materialise

innovators you can count on

AI Inference through Mimics Scripting

Contents

1 Content ..... **Error! Bookmark not defined.**
2 Setting up Mimics with Conda 2
3 Setting up Mimics with Python for scripting 2
4 AI Inference in Mimics 3

1 Prerequisites

This document specifies how you can perform inference from within Mimics through the Scripting package. The steps involved in the following section apply to Mimics 24 and Python 3.7.

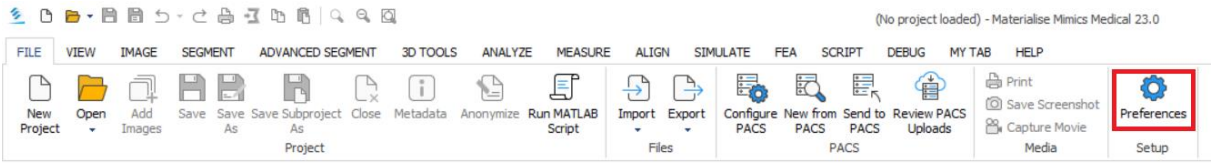
2 Setting up Mimics with Conda

At the moment Mimics scripting is not yet out of the box compatible with conda. However, at the moment we provide a workaround: through adding an extra line to your Python scripts. When adding the following lines at the beginning of your python scripts Mimics will be able to load all conda managed dependencies:

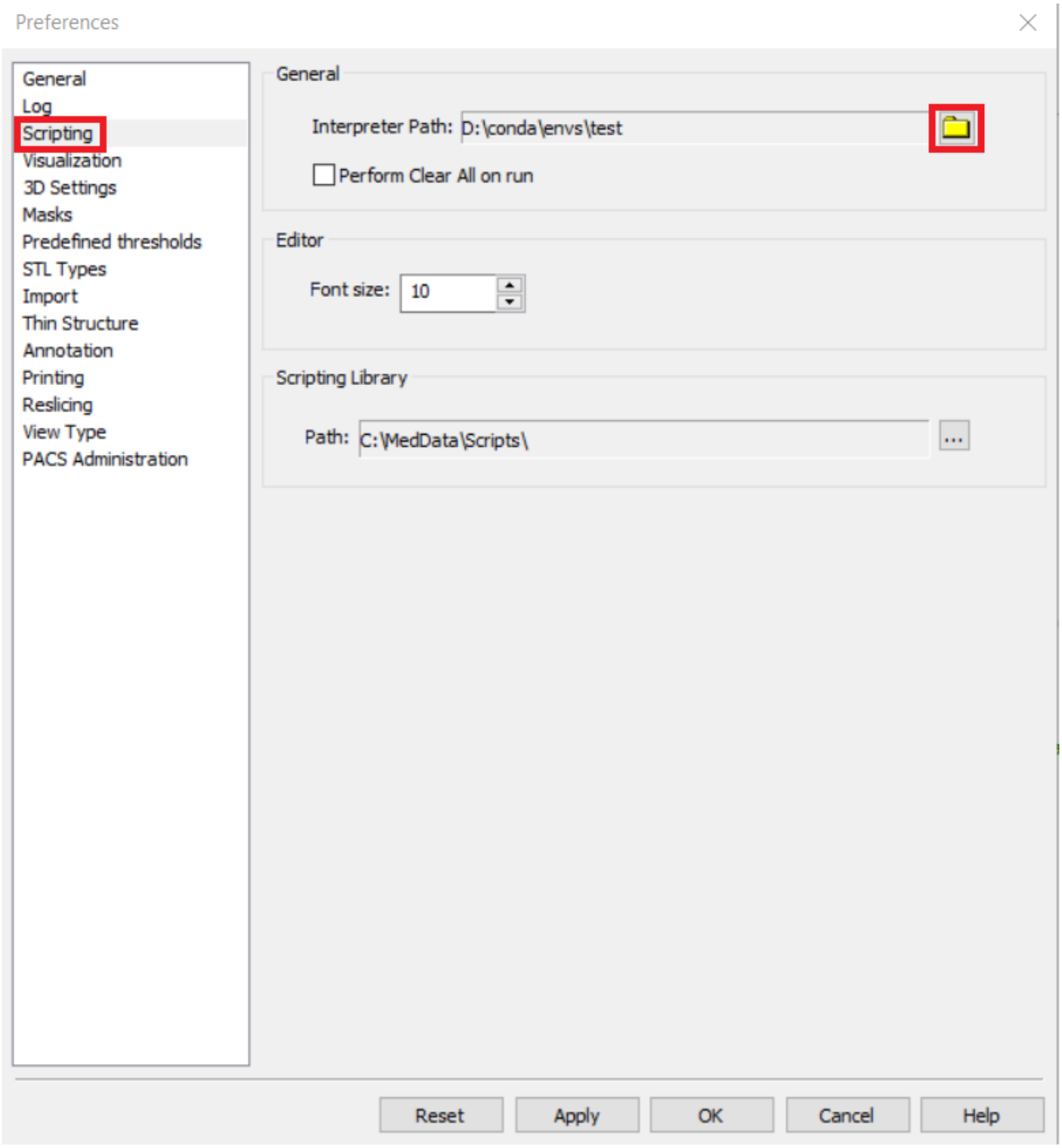
```
import os
python_interpreter_path = os.path.sep.join(os.__file__.split(os.path.sep)[:-2])
sub_folders = [r'Library\mingw-w64\bin', r'Library\usr\bin', r'Library\bin',
'Scripts', 'bin']
folders = [os.path.join(python_interpreter_path, sub) for sub in sub_folders]
path_addition = os.pathsep.join(folders)
os.environ['PATH'] += os.pathsep + path_addition
```

3 Setting up Mimics with Python for scripting

When using Mimics scripting it is important to specify the python interpreter location in Mimics' preferences. To do this click the Preferences button on the *File* Tab in the Mimics Ribbon:



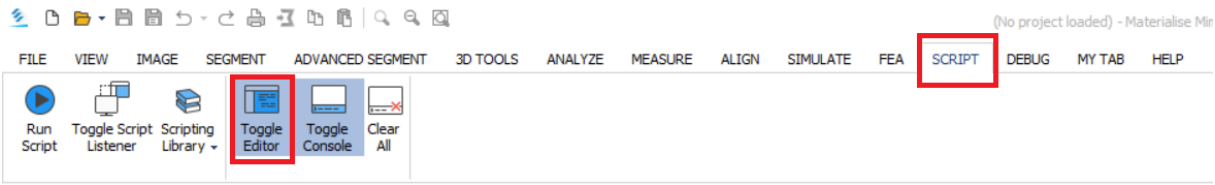
Next click on the scripting preferences on the left. Next you need to browse to the folder where your python interpreter is installed. Finally click on 'Apply' and next click on 'OK':



Now Mimics is ready for your AI scripting projects.

4 AI Inference in Mimics

In the Mimics ribbon click on the *Scripting* Tab, followed by clicking on the *Toggle Editor* Button:



This will open a new window in which you can create a Python script and execute the script. We have added an example script below. In Lines 1 to 7 we have added the code to add the conda depencies to the system path in order for Mimics to find them. Next in Lines 9 to 12 we do the normal imports we need for our inference script, like: numpy, tensorflow, or our own python packages.

```
1  import os
2  import sys
3  python_interpreter_path = sys.executable
4  sub_folders = [r'Library\mingw-w64\bin', r'Library\usr\bin', r'Library\bin', 'Scripts', 'bin']
5  folders = [os.path.join(python_interpreter_path, sub) for sub in sub_folders]
6  path_addition = os.pathsep.join(folders)
7  os.environ['PATH'] += os.pathsep + path_addition
8
9  from knee_segmentation import preprocess_image, inference
10 import tensorflow as tf
11 import mimics
12 import numpy
13
14
15
16 # Load Mimics project
17 mimics.file.open_project(filename=r'H:\mimics_data\knee_scan.mcs')
18
19 # Extract Image buffer, spacing, origin
20 image = mimics.data.images.get_active()
21 voxel_buffer = image.get_voxel_buffer()
22 spacing = [image.logical_slice_distance, image.pixel_size, image.pixel_size]
23 origin = image.get_voxel_center((0,0,0))
24 size = numpy.array(voxel_buffer.shape)
25 directions = numpy.eye(3)
26
27 # Perform preprocessing
28 image, new_spacing, new_origin, new_directions  = preprocess_image(voxel_buffer, size, spacing, origin, directions)
29
30
31 # Inference
32 masks = inference(image, new_spacing, new_origin, new_directions, size)
33
34 # Add masks back to mimics
35 mask_names = ['femur_cartilage', 'femur_bone', 'tibia_cartilage', 'tibia_bone']
36 mimics_masks = []
37 for i_mask in range(len(mask_names)):
38     mask = mimics.segment.create_mask(masks[i_mask,...].astype(numpy.uint8))
39     mask.name = mask_names[i_mask]
40     mimics_masks.append(mask)
41
42 # Post-processing: booleans, split_mask, ...
43 merged_femur = mimics.segment.boolean_operations(mimics_masks[0], mimics_masks[1],operation='Unite')
44
45 # Create meshes from these masks
46 for mask in mimics_masks:
47     mesh = mimics.segment.calculate_part(mask)
48     mesh.name = mask.name
49
```

On line 17 we open a specific Mimics file we would like to segment. On Lines 20 to 25 we extract the voxel buffer, its origin, spacing and size, from the active image in the Mimics project. On Line 28 we apply our preprocessing to the image: for example image normalization, rescaling, metal artifact removal, etc. Next on line 32 we apply our neural network code to the voxel buffer, in order to obtain the masks as *numpy.ndarray. O*n lines 35 to 40 we transform these numpy arrays to Mimics masks so we can visualize them in the GUI. Finally, on line 43 we apply some final post processing by combining two masks into one using a Boolean operation. Other post processing functionality present in mimics could also be called from here on. Below you can find a small, non-exhaustive list of some possible post-processing functions:

mimics.segment.boolean_operations(mask_a, mask_b, operation)

mimics.segment.split_mask(selection, regions)

mimics.segment.keep_largest(mask)

mimics.segment.morphology_operations(input_mask, operation, number_of_pixels, connectivity, target_mask_name, limited_to_mask)

mimics.segment.smooth_mask(mask)

mimics.segment.calculate_part(mask, quality)