Contents

1 Introduction

This application note will explain how create a workflow using the ZEN-MATLAB connection. It illustrates the possibility to run MATLAB code from within a ZEN OAD macro.

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.

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.

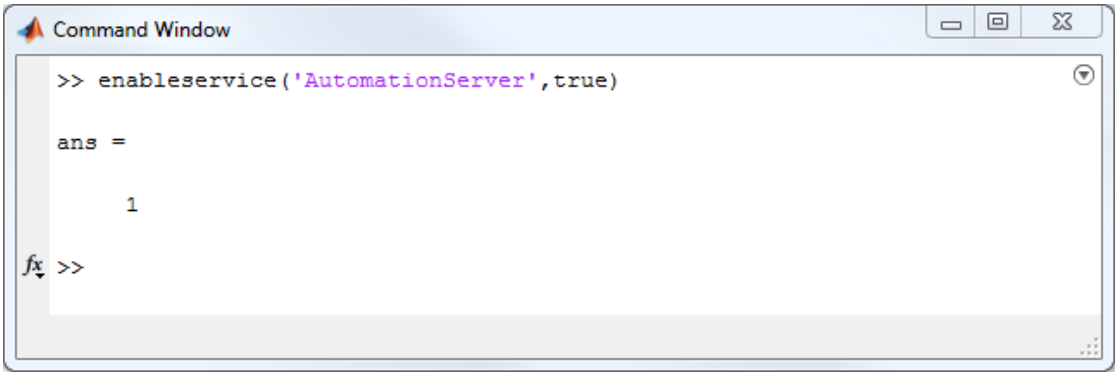


Figure 1: MATLAB - Enable Automation Server

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.

3 ZEN - Loading an CZI image

Just load a CZI image of your choice into ZEN. For this example the selected image has the following dimensions:.

- Timepoints: T = 1
- Z-Planes : Z = 22
- Channels : CH = 3
- Dimension Oder: XYCZT

Inside the ZEN Blue software this should similar tom this:

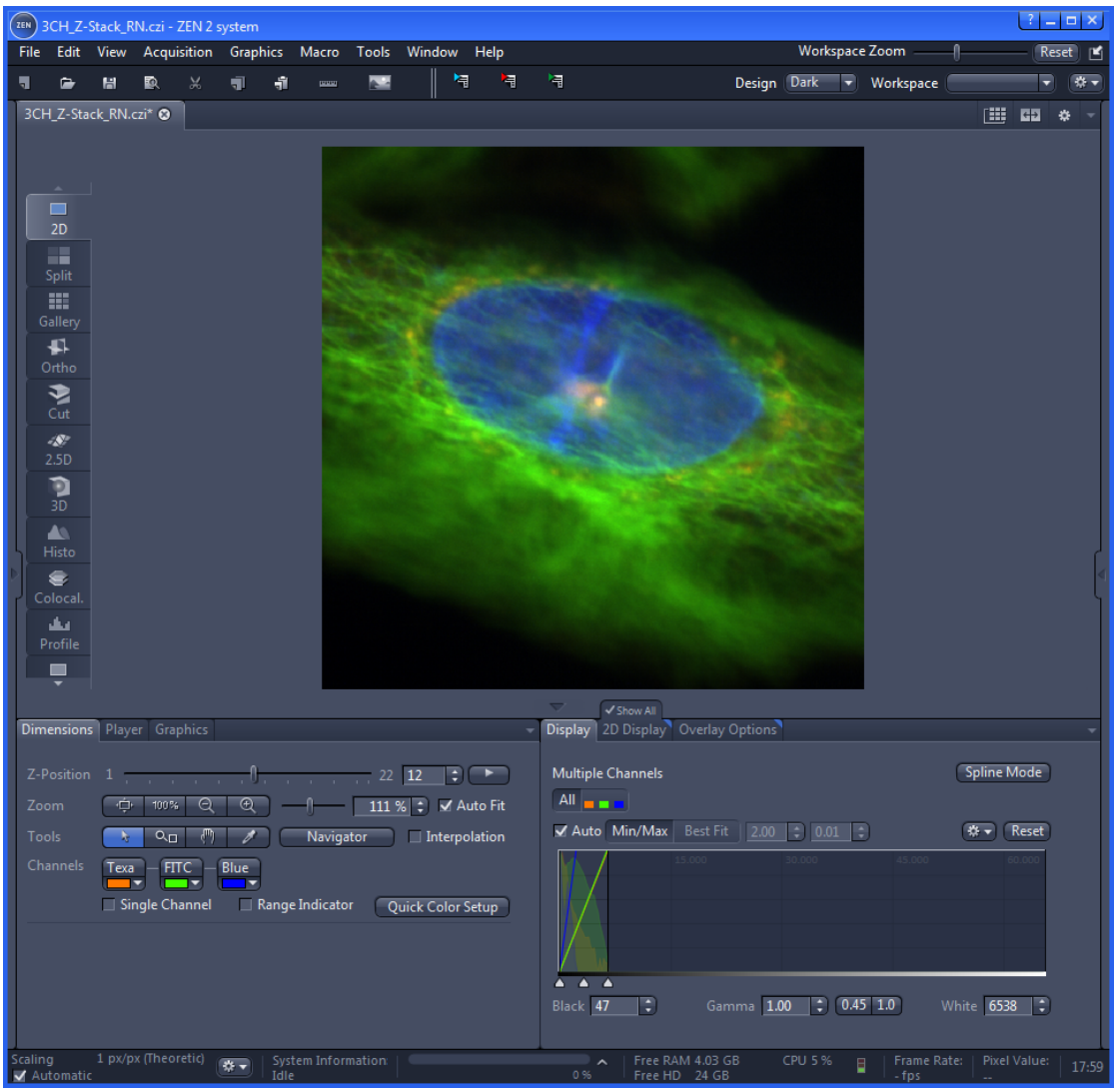


Figure 2: ZEN - The example image is loaded and displayed into ZEN

4 ZEN - Dialog to send the CZI to MATLAB via OAD

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 and vice versa.
This means you ncan either:

- Use OAD code directly within a M-File or from within the MATLAB command window
- Use MATLAB code directly within an OAD macro within ZEN

The following OAD script will show a very simple dialog that allows the user to select the dimension order and will send the active image to MATLAB using the second possibility mentioned above.

Important Remark: MATLAB must be already running and the Automation Server must be activated.

```
# Author: Sebastian Rhode
# Date: 26.03.2015
# Version: 1.2

# Send the current open image over to MATLAB using ReadImage6D.m --> has to be on the MATLAB path !!!
from System.Runtime.InteropServices import Marshal
from System.IO import Path, File, Directory, FileInfo
import os

CZIfiles_short = []
CZIdict = {}
# get all open documents
opendocs = Zen.Application.Documents
for doc in opendocs:
    image = Zen.Application.Documents.GetByName(doc.Name)
    if image.FileName.EndsWith('.czi'):
        # get the filename of the current document only when it ends with '.czi'
        CZIfiles_short.append(Path.GetFileName(image.FileName))
        CZIdict[Path.GetFileName(image.FileName)] = image.FileName

## Activate GUI
wd = ZenWindow()
wd.Initialize('Sent CZI to MATLAB',470,200,True,True)
## add components to dialog
wd.AddLabel('Sends selected CZI image to MATLAB.', '0', '0')
wd.AddLabel('Uses MATLAB wrapper (bfopen) for BioFormats.', '1', '0')
wd.AddDropDown('czi', 'Select CZI Image Data', CZIfiles_short, 0, '2','0')
## show the window
result=wd.Show()
## check, if Cancel button was clicked
if result.HasCanceled == True:
    sys.exit('Macro aborted with Cancel!')

## get the input values and store them
cziname = result.GetValue('czi')

try:
    MATLAB = Marshal.GetActiveObject('MATLAB.Application.8.5')
    print 'ZEN-MATLAB bridge is OK.'
    MLOK = True
except:
    print 'MATLAB not running'

if MLOK == True:
```

```
## get current active document
CZIfolder = os.path.dirname(CZIdict[cziname])
print 'Transfer: ', CZIdict[cziname]
## create MATLAB variables
MATLAB.execute("filename = '"+CZIdict[cziname]+"'")
MATLAB.execute("CZIimage = ReadImage6D(filename);")

print 'Done'
```

The OAD macro will open the following dialog:

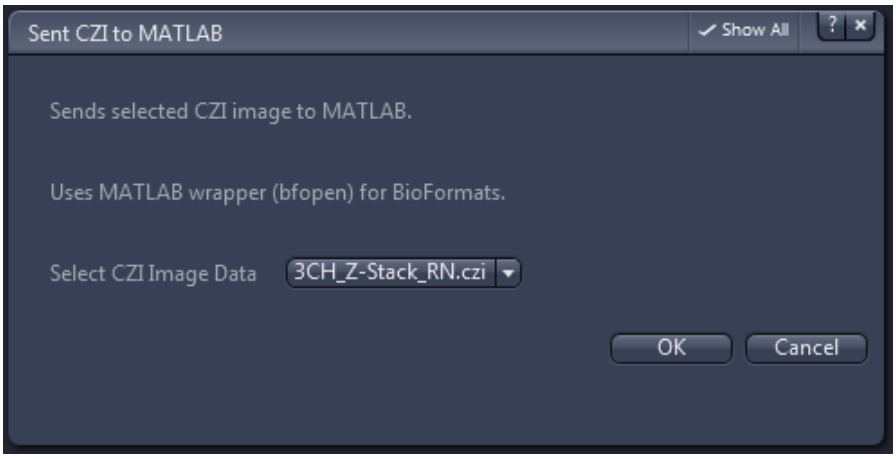


Figure 3: ZEN - Display the dialog to send the image to MATLAB

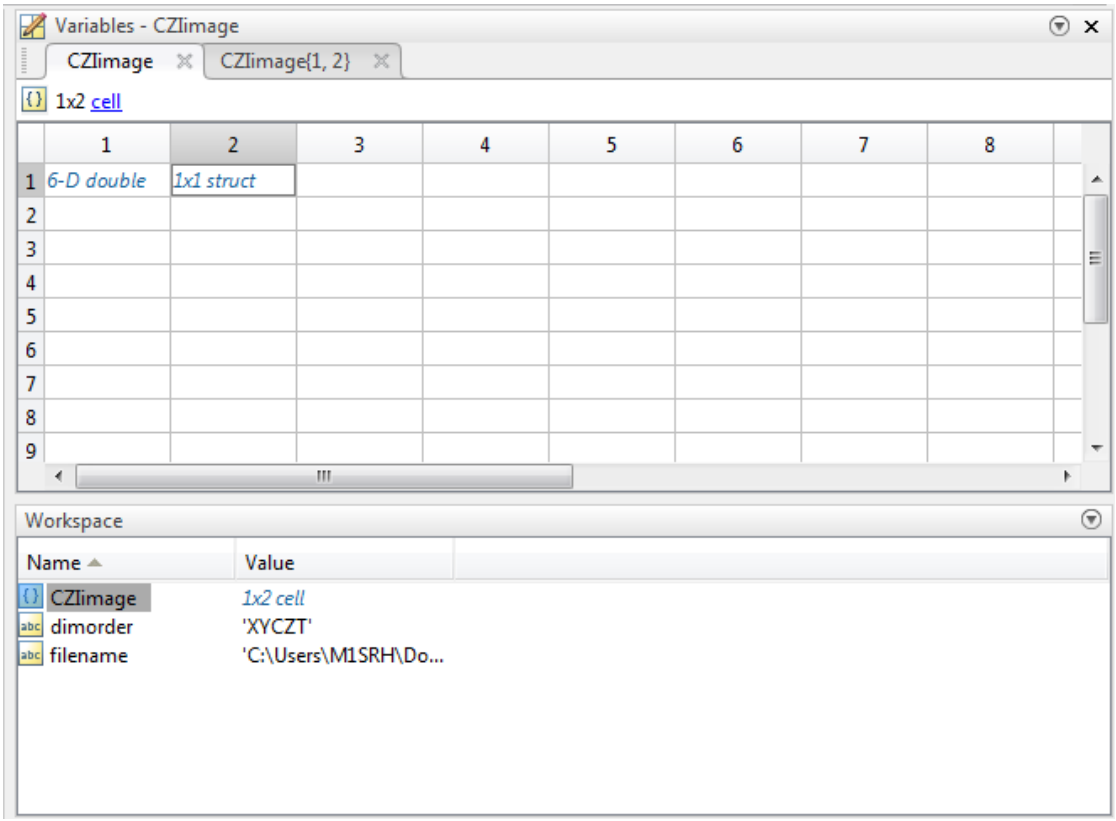
The DropDown box contains all currently open CZI image documents. All the user has to do is to select the image to be send.the correct dimension order and press **OK**. The dialog then calls a number of MATLAB commands and used BioFormats to read the CZI into an 6D image array and the corresponding metadata into MATLAB

5 ZEN-MATLAB at work

The OAD macro behind this little dialog will execute just three lines of MATLAB code:

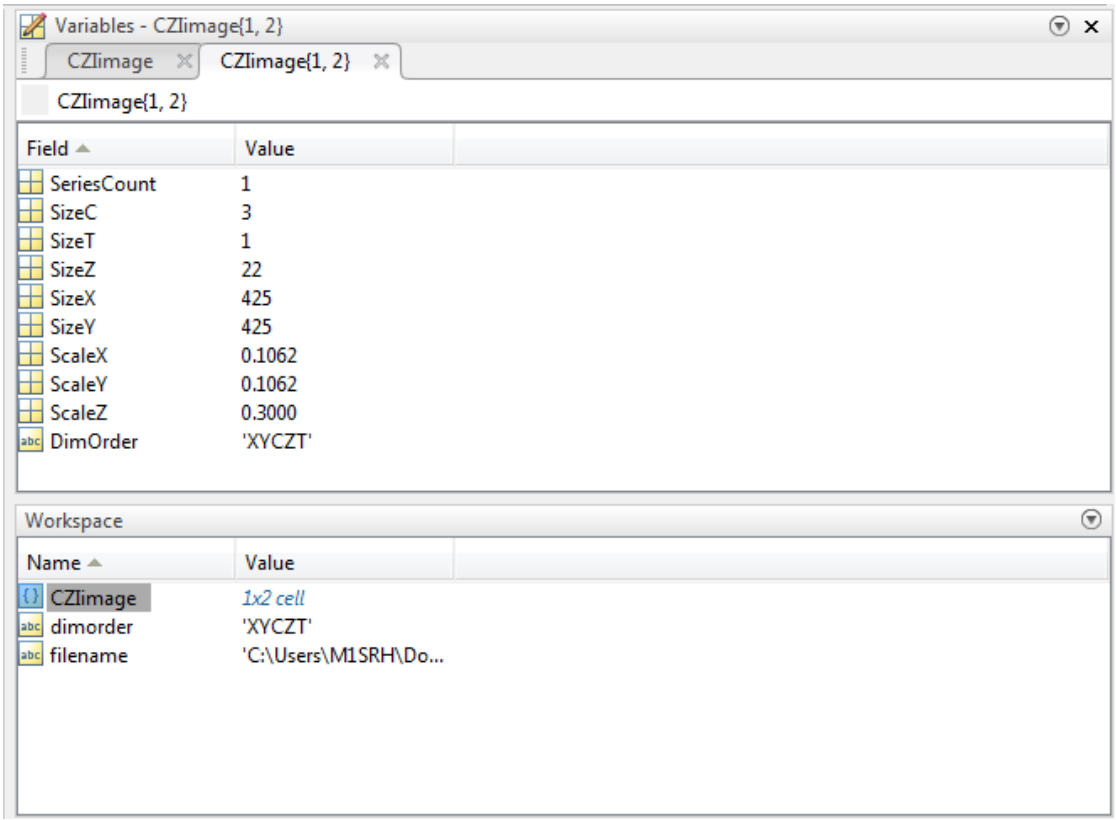
- filename = CZIfilename
- dimorder = DimOrder
- CZIimage = ReadImage6D(filename, dimorder)

Inside MATLAB the result will look like this:



As a result one will get a cell in MATLAB that contains two elements.

- a 6D matrix with the dimension [series, T, Z, C, X, X] depending on the dimension order
- a structure array containing metadata
  - **SeriesCount** = number of image series
  - **SizeC** = channels
  - **SizeT** = time points
  - **SizeZ** = focal planes
  - **SizeX** = pixels in X
  - **SizeY** = pixels in Y
  - **ScaleX** = scaling X in micron
  - **ScaleY** = scaling Y in micron
  - **ScaleZ** = spacing between two focal planes in micron
  - **DimOrder** = dimension order



5.1 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 **6.1**

5.2 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 **6.2**.

6 Appendix: MATLAB M-Files

6.1 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;

```

6.2 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()
```

7 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 example. By running this example you agree to this disclaimer.