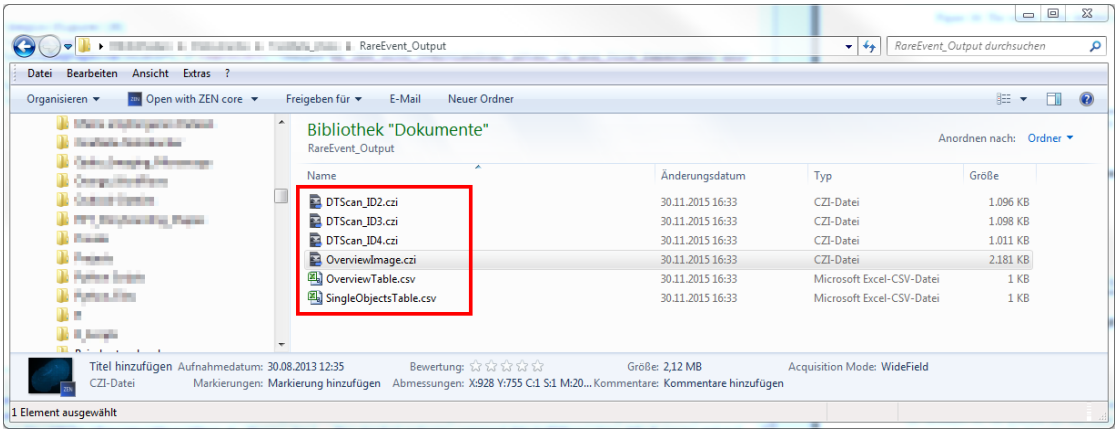

Figure 11: The results of the detailed scan stored inside a folder.

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.

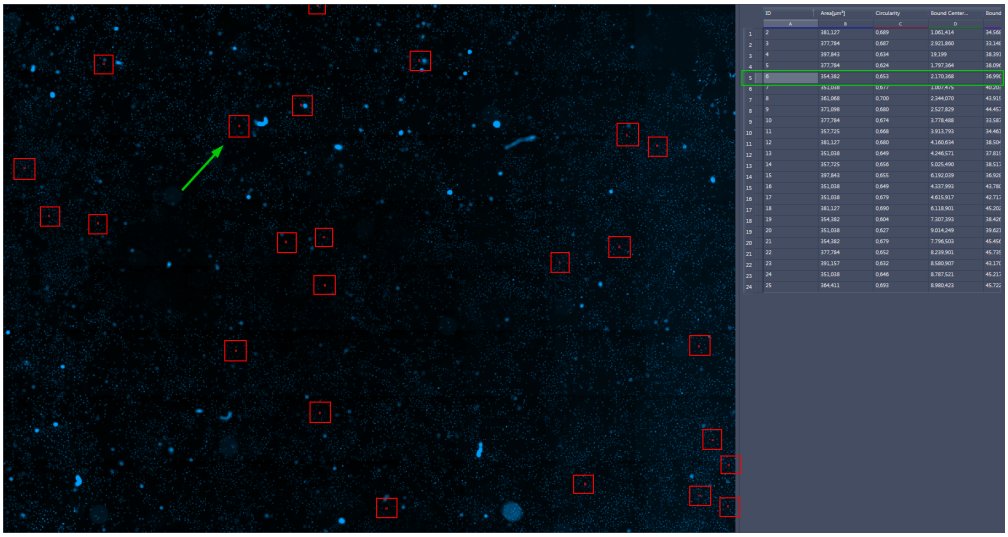

Figure 12: The results of the overview scan and image analysis.

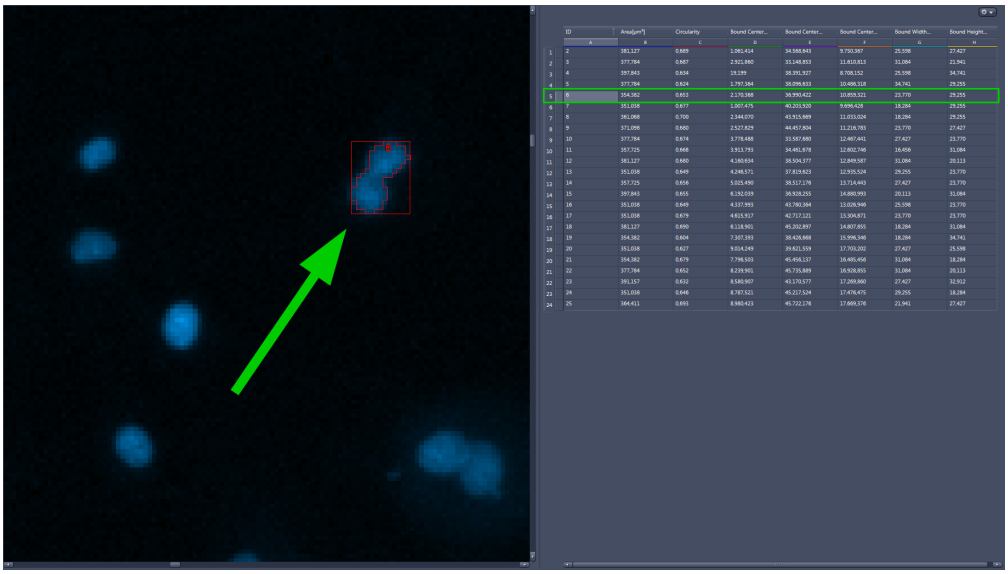

Figure 13: Detailed view on a detected object.

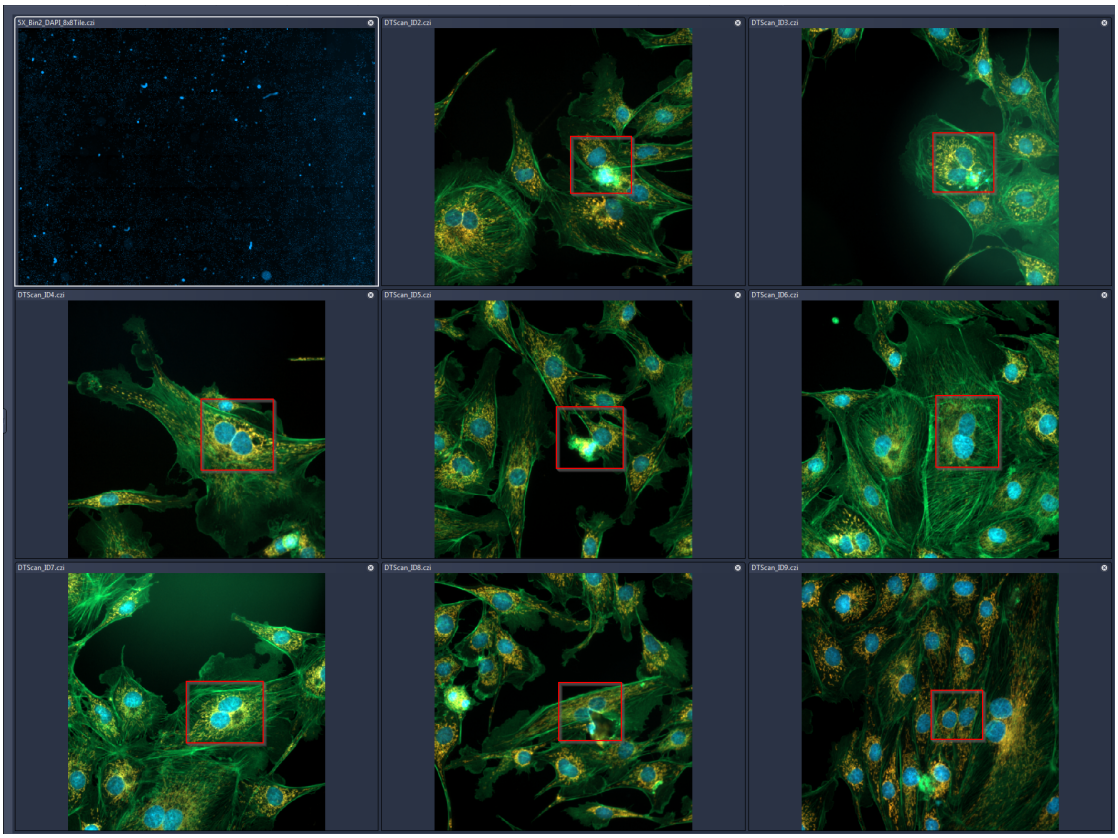

Figure 14: Detailed images from some of the detected objects.

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 eb 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.

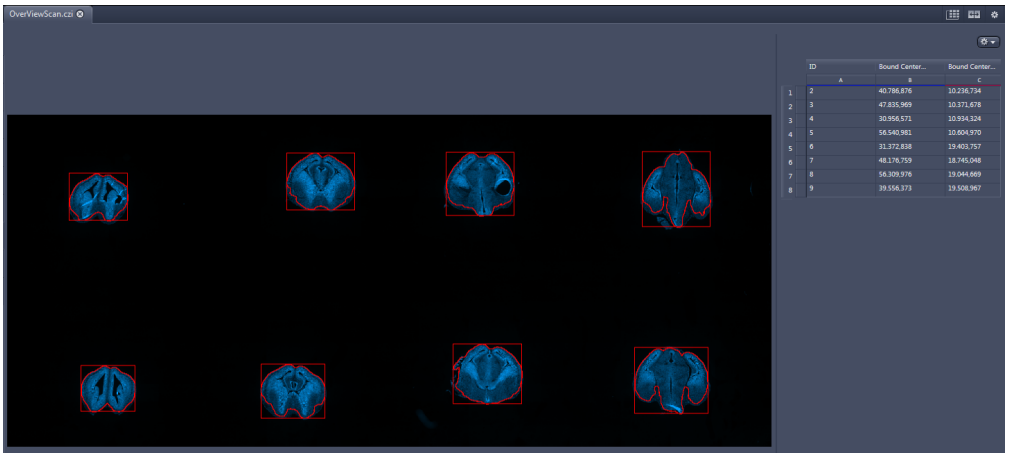

Figure 15: The results of the overview scan and image analysis.

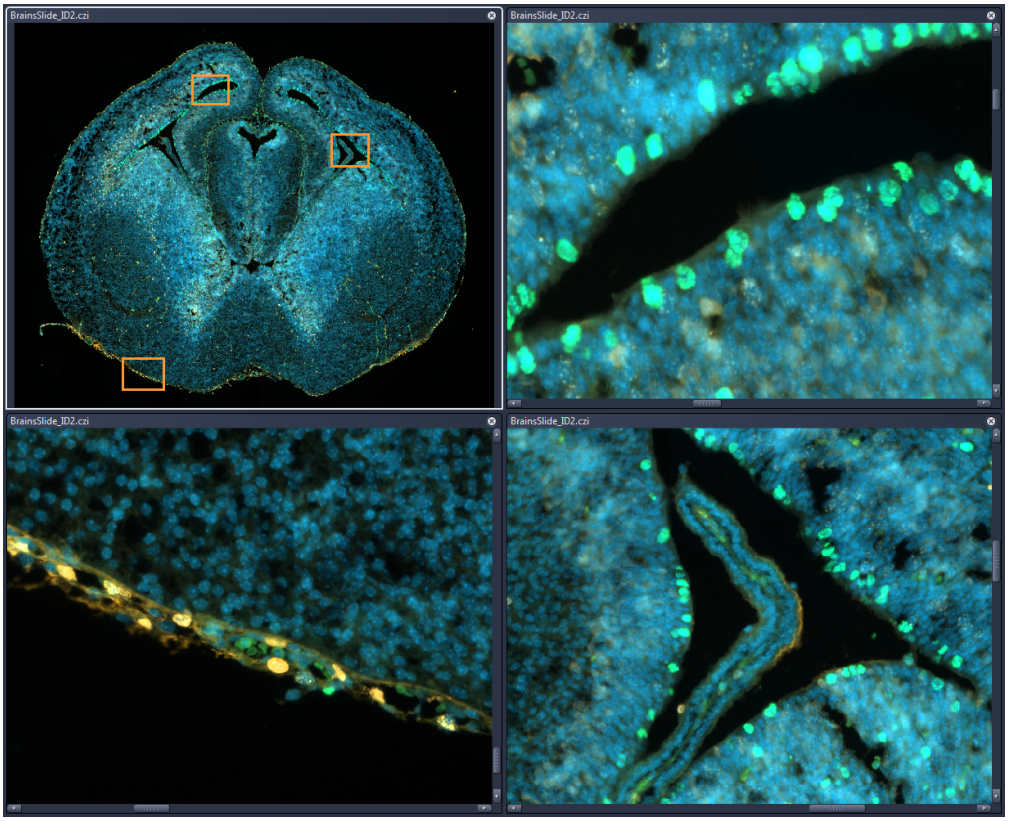

Figure 16: A more detailed image of the first specimen with three zoomed-in regions.

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 Appendix: Python Scripts

6.1 Guided_Acquisition.czmac

This the the complete ZEN Blue python script used to realize the workflow describe 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.

```
"""
Author: Sebastian Rhode
Date: 2017_08_06
File: Guided_Acquistion_DLLs.czmac
Version: 4.3

Optimzed for the use with Celldiscoverer 7 and DF2.
Please adapt Focussing commands, especially FindSurface when using with other stands

1) - Select Overview Scan Experiment
2) - Select appropriate Image Analysis Pipeline
3) - Select Detailed Scan Experiment
4) - Specify the output folder for the image and data tables

Requires the following DLLs to be found inside the Zen program folder:

- RETools.dll
- TileTools.dll
"""

# import DLLs
import clr
clr.AddReference('RETools.dll')
clr.AddReference('TileTools.dll')
import RETools
import Tiles
import time
from datetime import datetime
import errno

from System.IO import File, Directory, Path
import sys

# optional debugging output
verbose = True
# version number for dialog window
version = 4.3
# file name for overview scan
ovscan_name = 'OverviewScan.czi'
# additional XY offest for possible LSM port relative to the Camera port (zero)
dx_LSM = 0.0
dy_LSM = 0.0

def dircheck(basefolder):

    # check if the destination basefolder exists
    base_exists = Directory.Exists(basefolder)
    if base_exists:
        # specify the desired output format for the folder, e.g. 2017-08-08_17-47-41
        format = '%Y-%m-%d_%H-%M-%S'
        # create the new directory
        newdir = createfolder(basefolder, formatstring=format)
        print 'Created new directory: ', newdir

    return newdir


def createfolder(basedir, formatstring='%Y-%m-%d_%H-%M-%S'):
    # construct new directoty name nased on date and time
    newdir = Path.Combine(basedir, datetime.now().strftime(formatstring))
    # check if the new directory (for whatever reasons) already exists
    try:
```

```
        newdir_exists = Directory.Exists(newdir)
        if not newdir_exists:
            # create new directory if is does not exist
            Directory.CreateDirectory(newdir)
        if newdir_exists:
            # raise error if it really already exists
            raise SystemExit
    except OSError as e:
        if e.errno != errno.EEXIST:
            newdir = None
            raise  # This was not a "directory exist" error..

    return newdir

# clear console output
Zen.Application.MacroEditor.ClearMessages()

# check the location of folder where experiment setups and image analysis settings are stored
docfolder =  Zen.Application.Environment.GetFolderPath(ZenSpecialFolder.UserDocuments)
#imgfolder =  Zen.Application.Environment.GetFolderPath(ZenSpecialFolder.ImageAutoSave)
imgfolder = r'c:\Output\Guided_Acquistion'
format = '%Y-%m-%d_%H-%M-%S'

# get list with all existing experiments and image analysis setup
expfiles = RETools.RareEventTools.GetExperimentSetups(docfolder)
ipfiles = RETools.RareEventTools.GetImageAnalysisSetups(docfolder)

# collect information about the configured objectives and optovars
objectives = RETools.RareEventTools.CreateObjectiveInfo()
optovars = RETools.RareEventTools.CreateOptovarInfo()
if verbose:
    print('Objectives:')
    print('Number: ', objectives.Count)
    print('Names: ', objectives.Names)
    print('Positions: ', objectives.Positions)
    print('-----------------------------------------')
    print('Optovars:')
    print('Number: ', optovars.Count)
    print('Names: ', optovars.Names)
    print('Positions: ', optovars.Positions)
    print('\n')

# Activate Zen.Application
RareEventDialog = ZenWindow()
RareEventDialog.Initialize('Guided Acquisition - Version : ' + str(version), 650, 750, True, True)
# add components to dialog
RareEventDialog.AddLabel('1) Select Overview Experiment  ----------------------------')
RareEventDialog.AddDropDown('overview_exp', 'Overview Scan Experiment', expfiles, 0)
RareEventDialog.AddDropDown('objectiveOV', 'Objective OvervIew Scan', objectives.Names, 1)
RareEventDialog.AddDropDown('optovarOV', 'After-Mag OverViewScan', optovars.Names, 2)
RareEventDialog.AddCheckbox('SWAF_before_overview', 'OPTION - (Find Surface) & SWAF before Overview', True)
RareEventDialog.AddLabel('2) Select Image Analysis to detect objects  -----------------')
RareEventDialog.AddDropDown('ip_pipe', 'Image Analysis Pieline', ipfiles, 0)
RareEventDialog.AddLabel('3) Select Detail Scan Experiment  --------------------------')
RareEventDialog.AddDropDown('detailed_exp', 'Detailed Scan Experiment', expfiles, 1)
RareEventDialog.AddDropDown('objectiveDT', 'Objective Detailed Scan', objectives.Names, 2)
RareEventDialog.AddDropDown('optovarDT', 'After-Mag Detailed Scan', optovars.Names, 2)
RareEventDialog.AddCheckbox('checkoffset', 'OPTION - Check offset between FS and SWAF', False)
RareEventDialog.AddCheckbox('manualoffset', 'OPTION - Enter Offset manually', False)
RareEventDialog.AddDoubleRange('offsetvalue', 'Enter manual Offset [micron]', 0, 0, 100)
RareEventDialog.AddLabel('4) Specify Output Folder to save the images -----------------')
RareEventDialog.AddFolderBrowser('outfolder','Output Folder for Images and Data Tables', imgfolder)

# show the window
result = RareEventDialog.Show()
if result.HasCanceled:
    message = 'Macro was canceled by user.'
    print message
    raise SystemExit

# get the values and store them
OverViewExpName = str(result.GetValue('overview_exp'))
ImageAS = str(result.GetValue('ip_pipe'))
DetailExpName = str(result.GetValue('detailed_exp'))
OutputFolder = str(result.GetValue('outfolder'))
SWAF_beforeOV = result.GetValue('SWAF_before_overview')
CheckOffset = result.GetValue('checkoffset')
ManualOffset = result.GetValue('manualoffset')
OffsetValue = result.GetValue('offsetvalue')
objOV = result.GetValue('objectiveOV')
optoOV = result.GetValue('optovarOV')
objDT = result.GetValue('objectiveDT')
optoDT = result.GetValue('optovarDT')

# print values - this is optional
print 'Overview Scan Experiment : ' + OverViewExpName
print 'Combination OverView Scan: ', objOV + ' + ' + optoOV
print 'Image Analysis Pipeline : ' + ImageAS
print 'Detailed Scan Experiment : ' + DetailExpName
```

```
print 'Combination Detailed Scan: ', objDT + ' + ' + optoDT
print 'Output Folder for Data : ' + OutputFolder, '\n'

# check directory
OutputFolder = dircheck(OutputFolder)

# check Detail Scan experiment for tiles
DetailScan = Zen.Acquisition.Experiments.GetByName(DetailExpName)
DetailScan.SetActive()
DetailIsTileExp = Tiles.TileTools.IsTilesExperiment(DetailScan)
DetailScan.Close()

# check Overview Scan for tiles
OVScan = Zen.Acquisition.Experiments.GetByName(OverViewExpName)
OVScan.SetActive()
OVScanIsTileExp = Tiles.TileTools.IsTilesExperiment(OVScan)

if (OVScanIsTileExp == False or DetailIsTileExp == False):
    message = 'Overview or Detail Scan are not a Tile Experiment.'
    print message
    raise SystemExit
    OVScan.Close()

############# START OVERVIEW SCAN EXPERIMENT #################

# use the correct objective and optovar for the overview scan
RETools.RareEventTools.SetObjOptovarbyName(objOV, optoOV, False, False)

if SWAF_beforeOV==True:
    try:
        # initial focussing via FindSurface to assure a good starting position
        # requires DF2 for Observer or Celldiscoverer 7 --> otherwise comment line !!!
        Zen.Acquisition.FindSurface()
    except:
        print 'Was not able to run Find Surface.'

    try:
        # run the SWAF using the settings from the OVScan --> check SWAF  for overview experiment !!!
        Zen.Acquisition.FindAutofocus(OVScan)
    except:
        print 'Was not able to run SWAF using the seetings: ', OverViewExpName

# get the resulting z-position
znew = Zen.Devices.Focus.ActualPosition

# try to adapt the Tile Experiment with new Z-Position
try:
    Tiles.TileTools.ModifyTileRegionsZonly(OVScan, znew)
    print 'Adapted Z-Position of Tile OverView. New Z = ',znew
except:
    print 'Was not able to adapt Z-Position of Overview Scan Experiment.'

# execute the experiment
print 'Running OverviewScan Experiment.\n'
output_OVScan = Zen.Acquisition.Execute(OVScan)
OVScan.Close()
# For testing purposes - Load overview scan image automatically instead of executing the "real" experiment
#output_OVScan =
 ↪    Zen.Application.LoadImage(r'c:\Users\M1SRH\Documents\Testdata_Zeiss\RareEvent_Test_Wizard\OverViewScan_Test_raw.czi',
 ↪    False

# show the overview scan inside the document area
Zen.Application.Documents.Add(output_OVScan)
ovdoc = Zen.Application.Documents.GetByName(output_OVScan.Name)

# save the overview scan image inside the select folder
output_OVScan.Save(OutputFolder + '\\' + ovscan_name)

############# END OVERVIEW SCAN EXPERIMENT ###################

# Load analysis setting created by the wizard or an separate macro
ias = ZenImageAnalysisSetting()
# for simulation use: 000 - RareEventExample.czias
ias.Load(ImageAS)
# Analyse the image
Zen.Analyzing.Analyze(output_OVScan,ias)
# Create Zen table with results for all detected objects (parent class)
AllObj = Zen.Analyzing.CreateRegionsTable(output_OVScan)
# Create Zen table with results for each single object
SingleObj = Zen.Analyzing.CreateRegionTable(output_OVScan)

# check for existence of required column names
DetailScan = Zen.Acquisition.Experiments.GetByName(DetailExpName)
DetailScan.SetActive()
DetailIsTileExp = Tiles.TileTools.IsTilesExperiment(DetailScan)
out =  RETools.RareEventTools.CheckColumns(SingleObj, DetailIsTileExp)
DetailScan.Close()

# 1st item is a bool indicating if all required columns could be found
```

```
columnsOK = out.Item1

if columnsOK == False:
    print 'Execution stopped. Required Columns are missing.'
    raise Exception('Execution stopped. Required Columns are missing.')

# 2nd item is a dictionary containing the tile properties
# colID: used keys = 'BCcolx' / 'BCcoly' / 'BCWidthcolx' / 'BCHeightcoly'
colID = out.Item2

# show and save data tables to the specified folder
Zen.Application.Documents.Add(AllObj)
Zen.Application.Documents.Add(SingleObj)
AllObj.Save(OutputFolder + '\\OverviewTable.csv')
SingleObj.Save(OutputFolder + '\\SingleObjectsTable.csv')

# check the number of detected objects = rows inside image analysis table
num_POI = SingleObj.RowCount

# switch to new magnification and optovar before the detailed scan starts
RETools.RareEventTools.SetObjOptovarbyName(objDT, optoDT, False, False)

# determine offset between FindSurface and objects found by SWAF --> required DF2
if CheckOffset==True:
    # move to the first valid XY position to detemine the offset
    xpos_1st   = SingleObj.GetValue(0, colID['BCcolx']) # get x stage position from list
    ypos_1st   = SingleObj.GetValue(0, colID['BCcoly']) # get y stage position from list
    Zen.Devices.Stage.MoveTo(xpos_1st + dx_LSM, ypos_1st + dy_LSM)

    # get the actual offset using FindSurface followed by SWAF --> requires DF2
    try:
        dzFS = RETools.RareEventTools.getOffsetFindSurfaceSWAF(DetailExpName, False)
        print 'Automatic Offset dzFS measured [micron]: ', round(dzFS, 2)
        # store this new value inside the DF in order to use RecallFocus later on
        print 'Store Z-Value Offset for RecallFocus.\n '
        # store offset inside DF to use it with Recall Focus using DF2
        Zen.Acquisition.StoreFocus()
        useRecallFocus = True
    except:
        print 'Automatic Offset Check failed. Setting Offset dzFS = 0.'
        dzFS = 0.0
        useRecallFocus = False

elif CheckOffset==False:
    dzFS = 0.0
    useRecallFocus = False

if ManualOffset==True and CheckOffset==False:
    # use the manually entered offset only when dz was not measured automatically
    dzFS = OffsetValue
    print ' Manual Offset [micron]: ', round(dzFS, 2)

# wait just to be sure everything is finished
time.sleep(2)

############# START DETAILED SCAN EXPERIMENT #############

# execute detailed experiment at the position of every detected object
for i in range(0, num_POI, 1):

    # get the object information from the position table
    POI_ID = SingleObj.GetValue(i, 0) # get the ID of the object - IDs start with 2 !!!
    xpos = SingleObj.GetValue(i, colID['BCcolx']) # get X-stage position from table
    ypos = SingleObj.GetValue(i, colID['BCcoly']) # get Y-stage position from table

    # move to the current position
    Zen.Devices.Stage.MoveTo(xpos + dx_LSM, ypos + dy_LSM) # comment this, if one uses a simulated experiment
    print 'Moving Stage to Object ID:', POI_ID, ' at :', round(xpos, 2), round(ypos, 2)

    if useRecallFocus == False:
        # Initial FindSurface before the Detail Scan starts
        try:
            Zen.Acquisition.FindSurface()
        except:
            'Was not able to run Find Surface.'
        # calculate new focus position plus offset and move z-drive
        zpos = Zen.Devices.Focus.ActualPosition + dzFS
        print 'Move to Z-Position: ', round(zpos, 2)
        Zen.Devices.Focus.MoveTo(zpos)

    elif useRecallFocus == True:
        # alternative solution - use RecallFocus
        try:
            Zen.Acquisition.RecallFocus()
        except:
            print 'Was not able to run Recall Focus.'
        zpos = Zen.Devices.Focus.ActualPosition
        print 'New z-position after Recall Focus: ', zpos
```

```
    # load the predefined detailed scan experiment
    DetailScan = Zen.Acquisition.Experiments.GetByName(DetailExpName)

    # only modify the Tile Properties if required IAS features BoundWidth and BoundHeight were found
    # if experiment is a Tile Experiment
    if DetailIsTileExp == True:

        # Modify tile center position - get bounding rectangle width & height in microns
        bcwidth =  SingleObj.GetValue(i, colID['BCWidthcolx'])
        bcheight = SingleObj.GetValue(i, colID['BCHeightcoly'])
        print 'Width and Height : ', str(round(bcwidth, 2)), str(round(bcheight, 2))
        print 'Modifying Tile Properties XYZ Position and width & height.'
        # Modify the XYZ position of the tile region on-the-fly
        Tiles.TileTools.ModifyTileRegionsXYZ(DetailScan, xpos, ypos, zpos)
        # Modify ConturSize for the tile according to the size of the bounding rectangle
        Tiles.TileTools.ModifyTileRegionsSize(DetailScan, bcwidth, bcheight)
        print 'New Tile Properties: ', round(xpos, 2), round(ypos, 2), round(zpos, 2), round(bcwidth, 2), round(bcheight, 2)

    # execute the experiment
    print 'Running Detail Scan Experiment at new XYZ position.'

    # run the Detail Scan
    output_detailscan = Zen.Acquisition.Execute(DetailScan)
    DetailScan.Close()

    # get the image data name
    dtscan_name = output_detailscan.Name
    # save the image data to the selected folder and close the image
    output_detailscan.Save(OutputFolder + '\\' + output_detailscan.Name)
    output_detailscan.Close()
    # rename the CZI regarding to the object ID - Attention - IDs start with 2 !!!
    newname_dtscan = 'DTScan_ID' + str(POI_ID) + '.czi'
    if verbose:
        print 'Renaming File: ' + dtscan_name + ' to: ' + newname_dtscan + '\n'
    File.Move(OutputFolder + '\\' + dtscan_name, OutputFolder + '\\' + newname_dtscan)

############# END DETAILED SCAN EXPERIMENT #############

# show the overview scan document again at the end
Zen.Application.Documents.ActiveDocument = ovdoc
print 'All Positions done. RareEvent Detection finished.'
```

6.2 Custom DLLs

The python script required two custom DLLs:

- **RETools.dll** - provides required functions for the Guided Acquisition.
- **TileTools.dll** - provides required functions to modify a ZEN Tile experiment.

Those DLLs have to be present in the ZEN program folder, which is usually located here (depending on the used software version):

"C:/Program Files/Carl Zeiss/ZEN 2/ZEN 2 (blue edition)".

7 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.
- More methods to modify experiments and tile regions are planned for the next bigger ZEN Blue release.

8 Disclaimer

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

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 this example you agree to this disclaimer.