PoreSizeExcel ImageJ plugin – User manual

1. Plugin summary

PoreSizeExcel is a Java-based ImageJ plugin that enables running macros from an Excel file. Our primary aim was to evaluate pore size in microporous media, but it can be used for many other purposes.

This plugin provides two independent, but linked parts: an Excel-based macro runner and an adaption of Beat Münch's pore size distribution algorithm[1] to be directly used with the Excel macro runner.

The Excel-based macro runner functionality allows to read an Excel file containing a list of images and apply user-defined macros to these images and collect the results. There is also limited parsing capacity regarding the macro evaluation such that values from different columns in Excel can be used to customize the macros.

The pore size evaluation part is an interface to Beat Münch's xlib plugin that enables pore size analysis guided by a master Excel file by means of the macro runner functionality.

Our use for this plugin is to evaluate mean pore size and pore size distribution in microporous hydrogels scaffolds in an automated fashion, directed by Excel files tabulating the confocal images acquired for the purpose. However, it became clear that especially the Excel macro-runner part can be used for many other purposes, and is in part for this reason that we share this plugin here with the Open Source community.

2. Content

1. PLUGIN SUMMARY 1

2. CONTENT 1

3. INSTALLATION 2

3.1. PoreSizeExcel plugin 2

3.1.1. Download 2

3.1.2. ImageJ installation 3

3.2. Dependencies 4

3.2.1. Libraries 4

3.2.2. Other plugins 4

3.3. Testing installation 5

4. EXCEL MACRO RUNNER: USAGE EXAMPLES 6

4.1. The most elementary scenario: Running a macro on a list of files specified by Excel 7

4.1.1. Scenario 7

4.1.2. Execution synopsis 7

4.1.3. Pre-requisite: Excel file with image file listing 8

4.1.4. Plugin launch 8

4.1.5. Excel file loading *9*
4.1.6. Plugin dialog *9*
4.1.7. Running the plugin *11*
4.1.8. Result collection *11*
4.2. IMAGE FILES IN DIFFERENT FOLDERS 12
4.3. ADDITIONAL INFORMATION IN EXCEL 13
4.4. SIMPLE MACROS IN EXCEL 14
4.5. MACROS WITH MULTIPLE OUTPUT LINES 16
4.6. EXCEL COLUMN VALUES IN MACROS 17
4.6.1. Scenario *17*
4.6.2. Basic macro code *17*
4.6.3. Column substitution *17*
4.6.4. Excel file: test_column_substitution.xlsx *18*
4.6.5. Output: Template and actual macro *18*
4.6.6. Column substitution and macro specification within the plugin *18*
4.7. SAVING OF MODIFIED IMAGE FILES 19
4.7.1. Scenario *19*
4.7.2. Basic macro *19*
4.7.3. Absolute file path *19*
4.7.4. Excel file: test_save_file.xlsx *20*
4.7.5. Running the plugin *20*
5. PORE SIZE EVALUATION 21
5.1. PORE SIZE EVALUATION BY BEAT MÜNCH'S XLIB PLUGIN 21
5.2. PORESIZEEXCEL INTERFACE 21
5.3. INTEGRATING PORE SIZE EVALUATION INTO A AN EXCEL-IMAGEJ MACRO WORKFLOW 21
6. SPECIAL TASKS 21
6.1. PROGRAMMATICALLY DEFINING FILE LOCATIONS FOR DIFFERENT COLLABORATORS 21
7. BIBLIOGRAPHY 22

3. Installation

3.1. PoreSizeExcel plugin

3.1.1. Download

To install PoreSizeExcel, obtain the latest binary release from github:

https://github.com/tbgitoo/PoreSizeExcel/releases/download/v0.9/PoreSizeExcel_.jar

If you are interested in older releases, these are maintained in Github as well, the overview over the different releases can be found at:

https://github.com/tbgitoo/PoreSizeExcel/releases

For each release, the relevant binary ".jar" file can be found under "Assets".

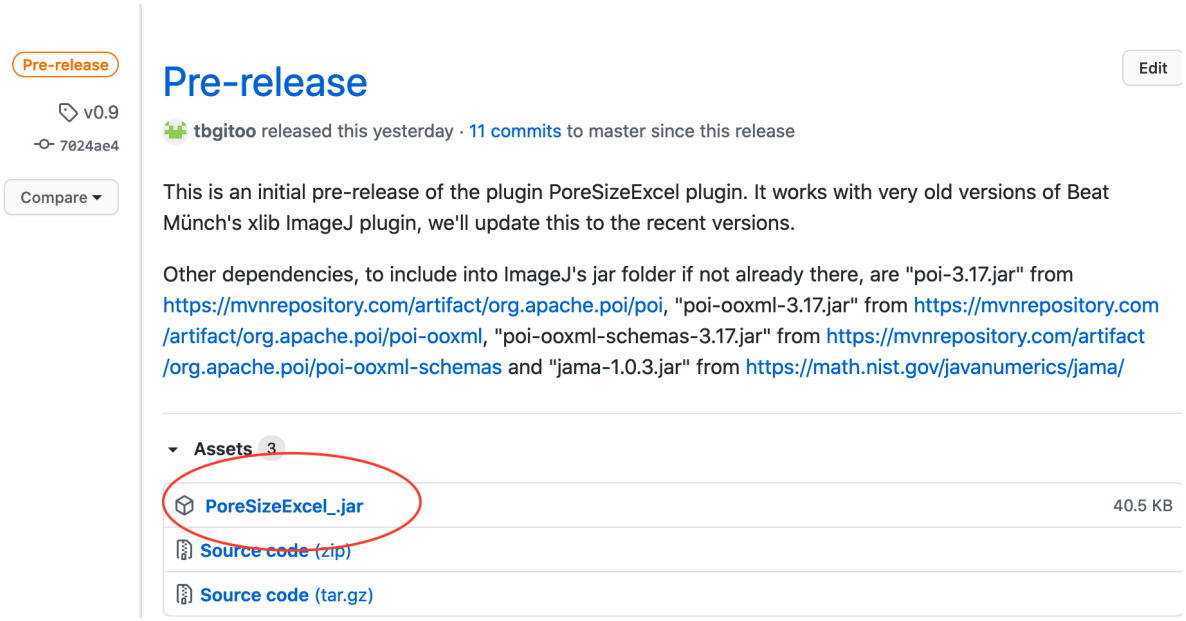


Figure 1. Obtaining the binary jar file. We provide the binary jar (Java archive) files with each release. The release overview can be accessed at https://github.com/tbgitoo/PoreSizeExcel/releases. For each release, the binary jar required for plugin installation can be accessed under the Assets tab, as shown.

3.1.2. ImageJ installation

The PoreSizeExcel plugin can be installed in ImageJ just like any other ImageJ plugin. At present, installation is manual, but our intention is to add an update site shortly.

Manual installation is done by dropping the downloaded PoreSizeExcel_.jar file into the plugins folder of your ImageJ installation, as shown in Figure 2. In general, the plugin will only partially work out of the box, since it depends on additional libraries that may, or also may not be installed due to the presence of other plugins. Hence, it is mandatory to not only install the PoreSizeExcel_.jar file, but also follow the instructions in section 3.2 for the dependencies.

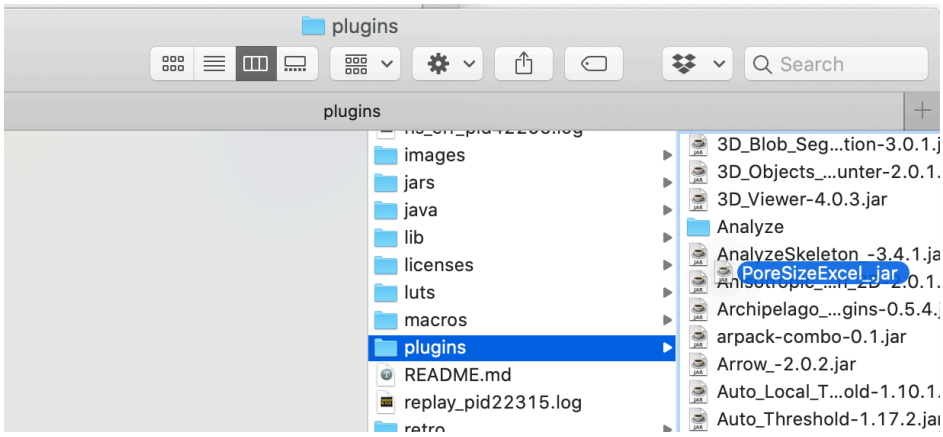


Figure 2. Manual installation of the PoreSizeExcel_.jar plugin, by dropping it into the plugins folder of the local ImageJ installation.

3.2. Dependencies

3.2.1. Libraries

The PoreSizeExcel plugin depends on the following additional libraries:

A) "poi-3.17.jar" or higher: Java-Excel interface, Apache library, from
https://mvnrepository.com/artifact/org.apache.poi/poi
B) "poi-ooxml-3.17.jar" or higher: Java-Excel interface, Apache library, from
https://mvnrepository.com/artifact/org.apache.poi/poi-ooxml
C) "poi-ooxml-schemas-3.17.jar" or higher: Java-Excel interface, Apache library, from
https://mvnrepository.com/artifact/org.apache.poi/poi-ooxml-schemas
D) "jama-1.0.3.jar": Generic Java math library, from
https://math.nist.gov/javanumerics/jama

These library jar files need to be downloaded and installed into the “jars” folder of your ImageJ installation (as shown in Figure 3). In many cases, these dependencies are fully or partially satisfied due to the presence of other plugins which also depend on them. In this case, it is not necessary to install them.

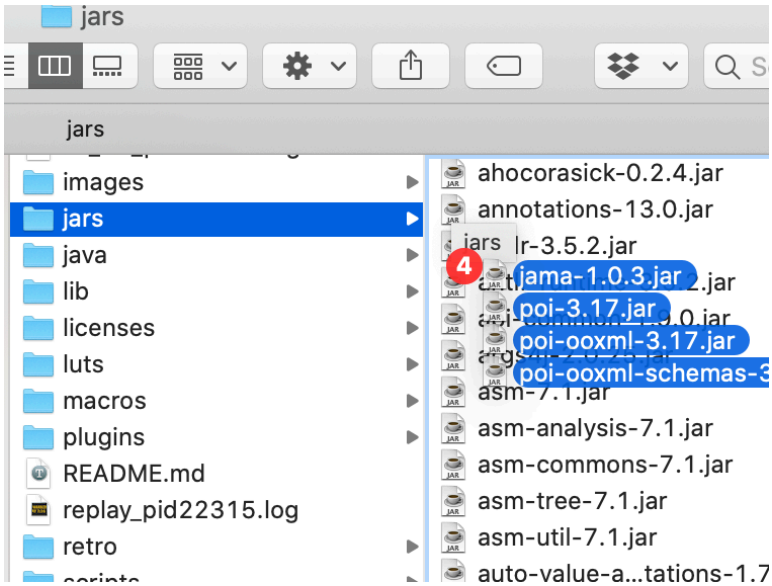


Figure 3. Installing the library dependencies.

3.2.2. Other plugins

Specifically for the pore size estimation, the PoreSizeExcel plugin depends on Beat Münch’s xlib plugin (see https://imagej.net/Xlib). This plugin is regularly updated, the latest version can be downloaded from https://sites.imagej.net/Xlib/plugins/. As a caveat, the files downloaded from https://sites.imagej.net/Xlib/plugins/ typically have file names of the type xlib_.jar-20191001174205, we find it necessary to remove the numerical part (so that the the file name reads “xlib_.jar”) since otherwise, ImageJ won’t recognize it as a Java Archive (jar) file.

The “xlib_.jar” file thus obtained needs to put into the plugin folder of your local ImageJ installation (as shown in Figure 4).

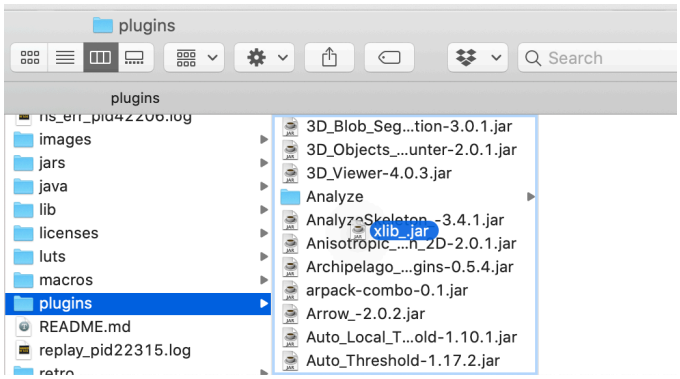


Figure 4. Installing xlib_.jar.

3.3. Testing installation

To test whether the installation was successful, restart your ImageJ or Fiji. Then, check for the presence of the “Beat” entry for Beat Münch’s xlib plugin in the plugins menu as shown in Figure 5.

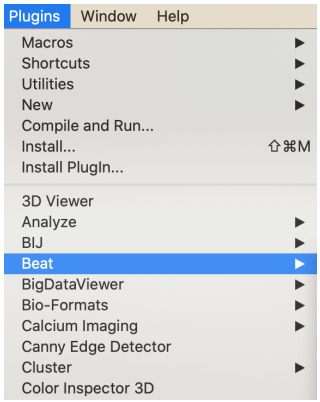


Figure 5. Succesful installation of Beat Münch’s xlib plugin is indicated by the presence of the menu point “Beat” in the plugins menu of ImageJ.

Next, check for the presence for the “Microniche PoreSizeExcel” entry, also in the Plugins menu in ImageJ.

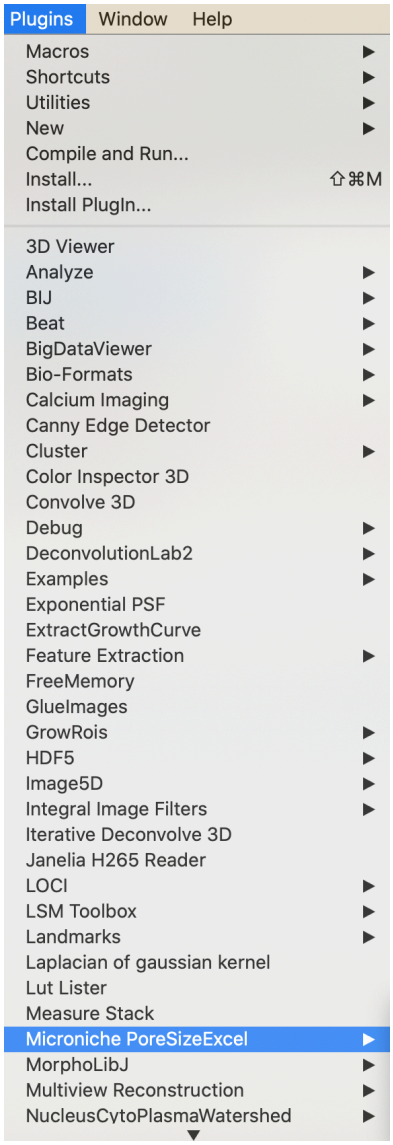


Figure 6. Succesfull installation of the PoreSizeExcel plugin as indicated by the presence of the menu point "Microniche PoreSizeExcel" in the Plugins menu in ImageJ (or Fiji).

4. Excel macro runner: usage examples

The example images and Excel files needed to run these examples can be found in the Github repository, at https://github.com/tbgitoo/PoreSizeExcel/tree/master/examples. See Figure 7. If you want to run these examples as a tutorial or for plugin testing, download these files locally, conserving the relative folder structures (i.e. subfolders folderA and folderB, as in Figure 7).

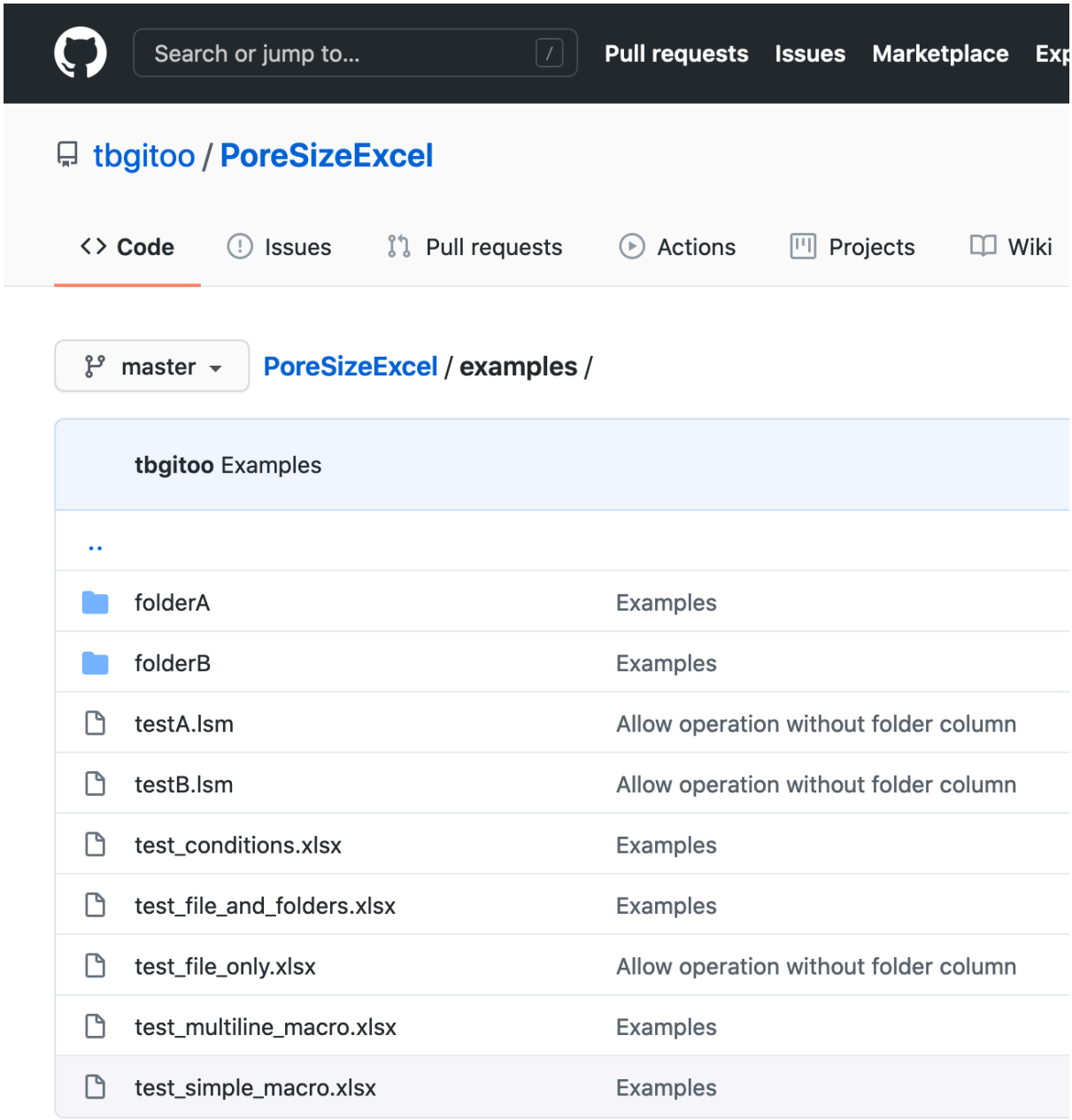


Figure 7. Files required for reproducing the examples

4.1. The most elementary scenario: Running a macro on a list of files specified by Excel

4.1.1. Scenario

In a very simple application, an Excel spreadsheet is used to tabulate a series of image files to be analyzed, and an identical macro is applied to each one of the images, with collection of possible text or numerical output in a common table.

Such a scenario is implemented by the “test_file_only.xlsx” Excel file, and the two image files “testA.lsm” and “testB.lsm” located in the same folder as “test_file_only.xlsx”.

4.1.2. Execution synopsis

In this simplistic scenario, the PoreSizeExcel plugin reads an Excel file. It applies a user-specified macro to each of the images listed in the Excel file, and collects the results in a common table.

4.1.3. Pre-requisite: Excel file with image file listing

A pre-requisite for running ImageJ macros using Excel files to direct the choice of files to be analyzed is to draft an Excel file providing at least the names of the files to be handled. In this most prototypical scenario, this is exemplified by the “test_file_only.xlsx”. This spreadsheet contains a single sheet (“Sheet1”), and data only in a single column, labelled “File”. The data entries correspond to the files to be analyzed, here, “testA.lsm” and “testB.lsm”. This is shown in Figure 8. Although we conventionally use “File” as a column header, this is not necessary, it could be “file path” or in fact anything else. However, the plugin does expect column headers in the first row, and also expects the table to start in cell A1.

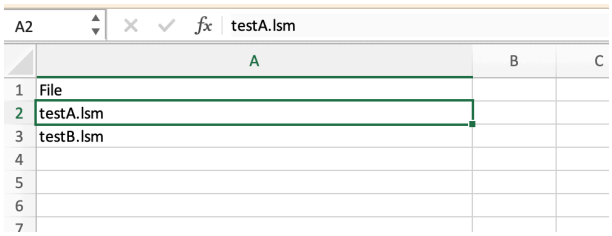


Figure 8. Contents of “ test_file_only.xlsx”.

4.1.4. Plugin launch

Assuming correct installation of the PoreSizeExcel plugin, the Excel batch processing can be launched in ImageJ (or equivalently, Fiji) by choosing Plugins > Microniche PoreSizeExcel > Batch Process with Excel File as shown in Figure 9.

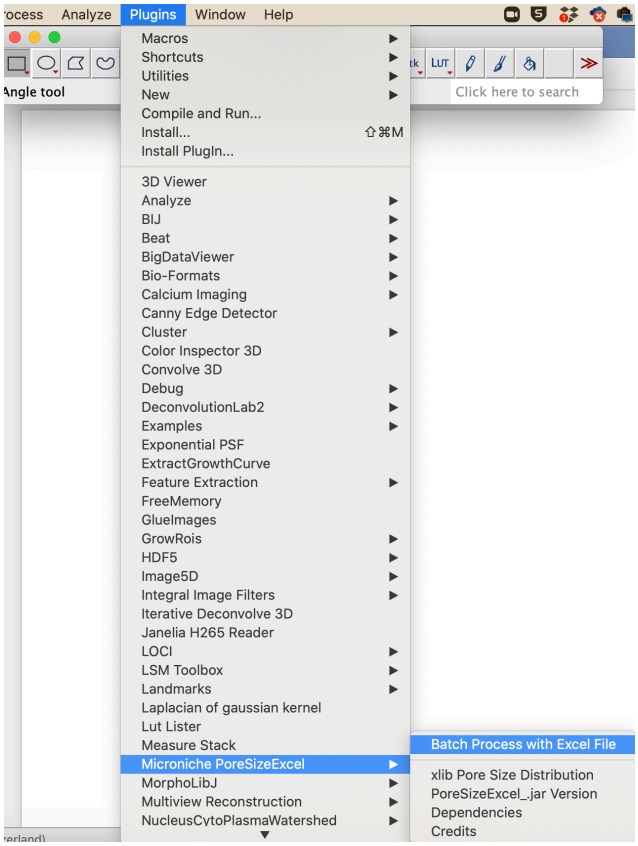


Figure 9. Plugin launch

4.1.5. Excel file loading

The PoreSizeExcel plugin next allows to select the Excel file for use. This is a standard file selection dialog, as shown in Figure 10. Select the appropriate Excel file, here "test_file_only.xlsx"

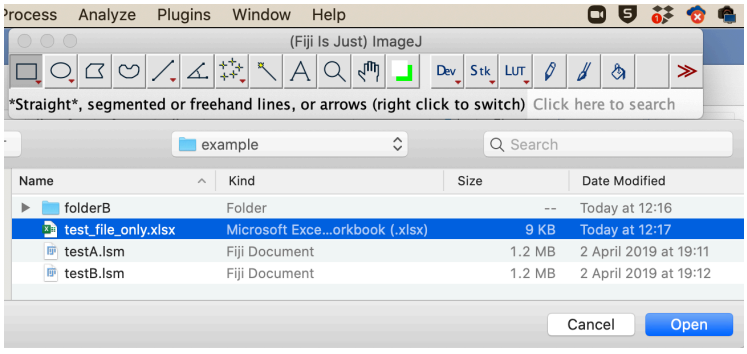


Figure 10. Excel file selection

In "test_file_only.xlsx", there is only a single Sheet, and so the PoreSizeExcel plugin will directly proceed to the main plugin dialog, otherwise, a sheet selection dialog would be open (see more complex scenarios).

4.1.6. Plugin dialog

The PoreSizeExcel plugin next provides the main plugin options dialog, shown in Figure 11.

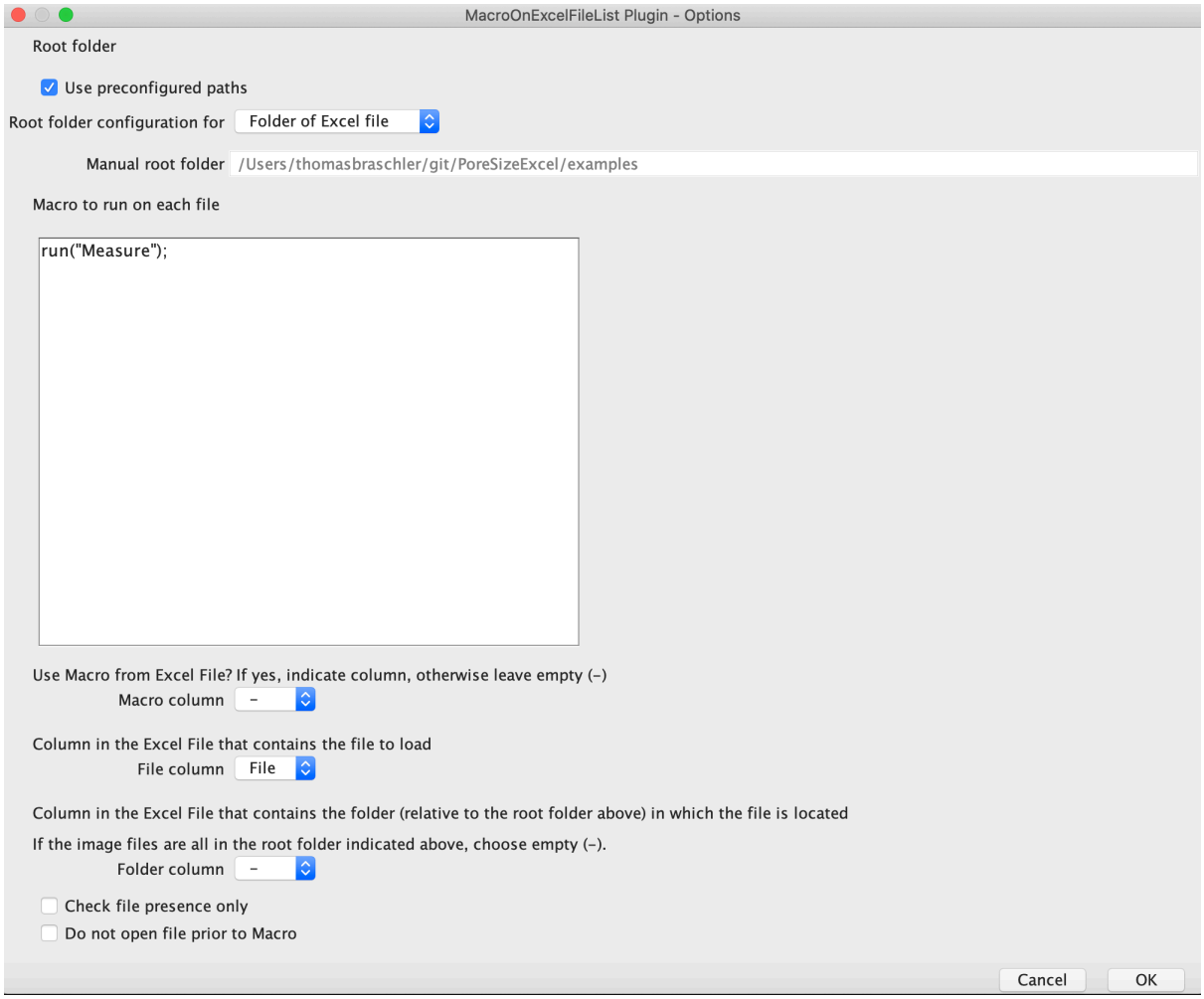


Figure 11. Plugin dialog

In the main plugin dialog, several options are available. First, the root folder must be configured. This is the folder relative to which images are located. The plugin offers either standard options (by default, the folder of the Excel file, but also programmatically configurable standard locations), or fully manual entry of the path. In Windows, the entry of the paths can be tricky. Either on your system, forward slashes are accepted, or you need to use backslashes ("\"), but then they must be doubled in order not to count as escape sequences. So it would be something like C:\\\\some\\folder.

Next, the plugin offers to execute macros on the image files listed in the Excel file. In the simplest configuration, an identical macro (here: run("Measure");) is executed on each image file. In more sophisticated scenarios (see below), it is also possible to execute specific macros on each image, for example if the file name needs to be used explicitly; this happens via the Excel file and is also explained below. Here, no macro information is taken from Excel, and so in the "Macro Column" choice field, "-" is selected.

In order to know which images need to be treated, the PoreSizeExcel plugin reads the Excel file. In the "File column" choice field, the column containing the file names can be selected.

Optionally, it is also possible to read the folder localization from the Excel as well. This is useful when the images resides in different subfolders, but is not needed for this simple example, and so no folder column ("-") is selected in the choice field.

For debugging purposes, it is possible to only check for the file presence, without running the actual macro. In this case, the checkbox "Check file presence only" should be checked.

Finally, the basic idea is that the PoreSizeExcel plugin opens the image, to then apply the macro. However, sometimes the image opening is more involved and needs to be handled as part of the macro. In this specific case, one would have to check the box "Do not open prior to Macro". However, this is not necessary for the simple test scenario here.

4.1.7. Running the plugin

After hitting "OK" in the plugin dialog, the PoreSizeExcel plugin runs through the list of files provided in the Excel file. Each file is opened. This happens by default using the ImageJ "open" macro command (i.e. box "Do not open prior to Macro" unchecked in Figure 11). Then, the macro is applied to the image. In the scenario here, this is the macro specified in the "Macro to run on each file" textbox (see Figure 11) textbox. In the concrete example is the measure command: run("Measure"). After execution of the macro, the image window is automatically closed.

4.1.8. Result collection

The PoreSizeExcel plugin keeps track of the macros executed and reports them as an ImageJ result table (Figure 12). At the very least, this table contains a copy of the columns in the original Excel file (i.e., the "File" column in Figure 12) as well as the actual macro that was run for each image (i.e. the "Actual macro" column in Figure 12). The actual macro comprises usually an "open" command, the user-specified macro, followed by a "close" command.

In addition to these minimal columns, the PoreSizeExcel plugin reports output generated during the user-defined macro. Indeed, some ImageJ commands, such as the run("Measure") command yield text output in the form of an ImageJ Results table, and this information is collected by the PoreSizePlugin. With the default measurement settings used here, this additional information corresponds to the area and mean grey value. These values are reported as shown in Figure 12 for each image. Hence, in the simplistic scenario presented here, user-defined characteristics of each image as specified in the Excel file are read and reported along with the macro used in an overview results table.

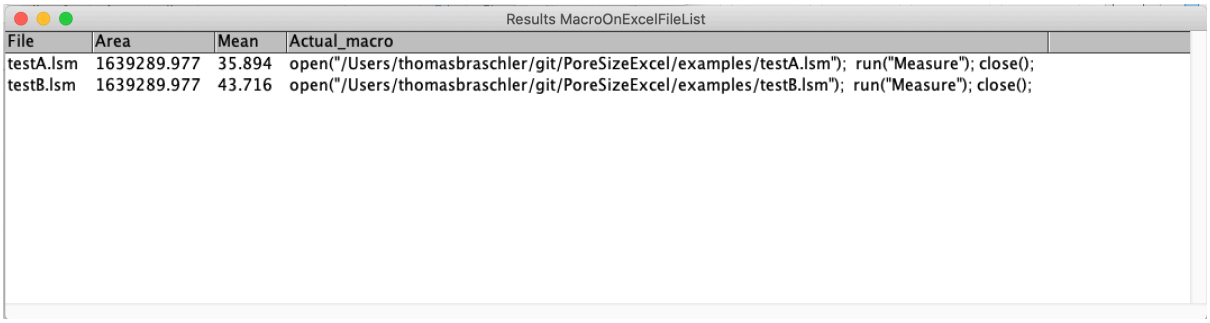


Figure 12. PoreSizeExcel output.

4.2. Image files in different folders

Especially with different experimental conditions, it happens frequently that the images are stored in different folders. One possibility is simply to adapt the file path in the Excel file to reflect the different localization, as in indicating “folderX/fileY.lsm” instead of “fileY.lsm”.

It is however often more convenient to separate file and folder information. In this case, the Excel file needs to have an additional column, specifying the folder in which the images reside, as shown in Figure 13. Empty cells in the Folder column signify that the image is located at the intended root folder of evaluation. With minor changes as compared to the simplistic scenario in section 4.1, the plugin evaluation remains the same. The few applicable changes are noted below.

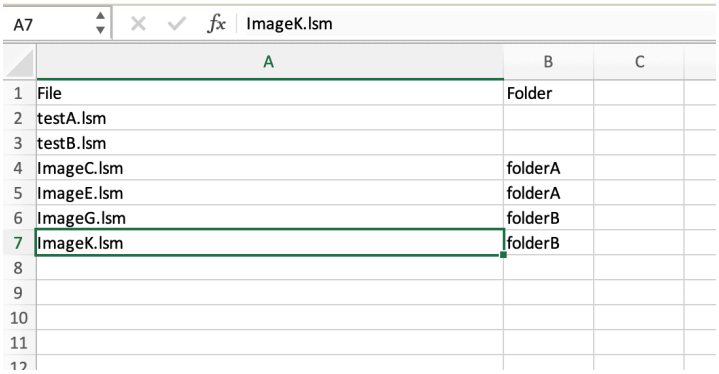


Figure 13. Excel file for images localized in different folders

In the plugin dialog (c.f. Figure 11), it is then necessary to select the Folder column for the file localization (as shown in detail in Figure 14).

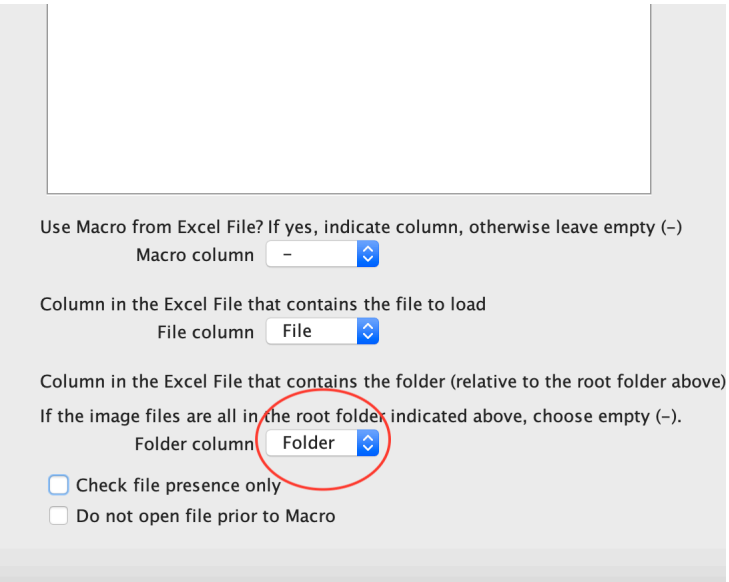


Figure 14. Selection of the folder column (red circle)

The PoreSizeExcel plugin will report all the columns of the Excel file in the results table, and so in the example, there will be the Folder column in addition to the result columns already present in Figure 12. This is shown in

Results MacroOnExcelFileList

File	Folder	Area	Mean	Actual_macro
testA.lsm		1639289.977	35.894	open("/Users/thomasbraschler/git/PoreSizeExcel/
testB.lsm		1639289.977	43.716	open("/Users/thomasbraschler/git/PoreSizeExcel/
ImageC.lsm	folderA	1639289.977	34.378	open("/Users/thomasbraschler/git/PoreSizeExcel/
ImageE.lsm	folderA	1639289.977	42.547	open("/Users/thomasbraschler/git/PoreSizeExcel/
ImageG.lsm	folderB	1639289.977	33.261	open("/Users/thomasbraschler/git/PoreSizeExcel/
ImageK.lsm	folderB	1639289.977	47.078	open("/Users/thomasbraschler/git/PoreSizeExcel/

Figure 15. Output with folder

4.3. Additional information in Excel

The PoreSizeExcel plugin copies all the information in the original Excel file to the results table. It is therefore possible to directly associate experimental conditions with the results. This is illustrated with the master file “test_conditions.xlsx “ (Figure 16). Running the example file “test_conditions.xlsx” with the Folder column as in section 4.2, the output should automatically include the additional columns (Figure 17).

D7 | 10

	A	B	C	D	E
1	File	Folder	Reactant1_mg/mL	Reactant2_mg/mL	
2	testA.lsm		0	0	
3	testB.lsm		0	0	
4	ImageC.lsm	folderA	10	0	
5	ImageE.lsm	folderA	10	0	
6	ImageG.lsm	folderB	0	10	
7	ImageK.lsm	folderB	0	10	
8					
9					
10					
11					
12					

Figure 16. Excel file with additional columns.

Results MacroOnExcelFileList

File	Folder	Reactant1_mg/mL	Reactant2_mg/mL	Area	Mean	Actual_ma
testA.lsm		0.0	0.0	1639289.977	35.894	open("/Us
testB.lsm		0.0	0.0	1639289.977	43.716	open("/Us
ImageC.lsm	folderA	10.0	0.0	1639289.977	34.378	open("/Us
ImageE.lsm	folderA	10.0	0.0	1639289.977	42.547	open("/Us
ImageG.lsm	folderB	0.0	10.0	1639289.977	33.261	open("/Us
ImageK.lsm	folderB	0.0	10.0	1639289.977	47.078	open("/Us

Figure 17. Output with additional columns.

4.4. Simple macros in Excel

Alternatively to running a macro defined in the main text field of the plugin dialog (cf. Figure 11), it is also possible to run ImageJ macros defined in the Excel file. For this, one typically has a Macro column in the Excel file, as shown in the example file “test_simple_macro.xlsx” (Figure 18). The macro can be different for each image, also as shown in Figure 18. Of note, it is usually easiest to obtain the correct syntax of the macros by recording them directly in ImageJ (i.e. Plugins > Macros > Record…).

AutoSave OFF | test_simple_macro — Saved to my Mac

Home | Insert | Draw | Page Layout | Formulas | Data | Review | View | Tell me

Paste | Calibri (Body) | 11 | General | Conditional Formatting | Format as Table | Cell Styles

B2 | run("Set Measurements...", "mean redirect=None decimal=3"); run("Measure");

	A	B	C	D	E	F	G	H
1	File	Macro						
2	testA.lsm	run("Set Measurements...", "mean redirect=None decimal=3"); run("Measure");						
3	testB.lsm	run("Add...", "value=100"); run("Measure");						
4								

Figure 18. Excel file for running macros

In the plugin dialog, it is necessary to specify the column containing the Macro, as shown in Figure 19.

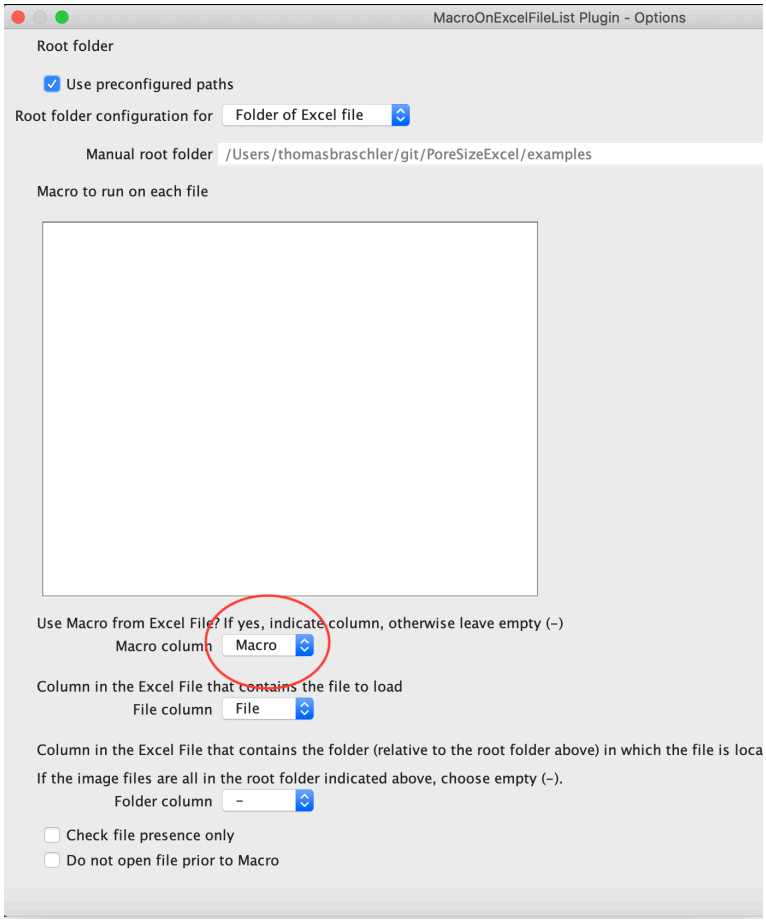
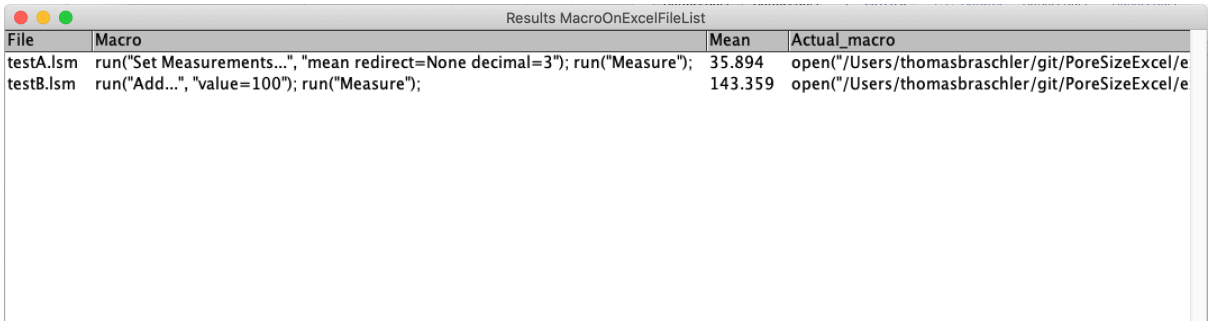


Figure 19. Indication of a macro column in the plugin dialog.

When the plugin is run, it specifically applies the macro defined for each image line in the Excel file. While the example given in "test_simple_macro.xlsx" has probably limited practical application (different macros to different files), applying different macros makes sense in several conditions. First, it is possible to list several times the same image in the Excel file, but with different macros. This allows to evaluate a given parameter of interest with different modalities, or different threshold values, and so forth. And second, it is also possible to simply have the same macro for all lines, for example by using Excel formulae.

In the output, the macro specified in the Excel file for each line, but also the exact applied macro with open and close commands is listed (Figure 20).

Note that the second macro in Figure 18 (and correspondingly Figure 20) modifies the image. Since there is however no file saving command in the macro, this will not change actual image file.

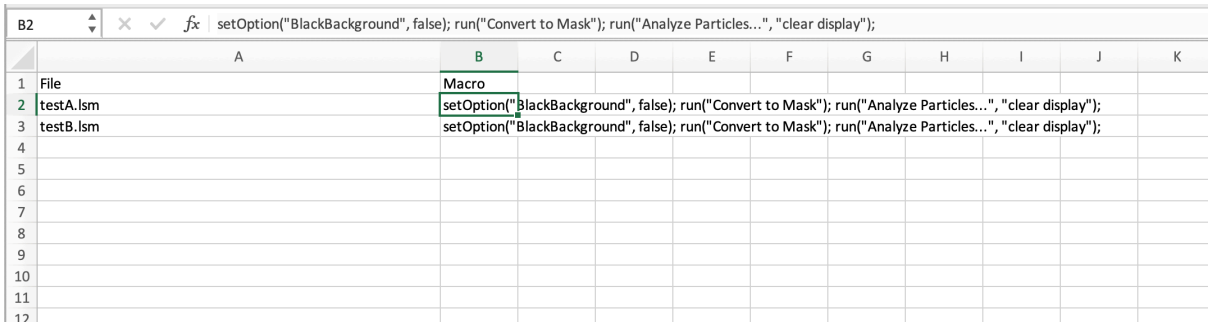

Results MacroOnExcelFileList

File	Macro	Mean	Actual_macro
testA.lsm	run("Set Measurements...", "mean redirect=None decimal=3"); run("Measure");	35.894	open("/Users/thomasbraschler/git/PoreSizeExcel/e
testB.lsm	run("Add...", "value=100"); run("Measure");	143.359	open("/Users/thomasbraschler/git/PoreSizeExcel/e

Figure 20. Output when running macros specified in Excel.

4.5. Macros with multiple output lines

Some ImageJ commands produce multi-line output from single images, particle analysis being an example. If a macro with multiple output lines is performed for an image, the PoreSizeExcel will report all the lines in its output, associated with the file, folder, macro and other columns. This leads to a duplication of the otherwise unique values, with the advantage that the results are directly usable in matrix-based programs such as Matlab or R.

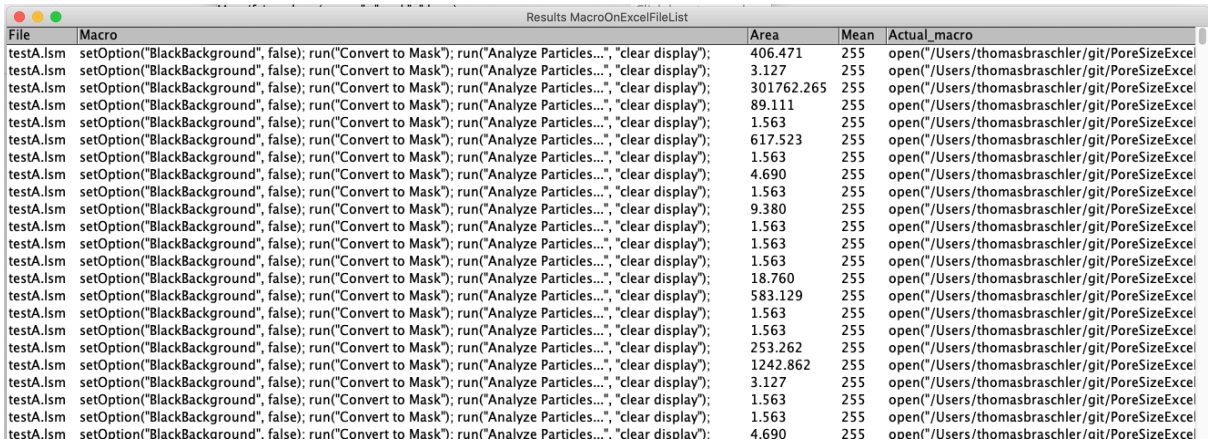

B2 | setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");

	A	B	C	D	E	F	G	H	I	J	K
1	File	Macro									
2	testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");									
3	testB.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");									
4											
5											
6											
7											
8											
9											
10											
11											
12											

Figure 21. Excel file test_multiline_macro.xlsx with macros that produce multiline output.

In the examples, “test_multiline_macro.xlsx” invokes such ImageJ macros with multiline output (Particle analysis after thresholding; see Figure 21).

The plugin output is shown in Figure 22; one can see that many lines have arisen from the first image (“testA.lsm”), leading to duplication of the otherwise unique values per line.


Results MacroOnExcelFileList

File	Macro	Area	Mean	Actual_macro
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	406.471	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	3.127	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	301762.265	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	89.111	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	1.563	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	617.523	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	1.563	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	4.690	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	1.563	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	9.380	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	1.563	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	1.563	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	1.563	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	18.760	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	583.129	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	1.563	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	1.563	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	253.262	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	1242.862	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	3.127	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	1.563	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	1.563	255	open("/Users/thomasbraschler/git/PoreSizeExcel
testA.lsm	setOption("BlackBackground", false); run("Convert to Mask"); run("Analyze Particles...", "clear display");	4.690	255	open("/Users/thomasbraschler/git/PoreSizeExcel

Figure 22. Plugin output in the presence of multiple result lines per image.

4.6. Excel column values in macros

There are occasions where the same macro functionality needs to be applied to a series of images, but with a different key parameter. For example, there may be a need to apply a manually determined threshold to the images before analysis of pore size or area fraction of bright pixels. This threshold may be different for each image, even though otherwise the macro to be applied is the same.

It is possible to either manually type the corresponding macro for each line or use Excel's string assembly capacities to assemble the macro for each line from different pieces of information. We provide here an additional possibility that allows to keep the macro as recorded in Excel identical, but permits some limited string substitution directly in the plug-in to use values recorded in different Excel columns to "fine-tune" the macro.

4.6.1. Scenario

Let us assume we need to measure the fraction of "bright" pixels in a series of images. For this, we need to threshold each image, and then measure the mean grey value – 255 would signify all pixels are bright, 0 none, with proportionality between. Let us also assume that it is not possible to use auto-thresholding, for instance because the threshold is determined by an external negative and positive control, specific to each image, and not some intrinsic image properties.

4.6.2. Basic macro code

The basic ImageJ macro needed to determine the fraction of bright pixels on an individual image 8bit greyscale image could be:

```
run("Subtract...", "value=50"); run("Multiply...", "value=255"); run("Divide...", "value=255");
run("Set Measurements...", "mean redirect=None decimal=5"); run("Measure");
```

In this sequence of macro commands, the subtraction of 50 leads a pixel value of 0 for all pixels with original intensities from 0 to 50. The subsequent multiplication by 255 fixes all the non-zero pixel at the maximum value of 255. The division by 255 causes the pixel value for the bright pixels to be reduced to 1, whereas the dark pixels remain at 0. This is then followed by setting the measurement target to mean grey value and actual measurement. The final measurement output is the fraction of bright pixels, in the range of 0 for completely "dark" original image and a 1 for a completely "bright" original image. Since no file saving command is included, the original image on disk would not be modified.

4.6.3. Column substitution

The threshold value in the code example above is 50, but in our scenario, this would be variable for different images. The idea of the column substitution is to be able to store this threshold separately in a specific "Threshold" column in Excel, the plugin then substituting the corresponding desired value for each image. We chose to use a "%Column_name%"

syntax for this type of substitution. The column-substitution macro for a threshold value tabulated in a “Threshold” column would then read:

run("Subtract...", "value=%Threshold%"); run("Multiply...", "value=255"); run("Divide...", "value=255"); run("Set Measurements...", "mean redirect=None decimal=5"); run("Measure");

4.6.4. Excel file: test_column_substitution.xlsx

The file “test_column_substitution.xlsx” contains an example of elementary column substitution with the macro listed above (section 4.6.3). Note that the column name of the “Threshold” column needs to exactly match the column name used for substitution (“%Threshold%”), including capitals and white spaces.

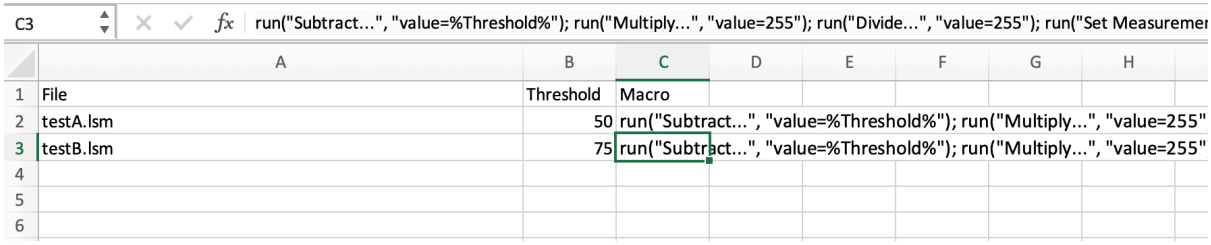


Figure 23. test_column_substitution.xlsx with a macro with column substitution.

4.6.5. Output: Template and actual macro

In its output, the PoreSizeExcel plugin documents both the original macro figuring in the Excel file (here, in column “Macro”) and the macro that was actually executed for each file. In the previous examples (cf. Figure 20) this only involved the addition of an “open” and “close” command. Here, additionally, the column substitution is documented in “Actual_macro” column of the output. This is shown in Figure 24: the actual macro was executed with the tabulated threshold values (50 and 75, as shown in Figure 23) rather than the “%Threshold%”.

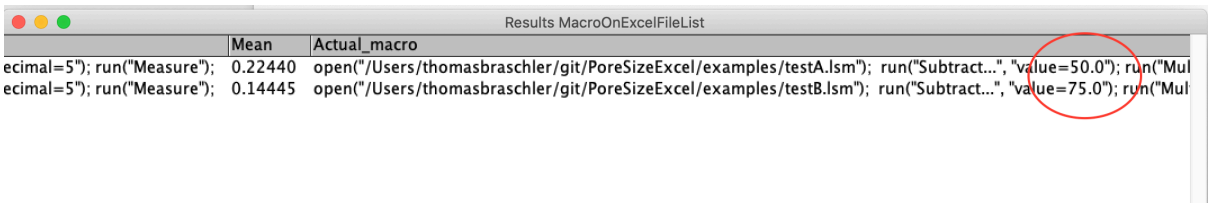


Figure 24. Plugin output when using column substitution. The encercled values correspond to the variable threshold values (compare to Figure 23).

4.6.6. Column substitution and macro specification within the plugin

It is also possible to use macro specification within the plugin (as in Figure 11) rather than from the Excel file. However, particularly with column substitution it becomes more subtle to reproduce the results because the template macro then doesn’t get stored in a file and needs to be reconstituted every time the plugin is used. Hence, we sometimes use this

option for debugging purposes but generally use reading of the Macro from the Excel file option.

4.7. Saving of modified image files

By default, the PoreSizeExcel file does not modify images on the hard drive. Also, we consider it not advisable to permanently save the modifications to the original files; anyways, with the confocal .lsm or .czi files that we usually treat, this is not easily possible in ImageJ or Fiji.

However, there are many occasions where one wants to save treated images. For instance, one may want to document intermediate steps such as thresholding, or one may be interested in the actual image output rather than some text or numerical output as tabulated in the previous examples.

In this case, it is necessary to incorporate some image saving or exporting command into the macros.

4.7.1. Scenario

Let us assume we are interested in the thresholded images generated intermediately in section 4.6, rather than in the fraction of bright pixels. In that case, it is necessary to save the images modified in memory before the PluginExcelFile closes them.

4.7.2. Basic macro

The basic macro for obtaining a black-and-white image thresholded at some value (here, 50), followed by file saving could be:

run("Subtract...", "value=50"); run("Multiply...", "value=255"); saveAs("Tiff", “some_file_name.tif");

It is quite evident that the file name (“some_file_name.tif”) cannot be constant for different images, otherwise ImageJ will save repeatedly onto the same file on disk and one will in the end only have the result of the last image. Hence, result file names need to be either hard-coded into the Excel file or they need to be generated by column substitution (section 4.6).

4.7.3. Absolute file path

When saving files, ImageJ (and Fiji) will save into the current working directory, unless an absolute file path (rather than just a file name as above) is specified. The working directory is given by the last file opening or saving operation, and may, or may not, correspond to the desired file location for saving.

We find it thus advisable to impose absolute rather than relative file paths. To do so, the PoreSizeExcel plugin provides an additional column-substitution variable: %root_folder%.

This variable can be used just as the column substitution (i.e. section 4.6), and always points to the root folder chosen in the plugin dialog (e.g. the content in the textbox after the “Manual root folder:” label, see Figure 11).

So to force file saving relative to the specified root folder, the macro would read:

run("Subtract...", "value=50"); run("Multiply...", "value=255"); saveAs("Tiff", “%root_folder%/some_file_name.tif");

Of course, one could use any actual absolute path as well instead, but this comes at the cost that executing the macro on a different machine will attempt to save to the old location – which will typically either fail or produce undesired results.

4.7.4. Excel file: test_save_file.xlsx

The listing of the test_save_file.xlsx file is shown in Figure 25. The %root_folder% variable permits to save relative to the common root folder chosen for plugin execution. In this example, the target file names (“testA_thresholded.tif”, “testB_thresholded.tif”) are hard-coded for simplicity, but it would typically also be possible to use column substitution for this (section 4.6). In that case, the file path would be something of the kind “%root_folder%/%File_name%”, and there would have to be an additional “File_name” column containing the target file names.

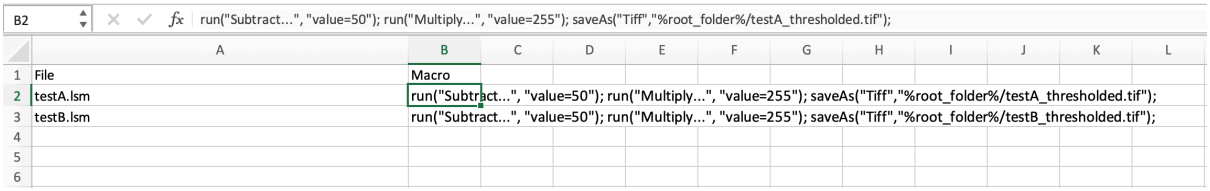


Figure 25. Listing of « test_save_file.xlsx ». Note the use of %root_folder% in the macro definition, this will get replaced by the root folder in which the plugin is run.

4.7.5. Running the plugin

When running the plugin, one has to keep in mind that whatever the root path indicated in the Plugin dialog box is will be the root path for saving the files if %root_folder% is used in the macro. This principle is shown in Figure 26.

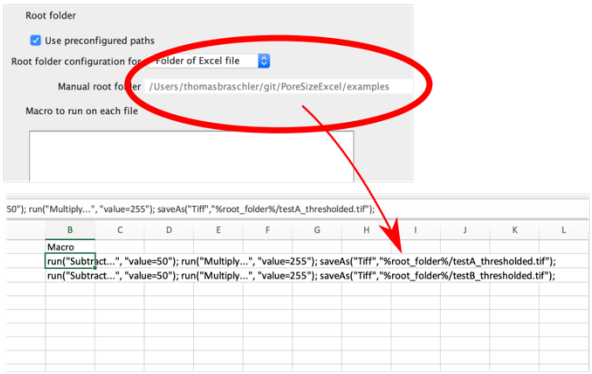


Figure 26. %root_folder% substitution. The PoreSizeExcel plugin substitutes occurrences of %root_folder% in the macro with the actual root folder configured in the root folder text field (encircled in the screen shot from the Plugin dialog).

In many cases, it is convenient to have the root folder identical to the folder in which the Excel file resides (default behaviour of the plugin), but there are also cases where one would like to separate data storage from the Excel files. For this, the check box in front of "Use preconfigured paths" in Figure 26 would have to unchecked an a manual path supplied in the text box. In addition, since a unique root folder is used for the automated file opening and user-defined macro operation is used, it would be necessary to hard-code the absolute paths for file opening.

5. Pore size evaluation

5.1. Pore size evaluation by Beat Münch's xlib plugin

5.2. PoreSizeExcel interface

5.3. Integrating pore size evaluation into a an Excel-ImageJ macro workflow

6. Special tasks

6.1. Programmatically defining file locations for different collaborators

7. Bibliography

1. Münch, B. & Holzer, L. Contradicting Geometrical Concepts in Pore Size Analysis Attained with Electron Microscopy and Mercury Intrusion. *Journal of the American Ceramic Society* **91**, 4059–4067 (2008).