**Using ImageJ ROIs for Imaging Analysis**

by David Whitney at Max Planck Florida Institute (Last update on 4/7/2018)

Please email any questions/comments here: David.whitney@mpfi.org

**Introduction**

In this tutorial, we’ll learn to take advantage of an ImageJ plug-in called Cell Magic Wand, which streamlines the process of selecting ROIs in a semi-automated way. The strength of a semi-automated approach relative to other methods is that the biologist has the final word on what constitutes an acceptable ROI. Furthermore, most state-of-the-art automated methods suffer from an acceptable level of precision, and still need to be manually curated afterwards by a biologist. The semi-automated approach fills this niche, as the biologist can rapidly draw new ROIs or correct erroneous ones with a single mouse-click, rather than having to painstakingly draw a ROI border around every biologically relevant feature of interest.

**Overview of ImageJ and Cell Magic Wand**

For those not yet familiar with ImageJ, it is an open-source Java-based image processing application developed and supported by the National Institutes of Health. The application is fast, lightweight, easy to use, and has many routines and plug-ins that are suitable for functional imaging analysis. Cell Magic Wand is just one fantastic example of the many useful plug-ins written for ImageJ. Briefly, the Cell Magic Wand tool automatically generates a ROI from a single user click on a biological feature of interest, whether it be a cell, spine, or bouton. Under the hood, the routine works its magic by converting a user’s click into a centered seed point. The algorithm then automatically determines a ROI border within a prespecified radius range around the seed point by using a polar transform and an edge-following algorithm. The original Cell Magic Wand plugin was written by the talented programmer Theo Walker, and can be downloaded at either of these locations:

* [Max Planck website](https://www.maxplanckflorida.org/fitzpatricklab/software/cellMagicWand/)
* [GitHub](https://github.com/fitzlab/CellMagicWand)

**Using ImageJ standalone (without MATLAB):**

* **Step 1: Download and install ImageJ.**

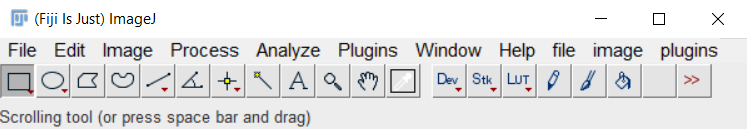
ImageJ can be downloaded here: <https://imagej.net/Fiji/Downloads>. The file should be named something like fiji-win64.zip. Open the \*.zip file and extract the Fiji.App folder to the desired location. For Windows users, I recommend copying the folder to the root path of the system drive (i.e. “C:\Fiji.app\”).

* **Step 2: Download Cell Magic Wand**

Download the Cell Magic Wand tool (see download links above). We will need to copy the Cell\_Magic\_Wand\_Tool.jar into the "plugins " subfolder of our ImageJ root path (i.e. “C:\Fiji.app\plugins\”). An alternative installation guide for the Cell Magic Wand tool is located here on the [Max Planck website](https://www.maxplanckflorida.org/fitzpatricklab/software/cellMagicWand/).

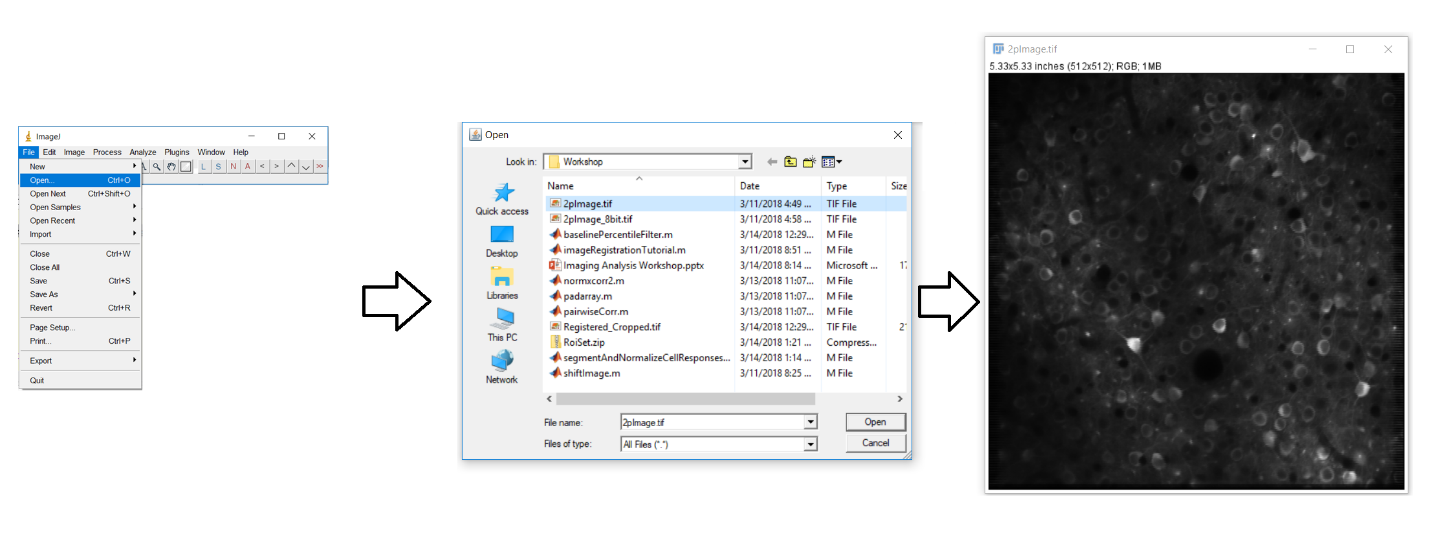
* **Step 3: Run ImageJ**

Run ImageJ-win64.exe in the Fiji.app folder to launch ImageJ. It should look like this:



* **Step 4: Open an Image in ImageJ:**

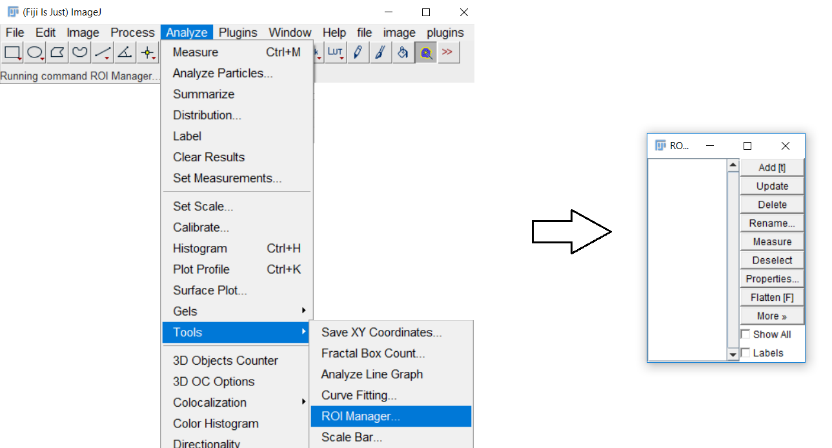
Now we’ll open up an image that we can use to find ROIs. On ImageJ’s main menu, click File 🡪 Open. A file dialog will pop-up, and we can navigate to the location of the imaging file that we will use to select ROIs.



**IMPORTANT TIP:** Generally, the best images to use for selecting ROIs tend to be either a mean or standard deviation projection image (either raw fluorescence or ΔF/F). Users can open either a full or partial imaging stack within ImageJ and generate a mean or standard deviation projection image using the Z-Projection tool. This can be accessed via ImageJ’s main menu: Image 🡪 Stacks 🡪 Z-Projection.

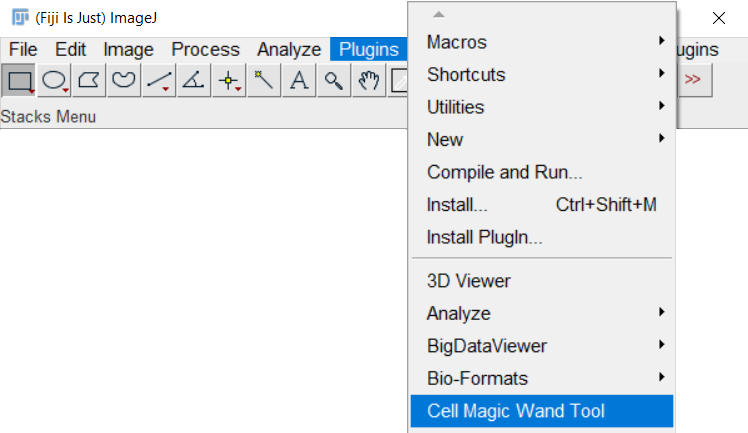
* **Step 5: Launch the ROI Manager**

On ImageJ’s main menu, we can launch ImageJ’s native ROI Manager by clicking Analyze 🡪 Tools 🡪 ROI Manager …

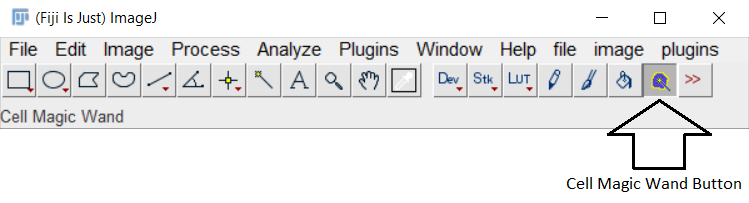


* **Step 6: Activating cell magic wand in ImageJ.**

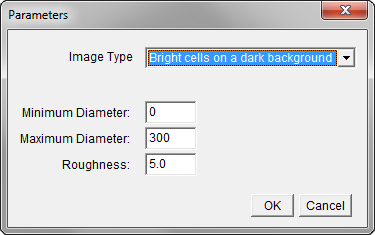
On ImageJ’s main menu, we can activate the Cell Magic Wand tool by clicking Plugins 🡪 Cell Magic Wand Tool.



We should now see the Cell Magic Wand cursor:



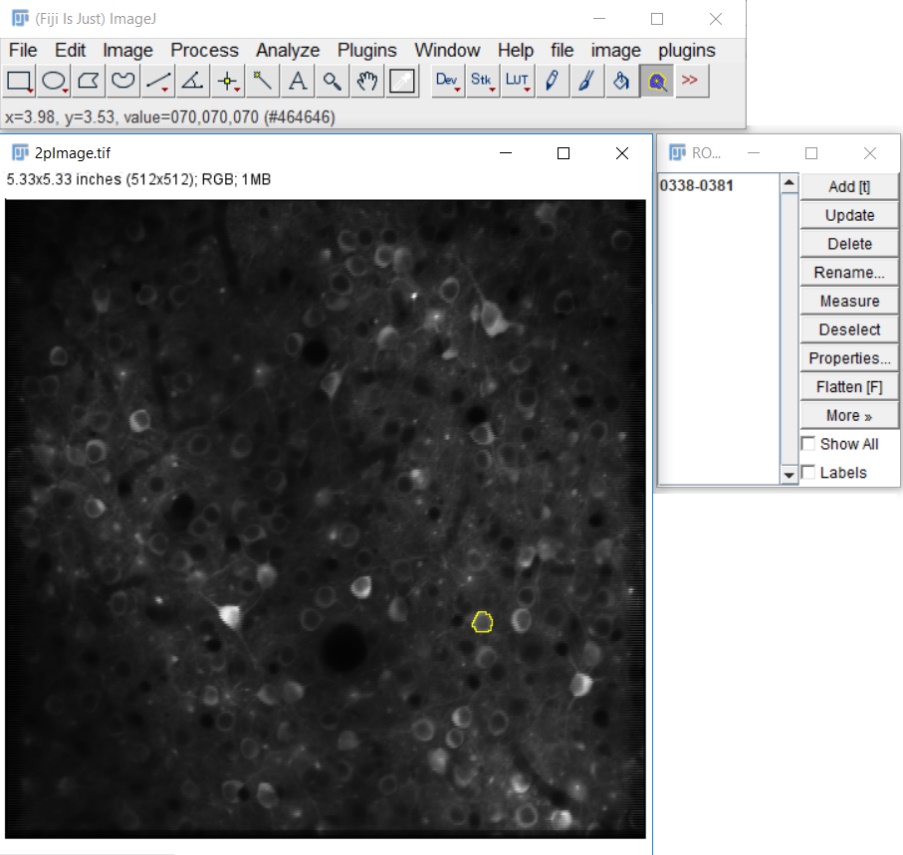
We can customize how the plugin works by double-clicking on the Cell Magic Wand button. The following parameters box will pop-up:



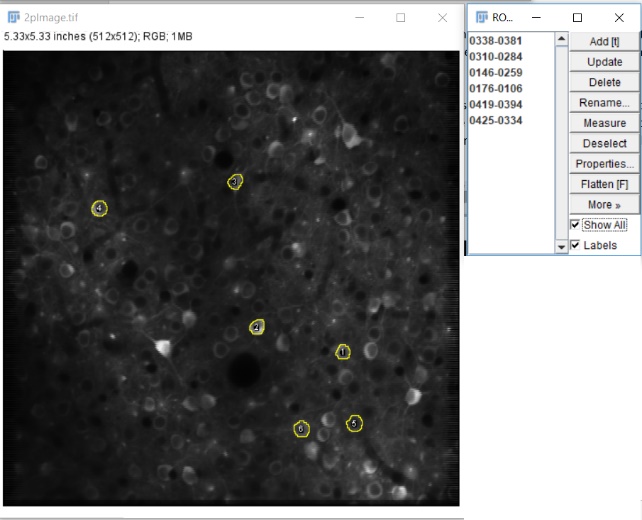
**IMPORTANT TIP:** For Cell Magic Wand to work effectively, we need to optimize the minimum and maximum diameter (in units of pixels) so that the generated ROI is at the approximate scale of the ROI that we expect. The roughness parameter can be varied to make the ROI smoother. A roughness value of 0 will give a smooth circle, while higher values (usually up to 10) will make the ROI reflect the actual detected “edge.”

* **Step 7: Using Cell Magic Wand to select ROIs**

We can select a ROI by simply left-clicking regions on our image. An enclosed yellow line will denote the algorithm’s generated ROI. If we are satisfied with the ROI, clicking ‘T’ will save the ROI in our ROI manager. The name of the selected ROI will be the (X,Y) location of the seed point.



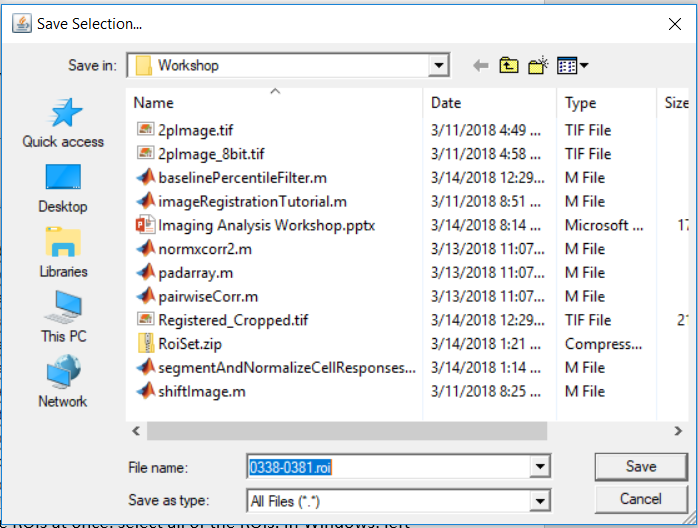
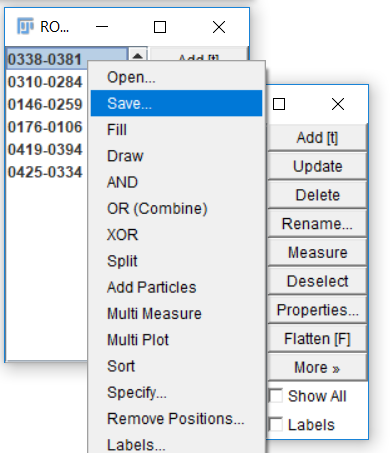
We can select additional ROIs by left-clicking on the image again. If we want to show all the ROIs at once, we need to make sure that the “Show All” box is clicked on the ROI Manager. Also, if we ever find that we cannot select additional ROIs, try re-clicking the Cell Magic Wand button to reactivate the plugin.



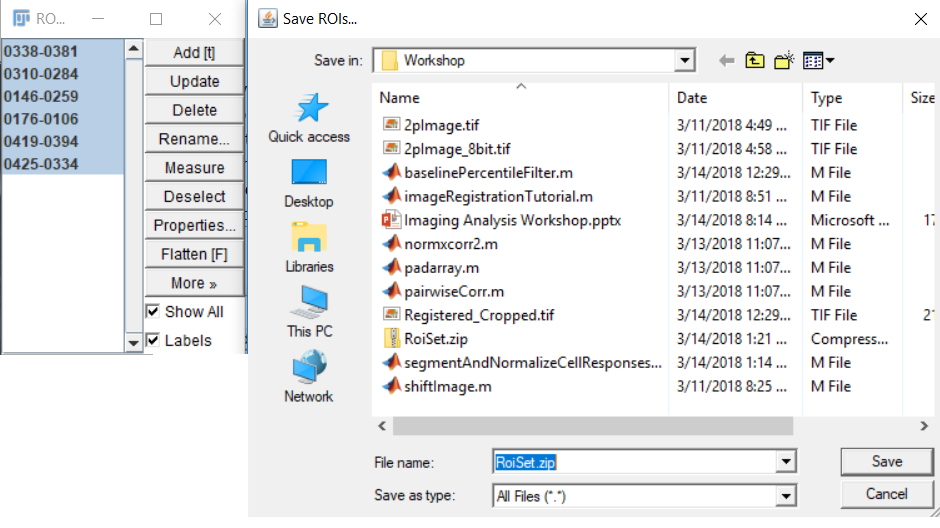
**IMPORTANT TIP:** If at any point we are dissatisfied with the ROI, we can click a nearby location to force Cell Magic Wand to revise the detect edge. If this action doesn’t really improve upon the ROI, we will then need to optimize the parameters used by the Cell Magic Wand algorithm (see Step 6 for more details).

**Step 8: Saving the ROIs**

We can save an individual ROI by right clicking on the ROI in the ROI Manager, and then selecting the save command. A pop-up dialog will query where to save the ROI. The file format is a \*.ROI file.



If we want to batch save every ROI together, we need to first select all of the ROIs. In Windows, left-click on one of the ROIs in the ROI manager and click Ctrl+A. Now right click again and then click the save command. In this case, the saved file format is a \*.zip file containing a \*.ROI file for each ROI.



**Step 9: Open up ROIs in other analysis software**

We have several different ways to open up ImageJ ROIs in different analysis software. For example, other pioneering researchers have already written up applications to read ImageJ ROIs into both MATLAB and Python (both popular programming languages in biology):

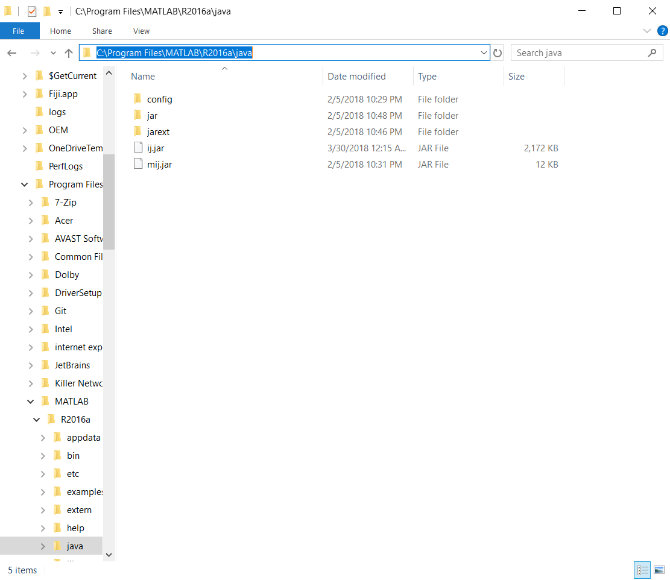
* + **MATLAB:** [ReadImageJROI (by Dylan Muir)](https://www.mathworks.com/matlabcentral/fileexchange/32479-readimagejroi)
  + **Python:** [Read-ROI (by Hadrien Mary)](https://pypi.python.org/pypi/read-roi), [ijroi (by Luis Coelho)](https://pypi.python.org/pypi/ijroi)
  + **Other:** If neither MATLAB or Python are suitable, please refer to the [ROIDecoder from the ImageJ API](https://imagej.nih.gov/ij/developer/source/ij/io/RoiDecoder.java.html) to learn how to decode ImageJ ROI files.

**Interfacing ImageJ directly with MATLAB**

**(for Windows and Mac Users)**

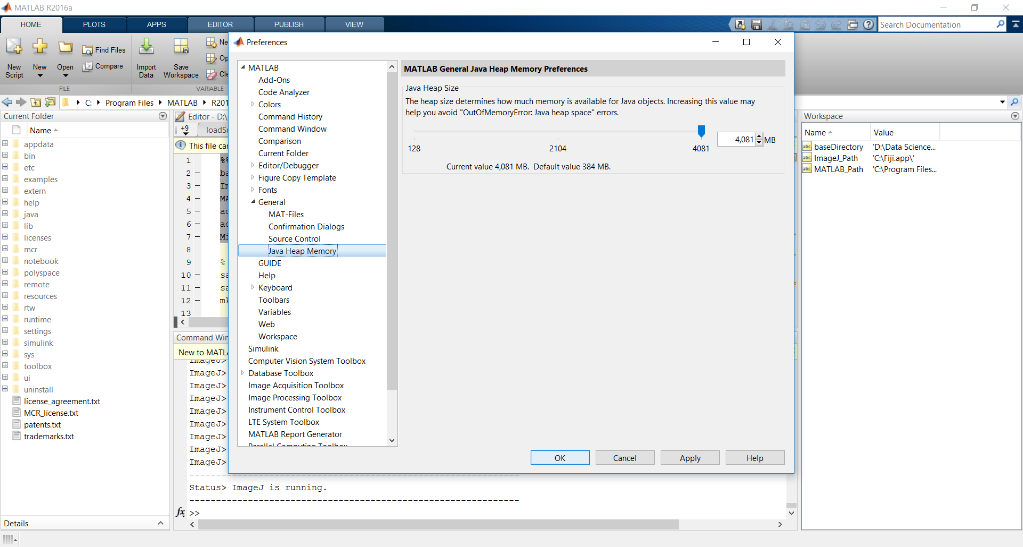
[MATLAB](https://www.mathworks.com/products/matlab.html) is a popular programming language in neuroscience that can be configured to run an active instance of ImageJ within a Java Virtual Machine that is directly accessible by MATLAB. This is fantastic because MATLAB users can take full advantage of the powerful analytical tools available in ImageJ. For those interested, the following instructions include the necessary steps to access ImageJ directly through MATLAB:

* Download the [mij.jar](http://bigwww.epfl.ch/sage/soft/mij/mij.jar) and [ij.jar](http://rsb.info.nih.gov/ij/upgrade/ij.jar) files from the [MIJ website](http://bigwww.epfl.ch/sage/soft/mij/), and then copy both files to the “java” subfolder of our MATLAB root path:
  + Example for Windows Users:
* Copy the mij.jar to 'C:\Program Files\MATLAB\R2017a\java\mij.jar'
* Copy the ij.jar to 'C:\Program Files\MATLAB\R2017a\java\ij.jar'
  + Example for Mac Users:
* Copy the mij.jar to '/Applications/MATLAB\_R2017a.app/java/mij.jar'
* Copy the ij.jar to '/Applications/MATLAB\_R2017a.app/java/ij.jar'



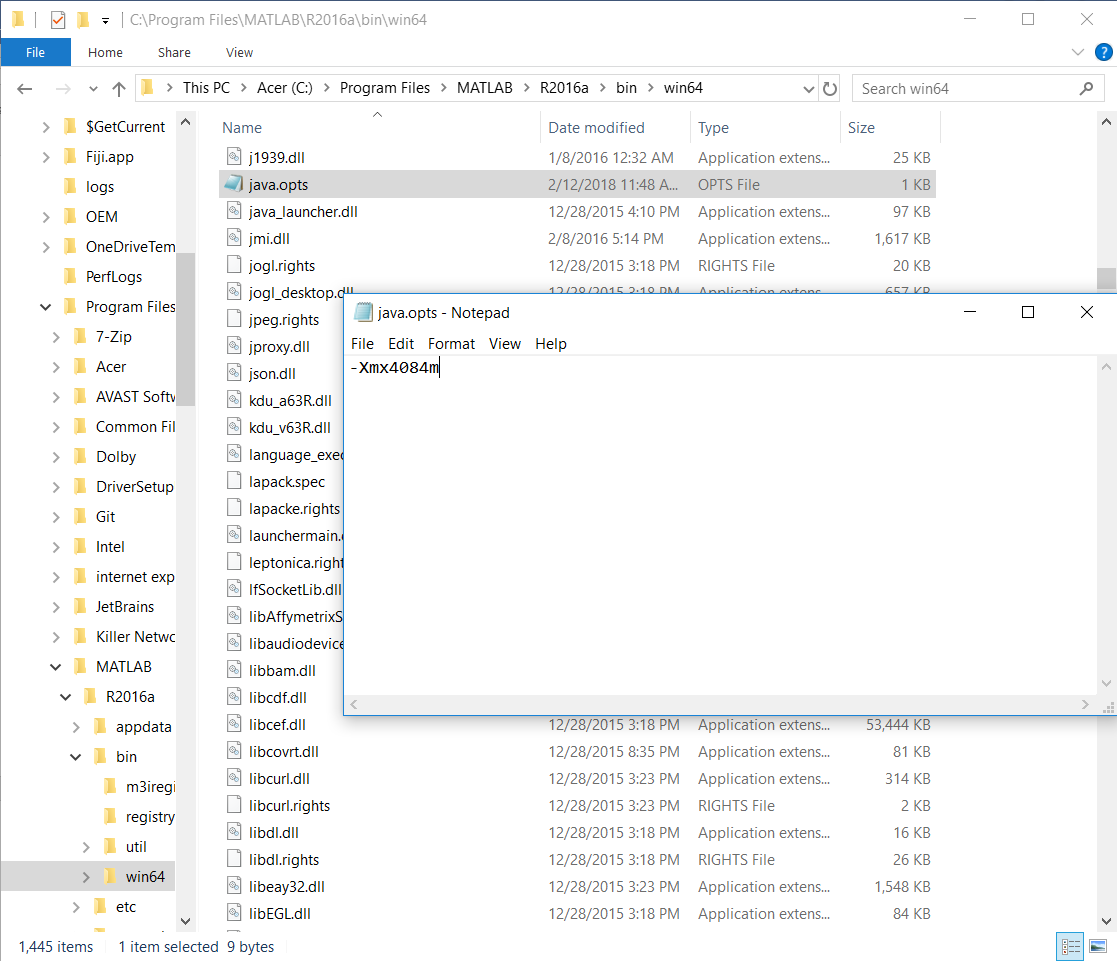
If the MATLAB root path is not directly known, we can find out the path via the "matlabroot" function in MATLAB.

* Next, we need to increase the Java Heap space allowed for ImageJ in MATLAB. If we don’t do this, we’ll likely run into memory issues loading even the smallest imaging stacks. The easiest way to increase our Java Heap space is to make the changes directly in MATLAB. From the MATLAB Main Menu, click Preferences 🡪 MATLAB 🡪General 🡪 Java Heap Memory. Increase the Java Heap Size by sliding the slider to the right.



Unfortunately, the maximal amount of RAM that can be allocated this way is 25% of our total RAM. If more RAM needs to be allocated to the Java Heap Size, open a text editor and write the following on a single line: “-Xmx4096m” (without the quotes included). The 4096 here corresponds to 4GB=4096. We can increase this number further if more RAM needs to be allocated to the Java Heap Memory. Save the text file with the name “java.opts” into the "bin\win64" subfolder of our MATLAB root path.

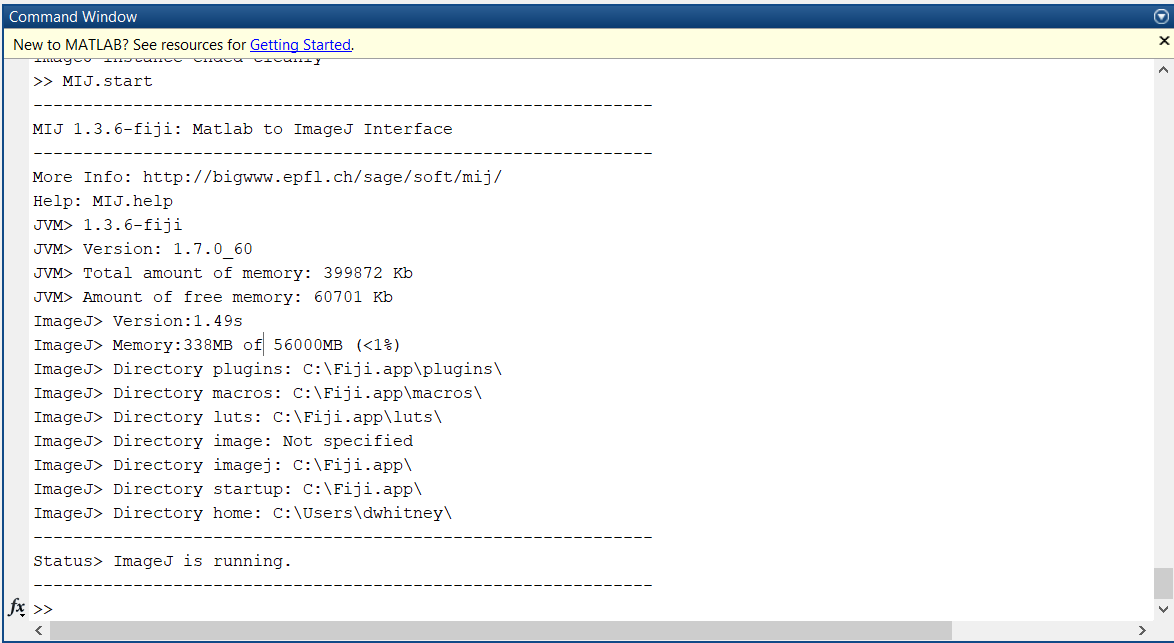
* + Example for Windows Users:
* 'C:\Program Files\MATLAB\R2017a\bin\win64\java.opts'
  + Example for Mac Users:
* '/Applications/MATLAB\_R2017a.app/bin/win64/java.opts'



* Finally to be able to run ImageJ in MATLAB, we need to add the mij.jar and ij.jar to MATLAB's java classpath each time we run MATLAB:
  + Example for Windows Users:
* Type: javaaddpath 'C:\Program Files\MATLAB\R2017a\java\mij.jar'
* Type: javaaddpath 'C:\Program Files\MATLAB\R2017a\java\ij.jar'
  + Example for Mac Users:
* Type: javaaddpath '/Applications/MATLAB\_R2017a.app/java/mij.jar'
* Type: javaaddpath '/Applications/MATLAB\_R2017a.app/java/ij.jar'

If we don't want to type these commands each time MATLAB launches, we have the option to add these lines of code directly to our startup.m file.

* Congratulations you're done configuring MATLAB to run an internal instance of ImageJ! Start ImageJ in MATLAB with "MIJ.start".



**How to interface with ImageJ tools in MATLAB**

This is admittedly trickier and requires that the user be comfortable with the idea of objects in object-oriented programming. Fortunately, one of the \*.jar files that we added to our path (MIJ.jar) provides some useful commands that makes it much easier to interact with ImageJ from MATLAB. Please refer to this guide for more [information](http://bigwww.epfl.ch/sage/soft/mij/), and an API is available [here](http://bigwww.epfl.ch/sage/soft/mij/doc/MIJ.html). The most relevant commands are:

* **MIJ.start()** – Starts an instance of ImageJ in MATLAB
* **MIJ.exit()** – Ends the active instance of ImageJ in MATLAB
* **MIJ.run(commandName, argumentList)** – Externally runs an ImageJ command denoted by the *commandName* string using the optional argument list specified by the string *argumentList*. Supplying the correct argument list may seem hard at first, but one of the best ways to learn how to supply the correct arguments is to use the ImageJ command Recorder as you try different ImageJ tools (Plugins 🡪 Macro 🡪 Record). This tool will output a string, which can be usually directly used in the *argumentList* string.
* **imgHandle=MIJ.createImage(imgName, imgStack, showImage)** – This function sends *imgStack* (a 2d or 3d array) to ImageJ with the name specified by the string *imgName*. The optional *showImage* Boolean flag denotes whether to show the image in ImageJ (default is true). The function returns a handle to the active image object in ImageJ.
* **imgStack = MIJ.getImage(imgName)** – This function returns a 2d or 3d array (*imgStack*) for the active image object with the name specified by the *imgName* string.