Contents

1 Introduction

This application note will explain how create a workflow using the ZEN-MATLAB connection. The basic idea is to control everything from within MATLAB (Master).

The ZEN Image Acquisition (Slave) software is "only" doing the image acquisition. The signal to start the experiment is send from MATLAB to ZEN.

When the experiment is finished, the CZI data are imported into MATLAB using Bio-Formats and "some" simple image analysis is carried out to underline the workflow concept.

2 Prerequisites

To be able to use the functionality described herein, you need

- **ZEN Blue 2012 or later with Macro Environment module**
- **MATLAB version R2013 - Standard**

To profit fully from the following example, you should have working experience with both applications. You should be familiar with the user interface in MATLAB, specifically with the Command Window and its use, and with the creation and use of scripts. Computer jargon has been avoided as much as possible; however, some basic, Wikipedia-level, knowledge about object models, classes and inter-process communication is expected.

2.1 Register ZEN Functionality

To be able to use ZEN services in a .COM environment, provided by MATLAB, the ZEN functionality must be made registered as follows using a BAT-File, which **must be executed as administrator**.

```
echo off
pushd "C:\Windows\Microsoft.NET\Framework64\v4.0.30319"

SET dll-1="C:\Program Files\Carl Zeiss\ZEN 2\ZEN 2 (blue edition)\Zeiss.Micro.Scripting.dll"
regasm /u /codebase /tlb %dll-1%
regasm /codebase /tlb %dll-1%

SET dll-2="C:\Program Files\Carl Zeiss\ZEN 2\ZEN 2 (blue edition)\Zeiss.Micro.LM.Scripting.dll"
regasm /u /codebase /tlb %dll-2%
regasm /codebase /tlb %dll-2%
popd
pause
```

The final result of this set of commands is that the .NET classes and structures within **Zeiss.Micro.Scripting.dll** and **Zeiss.Micro.LM.Scripting.dll** are made known and accessible within .COM (**Component Object Model**) environment as well.

Remark: Please note that you must edit the BAT-file and adapt the path location, where the DLLs can be found to your needs.

2.2 Starting MATLAB and ZEN

This 1st thing one must do is to start MATLAB and check, if the automation services are activated.

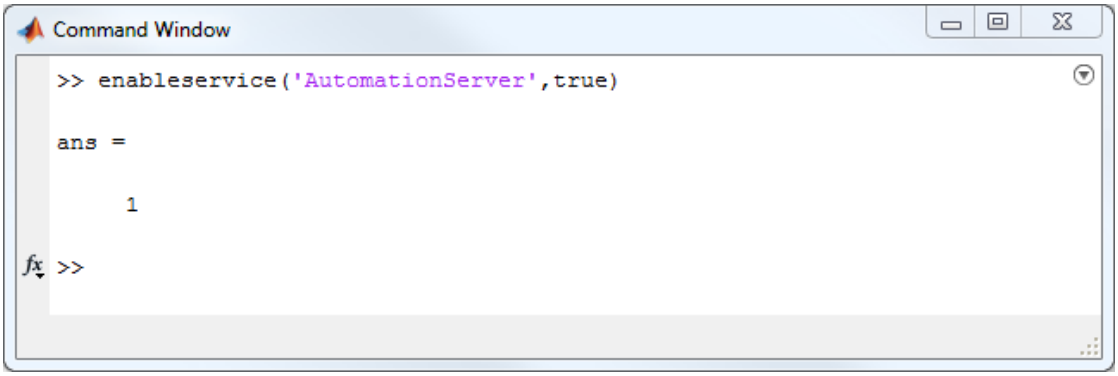

Figure 1: MATLAB - Enable Automation Server

Now you can start the ZEN Blue software and create the following OAD Python script. Import is to check if the correct MATLAB Application ID was used.

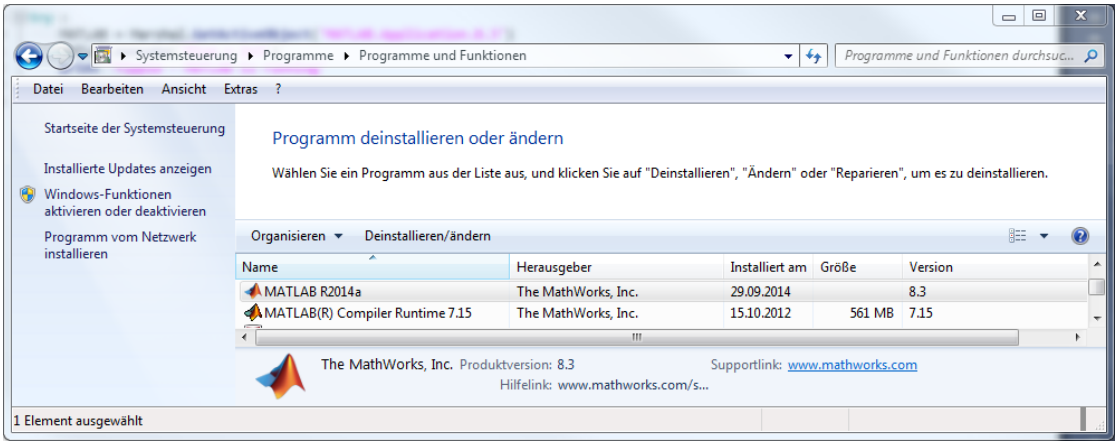

Figure 2: MATLAB - Get Application ID for the current MATLAB version

This is the OAD python script to test, if MATLAB is running.

```
# Author: Sebastian Rhode
# Date: 13.10.2014
# Version: 1.0

# A simple test to check if MATLAB is running from within an OAD python script

from System.Runtime.InteropServices import Marshal

try :
    MATLAB = Marshal.GetActiveObject('MATLAB.Application.8.3')
    MATLAB.execute('five = 2+3')
    print 'Yippie - MATLAB is running'
except :
    print 'Matlab is not running'
```

If one gets the correct output inside the ZEN Macro Editor Message Window, ZEN is fully aware of a "running MATLAB".

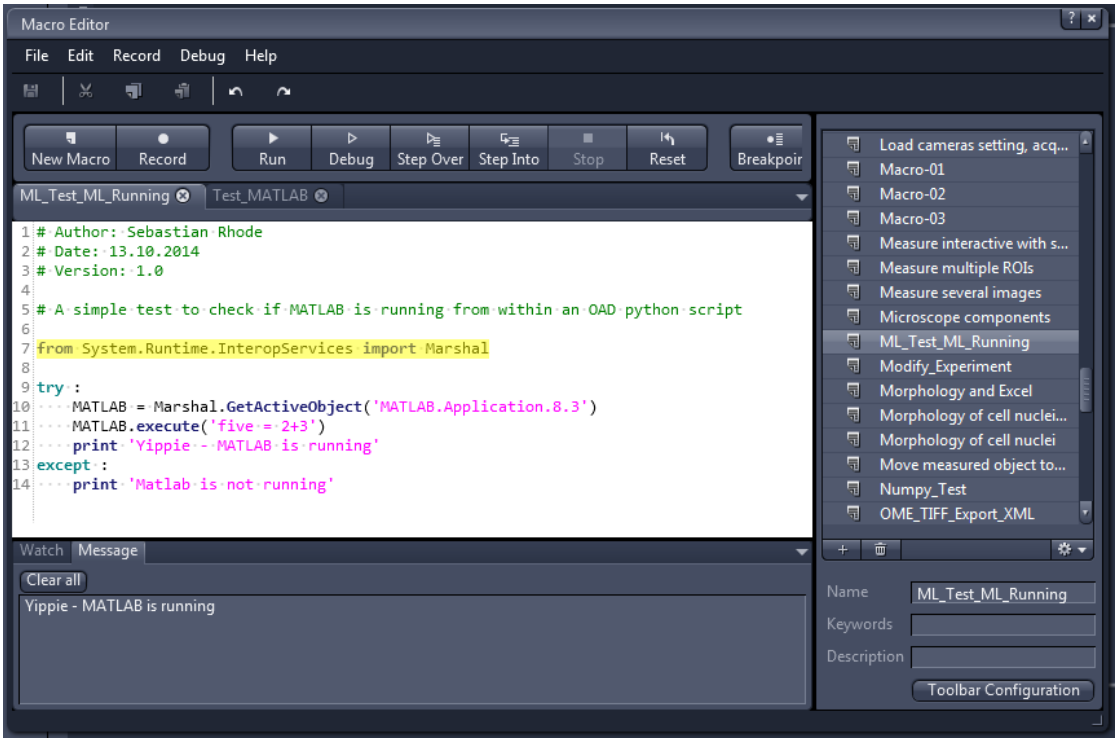

Figure 3: ZEN Blue - Test MATLAB from OAD python script

2.3 ZEN - Sample Camera

This is an extremely useful feature to test and demonstrate most of the capabilities of ZEN Blue Experiments without the need to have real hardware in place. The idea is to place some "real sample" images in a specific folder in order to use them for a simulated experiment including a real image analysis.

This requires a special XML-file which is usually located here: C:/ProgramData/Carl Zeiss/MTB2011/2.3.0.13/CZIS_Cameras.xml (Windows 7, depending on the current ZEN Blue and MTB version).

If configured correctly inside the MTB one can specify the image folder inside ZEN here:

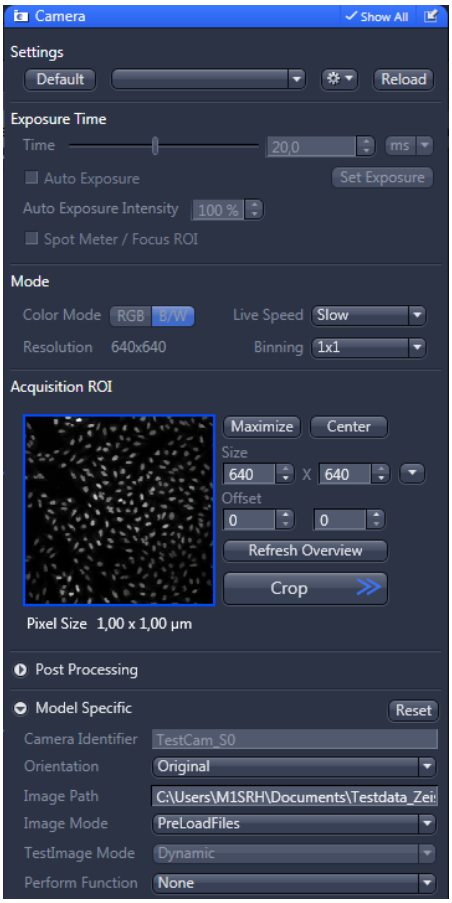

Figure 4: ZEN Blue - Sample Camera Configuration

Keep in mind that currently (12.09.2014) only single image files can be used to simulate for example a time lapse (10 single TIFF images = 10 frame CZI sequence).

3 ZEN Blue - Setting up the Experiment

The easiest way to setup a wellplate experiment is to use the Tile & Position Module. It is assumed that one is familiar with setting up ZEN Blue experiments in general since this is not explained in detail here.

- Specify the correct folder for your sample cam in case of a simulated experiment.
- Use Smart Setup to create a (DAPI) channel.
- Activate **Tiles**, change to **Position Setup** and select **Carrier** mode.
- Add **one** position per well.

Inside the ZEN Blue software this should similar tom this:

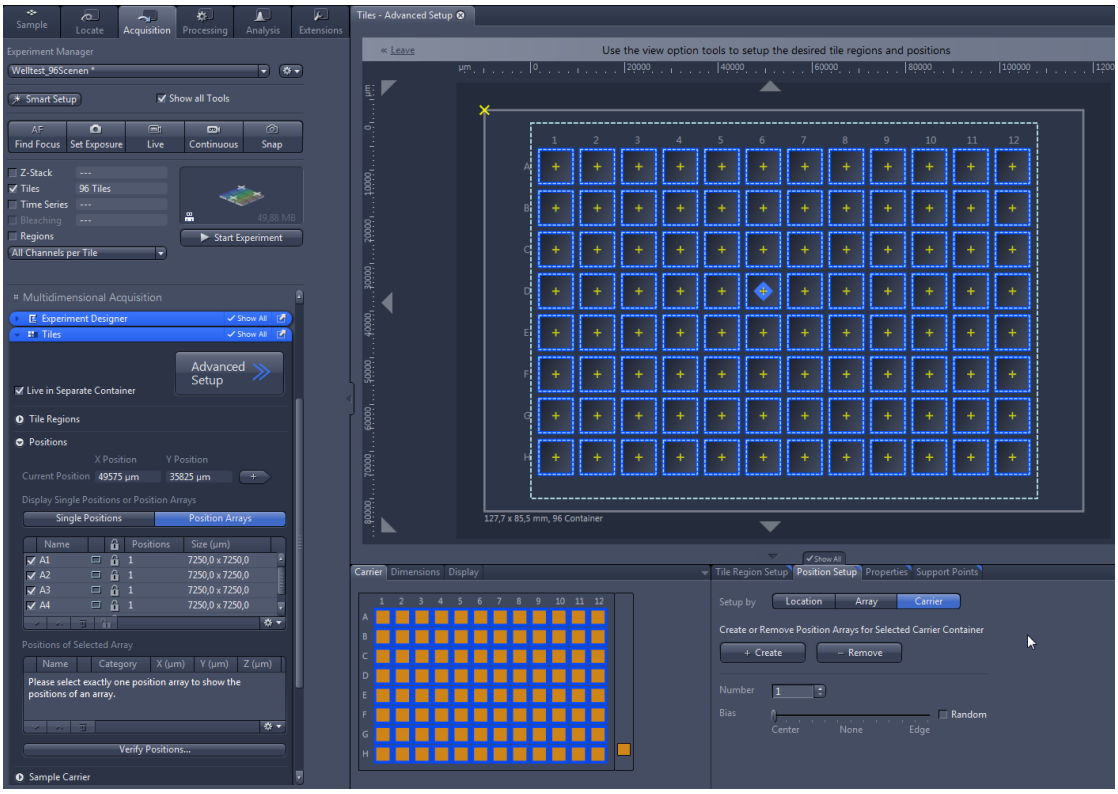

Figure 5: ZEN Blue - Wellplate Experiment Setup

4 MATLAB - Control ZEN from a M-File

The main idea behind controlling ZEN from within MATLAB is just to make MATLAB aware of the ZEN Scripting DLLs via the .COM interface. This way you can use all the commands available inside the OAD python scripts directly within MATLAB and any M-file.

The main ToDo´s in MATLAB inside this application note are:

- **Connect MATLAB to ZEN and run the actual experiment.**
- **Read CZI file using the MATLAB toolbox for the BioFormats library.**
- **Cycle trough all the images and do some "meaningful" analysis.**
- **Display the image analysis results.**

Of course the most important part of this example is to understand the implications of the 1st point. Once the ZEN-MATLAB bridge is established, the ZEN OAD Simple-API Interface becomes part of MATLAB an can be used in conjunction with the rest of MATLAB

4.1 Control ZEN from a M-file

This is the main M-file controlling the complete workflow. It establishes the connection to ZEN by making the OAD Simple-API available within MATLAB.

```
1  % Date: 17.03.2015
2  % Version: 1.1
3
4  % This MATLAB script demonstrates the capabilities of the .COM interface
5  % used to establish a connection between ZEN Blue and MATLAB.
6  % This connection allows to use ZEN Blue OAD Simple-API within a M-File and
7  % vice versa. This script only shows the 1st possibility.
8
9  % Import the ZEN Scripting into MATLAB
10 Zen = actxGetRunningServer('Zeiss.Micro.Scripting.ZenWrapperLM');
11
12 % Define place to store the CZI file
13 savefolder = 'C:\MATLAB_Output\';
14 % Define the experiment to be executed
15 ZEN_Experiment = 'ML_96_Wellplate_Observer.czexp';
16
17 % run the experiment in ZEN and save the data to the specified folder
18 exp = Zen.Acquisition.Experiments.GetByName(ZEN_Experiment);
19 img = Zen.Acquisition.Execute(exp);
20 % Show the image in ZEN
21 Zen.Application.Documents.Add(img);
22 % Use the correct save method - it is polymorphic ... :)
23 filename = [savefolder,img.Name]
24 img.Save_2(filename);
25
26 % Read the CZI using BioFormats
27 out = ReadImage6D(filename);
28 image6d = out{1};
29 metadata = out{2};
30
31 % Analyse the images - Example: Count Cells
32 [num, img2show, show] = AnalyzeSeries(image6d);
33
34 % Display data
35 DisplayObjectNumbers(num, img2show, show);
```

The most important line is number 10. Here you tell MATLAB to load the ZEN Scripting API. Once this is done, you can use all available ZEN objects (from the Simple API) the same way one would do it inside the ZEN software.

- **Define save folder and ZEN Experiment.**
- **Get the ZEN experiment using the correct name and run it.**
- **Read the CZI file using BioFormats.**
- **Analyze the image from every well and display the results.**

Of course there is little bit more behind the scenes than shown inside this script, but all the functionality to control ZEN from within MATLAB is located inside this short M-File.

This opens up a whole range of exciting uses cases. You can use all the possibilities of MATLAB for image and data analysis or even hardware control while having the option to use the powerful ZEN Blue acquisition engine for all your imaging experiments and controlling the microscope itself.

Remark:

In principle it is also possible to use MATLAB code within a ZEN OAD macro. In this case ZEN is the **Master** application and **MATLAB** acts as a Slave, but this workflow is **not** part of this example.

4.2 ZEN-MATLAB at work

Now we are ready to start the workflow by running **RunZenExperimentfromMATLAB.m** from within MATLAB.

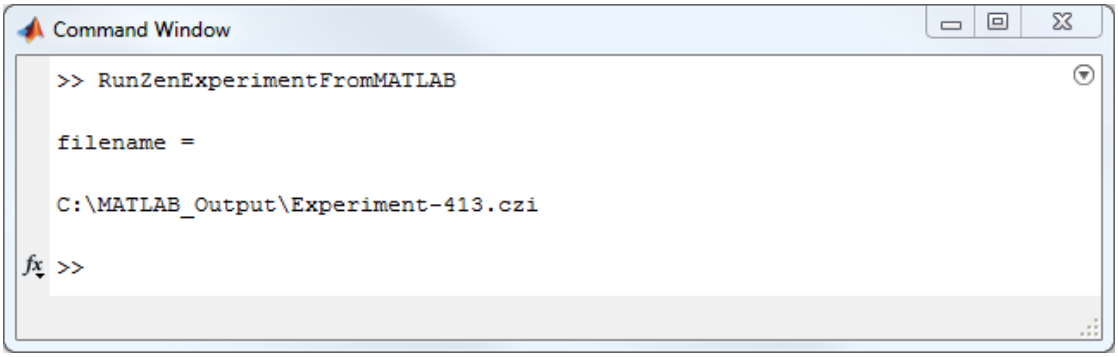

Figure 6: MATLAB - Start the workflow by running the M-file

The final result in MATLAB looks like this

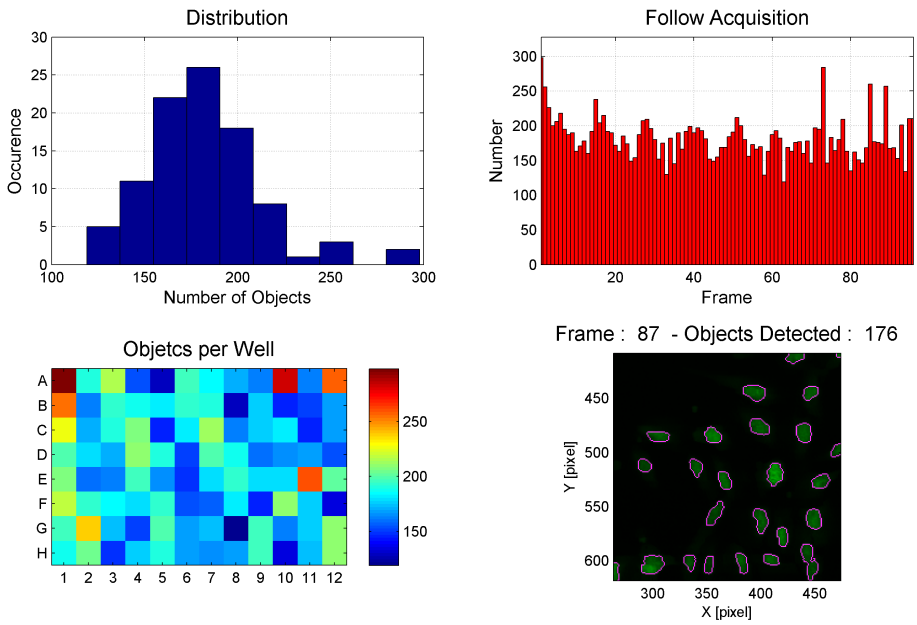

Figure 7: MATLAB - Display the results using built-in graphic tools

Remark: For better visibility the segmentation image (bottom-right) is zoomed in.

4.3 Importing CZI files into MATLAB

ZEN Blue used the CZI data format to store the image data. And due to its open concept it is very well supported by the BioFormats library. Instructions on the usage of the MATLAB wrapper for BioFormats can be found here:

http://www.openmicroscopy.org/site/support/bio-formats5/developers/matlab-dev.html

There is more than one way to import a CZI file, so the choice depends on the nature of the data. For this example we use a pretty generic approach tom read the CZI dataset. It is important to point out, the the way the data were acquired has a influence on the way one must import them into MATLAB. Especially the order of the dimension CZT (Channels - Z -Planes - Time Points) can be important.The script used to read the CZI data can be found in section **5.2**

4.4 Getting the CZI Metadata into MATLAB

A very important topic are the CZI Metadata. Again we can rely on the BioFormats library and just use the already existing functionality to get all the information we need. For the example we use the following script shown in **5.3**.

4.5 Displaying the analysis results

To make this example more descriptive a few additional scripts will be used to display the data and demonstrate the idea to automate a complete workflow within MATLAB. ZEN is a very powerful image acquisition software, but especially when it comes data analysis and crunching numbers MATLAB, one may needs to leave the ZEN world.

5 Appendix: MATLAB M-Files

5.1 RunZenExperimentFromMATLAB.m

This is the main M-file controlling the complete workflow.

```
% Date: 17.03.2015
% Version: 1.1

% This MATLAB script demonstrates the capabilities of the .COM interface
% used to establish a connection between ZEN Blue and MATLAB.
% This connection allows to use ZEN Blue OAD Simple-API within a M-File and
% vice versa. This script only shows the 1st possibility.

% Import the ZEN Scripting into MATLAB
Zen = actxGetRunningServer('Zeiss.Micro.Scripting.ZenWrapperLM');

% Define place to store the CZI file
savefolder = 'C:\MATLAB_Output\';
% Define the experiment to be executed
ZEN_Experiment = 'ML_96_Wellplate_Observer.czexp';

% run the experiment in ZEN and save the data to the specified folder
exp = Zen.Acquisition.Experiments.GetByName(ZEN_Experiment);
img = Zen.Acquisition.Execute(exp);
% Show the image in ZEN
Zen.Application.Documents.Add(img);
% Use the correct save method - it is polymorphic ... :)
filename = [savefolder,img.Name]
img.Save_2(filename);

% Read the CZI using BioFormats
out = ReadImage6D(filename);
image6d = out{1};
metadata = out{2};

% Analyse the images - Example: Count Cells
[num, img2show, show] = AnalyzeSeries(image6d);

% Display data
DisplayObjectNumbers(num, img2show, show);
```

5.2 ReadImage6D.m

This is the M-file for reading then CZI data using the MATLAB toolbox for the BioFormats library.

```
% File: ReadImage6D.m
% Author: Sebastian Rhode
% Date: 19.09.2016
% Version: 1.2

% Read CZI image data into image6d array

function out = ReadImage6D(filename)

% Get OME Meta-Information
MetaData = GetOMEData(filename);

% The main inconvenience of the bfopen.m function is that it loads all the content of an image regardless of its size.
% Initialize BioFormtas Reader
reader = bfGetReader(filename);

% add progress bar
h = waitbar(0,'Processing Data ...');
totalframes = MetaData.SeriesCount * MetaData.SizeC * MetaData.SizeZ * MetaData.SizeT;
framecounter = 0;

% Preallocate array with size (Series, SizeC, SizeZ, SizeT, SizeX, SizeY)
image6d = zeros(MetaData.SeriesCount, MetaData.SizeT, MetaData.SizeZ,  MetaData.SizeC, MetaData.SizeY, MetaData.SizeX);

for series = 1: MetaData.SeriesCount

    % set reader to current series
    reader.setSeries(series-1);
    for timepoint = 1: MetaData.SizeT
        for zplane = 1: MetaData.SizeZ
            for channel = 1: MetaData.SizeC

                framecounter = framecounter + 1;
                 % update waitbar
                wstr = {'Reading Images: ', num2str(framecounter), ' of ', num2str(totalframes),'Frames' };
                waitbar(framecounter / totalframes, h, strjoin(wstr))

                % get linear index of the plane (1-based)
                iplane = loci.formats.FormatTools.getIndex(reader, zplane - 1, channel - 1, timepoint -1) +1;
                % get frame for current series
                image6d(series, timepoint, zplane, channel, :, :) = bfGetPlane(reader, iplane);

            end
        end
    end
end

% close waitbar
close(h)

% close BioFormats Reader
reader.close();

% store image data and meta information in cell array
out = {};
% store the actual image data as 6d array
out{1} = image6d;
% store the image metainformation
out{2} = MetaData;

```

5.3 GetOMEData.m

This M-File reads all the relevant MetaData from the image. Please feel free to adapt it to your needs.

```
% File: GetOMEData.m
% Author: Sebastian Rhode
% Date: 13.09.2015
% Version: 1.3

function OMEData = GetOMEData(filename)
%
% Get OME Meta Information using BioFormats Library 5.1.10

% To access the file reader without loading all the data, use the low-level bfGetReader.m function:
reader = bfGetReader(filename);

% You can then access the OME metadata using the getMetadataStore() method:
omeMeta = reader.getMetadataStore();

% get ImageCount --> currently only reading one image is supported
imagecount = omeMeta.getImageCount();
% create empty cell array to store the image IDs
imageIDs_str = cell(1, imagecount);
imageIDs = cell(1, imagecount);

% try to get all the imageIDs as strings and numbers (zero-based)
try
    for id = 1:imagecount
        imageIDs_str{id} = omeMeta.getImageID(id-1);
        imageIDs{id} = id-1;
    end
    % store ine OMEData
    OMEData.ImageIDs = imageIDs;
    OMEData.ImageIDstrings = imageIDs_str;
catch
    OMEData.ImageIDs = 'na';
    OMEData.ImageIDstrings = 'na';
    msg = 'No suitable ImageIDs found.';
    warning(msg);
end

% use default imageID to read other metadata
imageID = imageIDs{1};

% get the actual metadata and store them in a structured array
[pathstr,name,ext] = fileparts(filename);
OMEData.FilePath = pathstr;
OMEData.Filename = strcat(name, ext);

% Get dimension order
OMEData.DimOrder = char(omeMeta.getPixelsDimensionOrder(imageID).getValue());

% Number of series inside the complete data set
OMEData.SeriesCount = reader.getSeriesCount();

% Dimension Sizes C - T - Z - X - Y
OMEData.SizeC = omeMeta.getPixelsSizeC(imageID).getValue();
OMEData.SizeT = omeMeta.getPixelsSizeT(imageID).getValue();
OMEData.SizeZ = omeMeta.getPixelsSizeZ(imageID).getValue();
OMEData.SizeX = omeMeta.getPixelsSizeX(imageID).getValue();
OMEData.SizeY = omeMeta.getPixelsSizeY(imageID).getValue();

% Scaling XYZ
try
    OMEData.ScaleX = round(double(omeMeta.getPixelsPhysicalSizeX(imageID).value()),3); % in micron
catch
    msg = 'Problem getting X-Scaling. Use Default = 1';
    warning(msg);
    OMEData.ScaleX = 1;
end

try
    OMEData.ScaleY = round(double(omeMeta.getPixelsPhysicalSizeY(imageID).value()),3); % in micron
catch
    msg = 'Problem getting Y-Scaling. Use Default = 1';
    warning(msg);
    OMEData.ScaleY = 1;
end


try
    OMEData.ScaleZ = round(double(omeMeta.getPixelsPhysicalSizeZ(imageID).value()),3); % in micron
```

```
catch
    % in case of only a single z-plane set to 1 micron ...
        msg = 'Problem getting Z-Scaling. Use Default = 1';
    warning(msg);
    OMEData.ScaleZ = 1;
end

% read relevant objective information from metadata
try
    % get the correct objective ID (the objective that was used to acquire the image)
    tmp = char(omeMeta.getInstrumentID(imageID));
    OMEData.InstrumentID = str2double(tmp(end));
    tmp = char(omeMeta.getObjectiveSettingsID(OMEData.InstrumentID));
    objID = str2double(tmp(end));
    % error handling --> sometime only one objective is there with ID > 0
    numobj = omeMeta.getObjectiveCount(OMEData.InstrumentID);
    if numobj == 1
        objID = 0;
    end

    OMEData.ObjID = objID;
catch
        msg = 'No suitable instrument and objective ID found.';
        warning(msg);
end

try
    % get objective immersion
    OMEData.ObjImm = char(omeMeta.getObjectiveImmersion(OMEData.InstrumentID, OMEData.ObjID).getValue());
catch
    msg = 'Problem getting immersion type.';
    warning(msg);
    OMEData.ObjImm = 'na';
end

try
    % get objective lens NA
    OMEData.ObjNA = round(omeMeta.getObjectiveLensNA(OMEData.InstrumentID, OMEData.ObjID).doubleValue(),2);
catch
    msg = 'Problem getting objective NA.';
    warning(msg);
    OMEData.ObjNA = 'na';
end

try
    % get objective magnification
    OMEData.ObjMag = round(omeMeta.getObjectiveNominalMagnification(OMEData.InstrumentID, OMEData.ObjID).doubleValue(),2);
catch
    msg = 'Problem getting objective magnification.';
    warning(msg);
    OMEData.ObjMag = 'na';
end

try
    % get objective model
    OMEData.ObjModel = char(omeMeta.getObjectiveModel(OMEData.InstrumentID, OMEData.ObjID));
catch
    msg = 'Problem getting objective model.';
    warning(msg);
    OMEData.ObjModel = 'na';
end

% get excitation and emission wavelengths for all channels
for c = 1:OMEData.SizeC
    try
        OMEData.WLEx{c} = round(omeMeta.getChannelExcitationWavelength(imageID, c-1).value().doubleValue());
        OMEData.WLEm{c} = round(omeMeta.getChannelEmissionWavelength(imageID, c-1).value().doubleValue());
    catch
        msg = 'Problem getting excitation and emission wavelengths. Set to zero.';
        warning(msg);
        OMEData.WLEx{c} = 0;
        OMEData.WLEm{c} = 0;
    end

    try
        OMEData.Channels{c} = char(omeMeta.getChannelName(imageID, c-1));
        OMEData.Dyes{c} = char(omeMeta.getChannelFluor(imageID,c-1));
    catch
        msg = 'No Metadata for current channel available.';
        warning(msg);
        OMEData.Channels{c} = 'na';
        OMEData.Dyes{c} = 'na';
    end

end

% close BioFormats Reader
reader.close()
```

5.4 Displaying the Data

5.4.1 AnalyzeSeries.m

This little program just cycles through all the series and calls an image analysis script. In case of this example it will be **CountObjectsSimple.m** (line 26). An example image for the segmentation is created for on random image out of the series.

```
1  % Analyze Series - Cycles through all existing series inside the data set
2  %
3  % Date: 13.10.2014
4  % Version: 1.0
5
6  function [numObjects, img2show, show] = AnalyzeSeries(image6d)
7
8  dims = size(image6d);
9  % series is 1st dimension !
10 series = dims(1);
11 % create random image number to shwo later on
12 show = randi([1 series]);
13 % add progress bar
14 h = waitbar(0,'Processing Data ...');
15
16 for tp=1: series
17
18     % update waitbar
19     wstr = {'Processed: ', num2str(tp), ' of ', num2str(series),'Frames' };
20     waitbar(tp / series, h, strjoin(wstr))
21     % get current image from series
22     img = image6d(tp, 1, 1, 1, :, :);
23     % analyze current image
24     if tp == show
25         result = CountObjectsSimple(img,true);
26         numObjects(tp) = result{1};
27         img2show = result{2};
28     else
29         result = CountObjectsSimple(img, false);
30         numObjects(tp) = result{1};
31     end
32 end
33
34 % close waitbar
35 close(h)
```

5.4.2 CountObjectsSimple.m

This is a simple example on how to detect and count objects using the image processing toolbox from MATLAB. It works fine for the sample data but is just a simple proof of principle.

```
% Simple Object Count inside a single frame
%
% Date: 13.10.2014
% Version: 1.0

function result = CountObjectsSimple(img, show)

% adapt to your needs ...
I = uint8(squeeze(img));
%I = uint16(squeeze(img));

% Create BW image from Otsu threshold and fill holes
th = graythresh(I);
bw = im2bw(I, th);
bw2 = imfill(bw,'holes');

% Remove all cells < XY pixels
bw4 = bwareaopen(bw2, 25);
bw4_perim = bwperim(bw4);
if show == true
    % creste image where object perimenter is visible
    overlay1 = imfuse(I, bw4_perim,'scaling', 'independent');
end

% get the number of objetcs
[L, numObjects] = bwlabel(bw4);

% Create overlay only once
if show == true
    mask = im2bw(L, 1);
    overlay2 = imfuse(I, mask);
end

result = {};
result{1} = numObjects;
if show == true
    result{2} = overlay1;
end
```

5.4.3 DisplayObjectNumbers.m

This M-file just displays the results in various ways.

```
% Display Object Numbers from Well
%
% Date: 13.10.2014
% Version: 1.0

function DisplayObjectNumbers(numObjects, segimage, show)

%create figure
figure('position', [100, 100, 1000, 600])  % create new figure with specified size

% 1st plot - Histogram
subplot(2,2,1)
hist(numObjects);
title('Distribution', 'FontSize', 14)
xlabel('Number of Objects', 'FontSize', 12)
ylabel('Occurence', 'FontSize', 12)
grid on

% 2nd plot - Follow Acquisition
subplot(2,2,2)
bar(numObjects, 1.0, 'r')
title('Follow Acquisition', 'FontSize', 14)
xlabel('Frame', 'FontSize', 12)
ylabel('Number', 'FontSize', 12)
grid on
axis([1 length(numObjects) 0 max(numObjects)*1.1])

% 3rd plot - Display Heatmap
subplot(2,2,3)
% Remark: This works for Comb-Style Acqusition only ...
well = reshape(numObjects, 8,12);
imagesc(well);
axis equal tight
set(gca,'xtick', [1:12]); % Well Columns
set(gca,'ytick', [1:8]);  % Well Rows
set(gca,'YTickLabel',{'A','B','C','D','E','F','G','H'});
colorbar
title('Objetcs per Well', 'FontSize', 14);

% 4th plot - Segmentation Example
subplot(2,2,4)
imagesc(segimage);
tstr = {'Frame : ', num2str(show), ' - Objects Detected : ', num2str(numObjects(show)) };
title(strjoin(tstr), 'FontSize', 14)
axis equal tight
xlabel('X [pixel]')
ylabel('Y [pixel]')
```

6 Disclaimer

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

Carl Zeiss Microscopy GmbH's ZEN software allows to connect to a the third party software MATLAB. Therefore Carl Zeiss Microscopy GmbH undertakes no warranty concerning MATLAB, makes no representation that MATLAB 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.