# ImageJ Batch Processing

**Alex Herbert**

MRC Genome Damage and Stability Centre
School of Life Sciences
University of Sussex
Science Road
Falmer
BN1 9RQ

a.herbert@sussex.ac.uk

## Table of Contents

# 1 Introduction

ImageJ is an open source image analysis program. The program provides a plugin framework for adding custom functionality. There are many analysis routines built in to ImageJ and hundreds more can be freely downloaded from the web.

This document describes how ImageJ can be used to automate the processing of large numbers of image files in a batch process.

## *1.1 Single Image Analysis*

### 1.1.1 Open the image

ImageJ can open a large number of different image types using the File > Open... command. If your image type is not supported then it may be possible to open it using the BioFormats plugin. You can check the list of supported formats here:

http://www.loci.wisc.edu/bio-formats/formats

The BioFormats plugin can be downloaded from The BioFormats website:

http://www.loci.wisc.edu/files/software/loci_tools.jar

Place the Jar file in the ImageJ plugins directory, restart ImageJ and access the BioFormats import functionality using Plugins > LOCI > Bio-Formats Importer.

### 1.1.2 Analysis using Plugins

ImageJ has a large number of analysis commands available from the Image, Process and Analyze menus. Standard features include particle analysis, thresholding and histogram statistics.

The analysis capabilities of ImageJ can be significantly expanded by using third party plugins. Plugins must conform to the ImageJ plugin framework. When placed in the ImageJ plugins directory they are automatically identified by ImageJ and become available for use (after restarting ImageJ or using the Help > Refresh Menus command).

There are many plugins freely available from the web. A large list is maintained on the ImageJ website:

http://rsbweb.nih.gov/ij/plugins/

Most plugins will add commands to the Plugins menu. However please consult the documentation of the plugin if available.

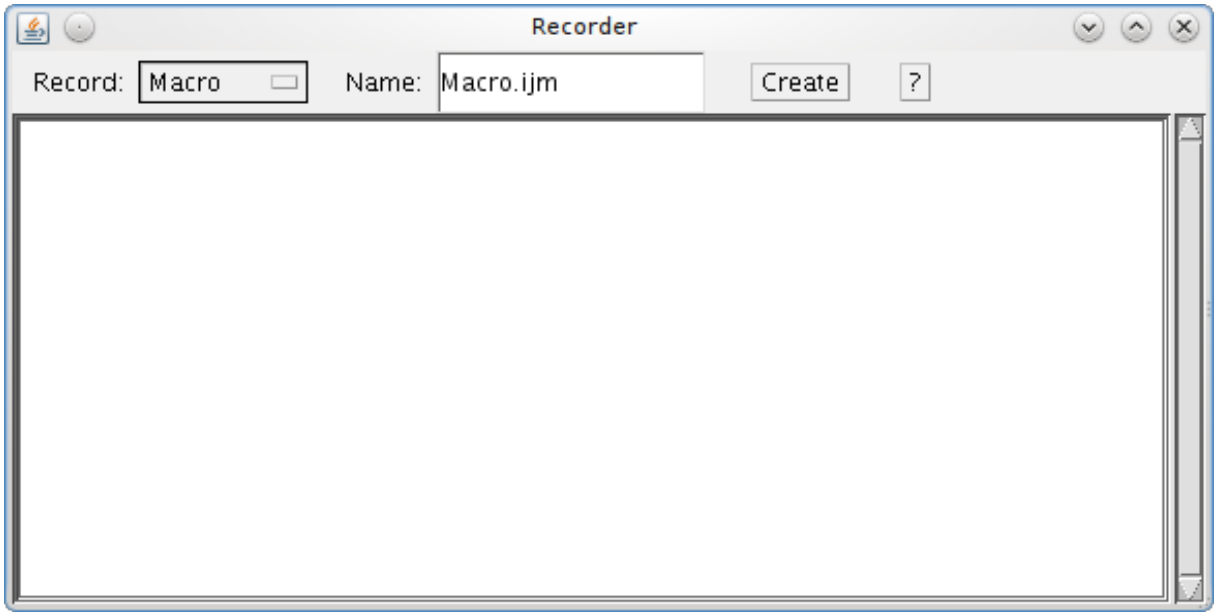
When a plugin is run the user is often presented with options to configure the plugin parameters. After configuration the plugin will perform the analysis and the results will appear. Plugins commonly make changes directly to an image. However some plugins provide results tables and other visual displays. In many cases the parameters are easy to configure and it is possible to run the analysis plugin without user interaction, i.e. in an offline mode. The analysis can then be automated to allow processing of many images in a batch.

# 2 Macro Recorder

ImageJ provides a recorder that can record any actions that are run by the user. These actions can then be saved as macros and re-run on other images. The macro recorder is a

powerful tool that can be used to automate the processing of large numbers of images.

To start the recorder choose Plugins > Macros > Record... This will present a recorder dialogue as shown below:

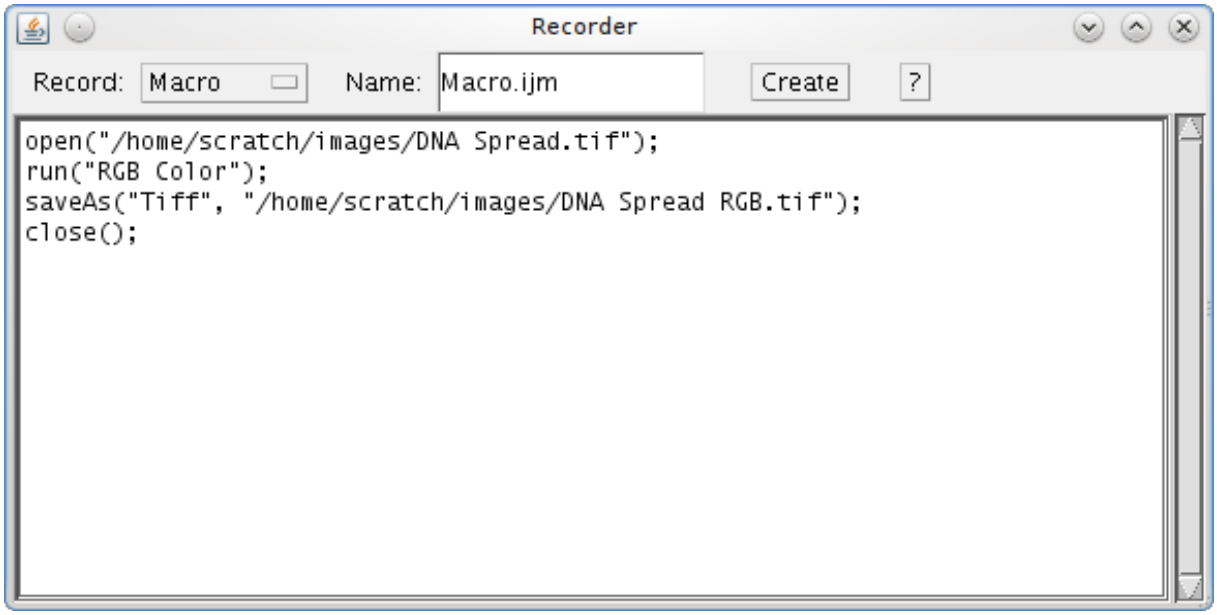



The Recorder provides the option to record the commands using different languages. The options are described below.

| Record Mode | Description |
|---|---|
| Macro | Uses the ImageJ macro language.<br>This language is used by the ImageJ macro tools. This is the correct option for automatically generating script commands to use in the Batch Command plugin |
| JavaScript | Uses the JavaScript programming language. |
| Plugin | Uses the Java programming language.<br>This can be used to generate a Java plugin to be installed in the ImageJ plugins directory |

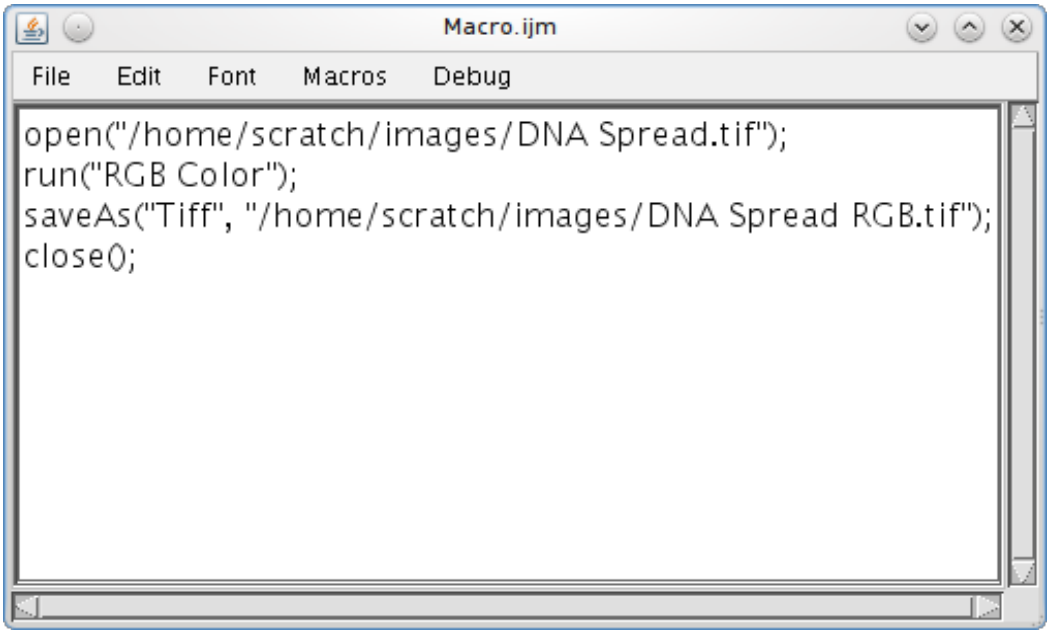
Unless you have a preference for writing scripts in JavaScript then it is better to use the ImageJ macro language (since there are more examples and help that can be found on the web for standard ImageJ macros).

The Recorder will record any command that supports the recorder functionality. This covers all of ImageJ's commands and most plugins. The follows shows the result of opening and image, converting it to a mask and then closing the image:




Note that the Recorder window is a text area that can be modified. Thus it is possible to change commands or delete commands that are not wanted in the macro.

When you have finished performing you work you can click the Create button and ImageJ will use the recorded commands to generate a new macro, script or plugin. The output type is controlled by the current Record mode. In the case of the Macro mode, ImageJ will create a new Macro window containing the macro. The macro generated for the previous recorded commands in shown below:




The new macro can be edited and saved. The macro can be run using Macros > Run Macro. This causes ImageJ to run each step of the macro in turn. Any dialogues normally required by the plugin command are suppressed and ImageJ will pass the specified parameters to the command. In the example above the parameters 'calculate black' are passed to the command 'Convert to Mask'.

## 2.1 Sequential Images in a Directory

A simple use for recorded macros is to process all images in a directory:

- Place all images for analysis in a single directory
- Open the macro containing your analysis commands (or generate a new one using the recorder)
- Open the first image in the directory
- Run the macro in the macro window (Macros > Run Macro)
- Open the next image using File > Open Next
- Re-run the macro
- Repeat this process for all images in the directory

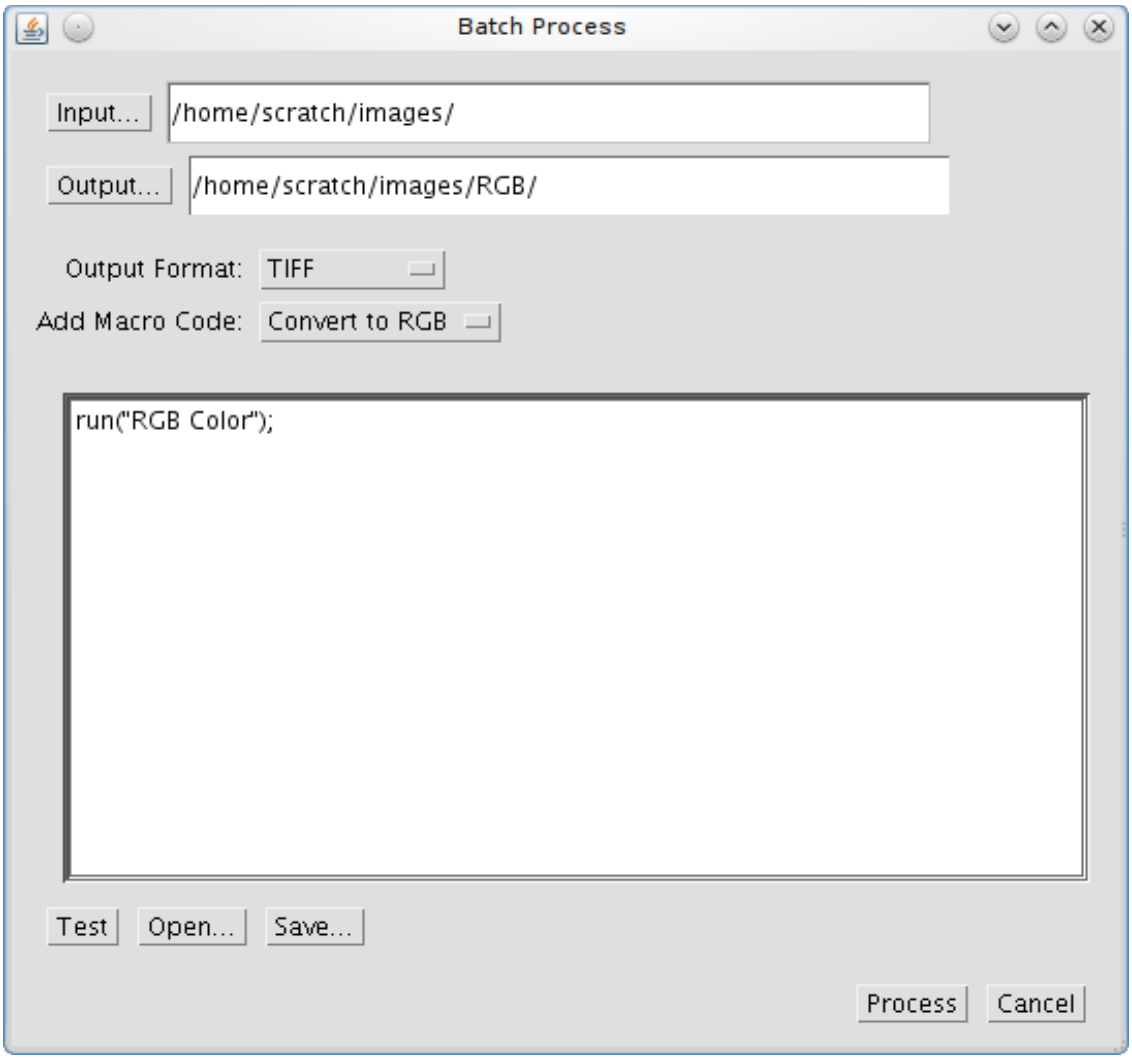
This procedure is useful for quickly viewing the results of commands on many images since the user can pause at each image to see the results. However if the user does not want to interact with the images then ImageJ can run a macro command on all images in a directory without pausing (see the Batch Process Command section).

Note that the Open Next command constantly cycles through images in the current directory. ImageJ does not inform the user when all images have been processed.

# 3 Batch Process Command

ImageJ provides the ability to perform the same actions to a set of images. These are available under the Process > Batch menu item. The most useful option is the Process > Batch > Macro which performs a macro command on each input image. Other option include running the standard ImageJ Measure command on the images (e.g. for minimum, maximum, etc) or resizing the images.

The following image displays an example of the Batch Process dialogue:




The dialogue allows you to select an input directory containing all your images. You can then specify an optional output directory and the format for the output images. The dialogue provides the ability to select from common commands which are then inserted into the text area. You can also open your own macros or paste the code from the ImageJ macro recorder. The Process button starts the batch processing.

For each image in the input folder ImageJ will open the image, apply the commands, save the image to the output folder (if present) and then close the image. Any results windows generated by the plugin commands will remain open.

Note that if you do not specify an output directory then the images will not be saved after running the command. This is useful if your macro command does not alter the input images.

## 4 Batch Macros

The Batch Process command provides a powerful tool for processing many images. However it can be useful to perform more selective processing to images. For example process only certain images in a directory or apply different parameters depending on the image name.

More control over the batch processing of images requires writing custom macros in the ImageJ Macro Languge. The language has hundreds of built-in commands and custom commands can be defined. Full details of the ImageJ Macro Language can be found here:

http://rsbweb.nih.gov/ij/developer/macro/macros.html

The following sections contain some examples of batch macros. In each case the main batch code is provided and a place-holder has been added to indicate where to put your macro code:

```
// INSERT MACRO HERE
```

You should be able to insert your recorded macro commands into the place-holder and run the batch macro.

## 4.1 Process Files in a Directory

The following macro code provides a macro version of the BATCH PROCESS command:

```
dir1 = getDirectory("Choose Source Directory ");
format = getFormat();
dir2 = getDirectory("Choose Destination Directory ");
list = getFileList(dir1);
setBatchMode(true);
for (i=0; i<list.length; i++) {
    showProgress(i+1, list.length);
    open(dir1+list[i]);

    // INSERT MACRO HERE

    if (format=="8-bit TIFF" || format=="GIF")
       convertTo8Bit();
    saveAs(format, dir2+list[i]);
    close();
}

function getFormat() {
    formats = newArray("TIFF", "8-bit TIFF", "JPEG", "GIF", "PNG",
       "PGM", "BMP", "FITS", "Text Image", "ZIP", "Raw");
    Dialog.create("Batch Convert");
    Dialog.addChoice("Convert to: ", formats, "TIFF");
    Dialog.show();
    return Dialog.getChoice();
}

function convertTo8Bit() {
    if (bitDepth==24)
        run("8-bit Color", "number=256");
    else
        run("8-bit");
}
```

The above code prompts for an output directory and the format required. If the output format is set to TIFF the code is much simpler:

```
dir1 = getDirectory("Choose Source Directory ");
dir2 = getDirectory("Choose Destination Directory ");
list = getFileList(dir1);
setBatchMode(true);
for (i=0; i<list.length; i++) {
    showProgress(i+1, list.length);
    open(dir1+list[i]);

    // INSERT MACRO HERE

    saveAs("TIFF", dir2+list[i]);
    close();
}
```

It is possible to extend the above stub code with specific functionality. For example to process only the .tif files use the following code (new code is shown in **green**):

```
dir1 = getDirectory("Choose Source Directory ");
dir2 = getDirectory("Choose Destination Directory ");
list = getFileList(dir1);
setBatchMode(true);
for (i=0; i<list.length; i++) {
    showProgress(i+1, list.length);
    filename = dir1 + list[i];
    if (endsWith(filename, "tif")) {
        open(filename);

        // INSERT MACRO HERE

        saveAs("TIFF", dir2+list[i]);
        close();
    }
}
```

Note that when a open parenthesis is used to define a block of code it must be closed. This is why the **{** and **}** are marked as new code.

## 4.2 Process Open Images

It may be desired to run a command on all the currently open images. Since the command may open new result images this must be achieved by building a list of the images and then processing each one.

The following code provides a stub for processing all the currently open images:

```
setBatchMode(true);
imgArray = newArray(nImages);
for (i=0; i<nImages; i++) {
  selectImage(i+1);
  imgArray[i] = getImageID();
```

```
}

//now we have a list of all open images, we can work on it:

for (i=0; i< imgArray.length; i++) {
  selectImage(imgArray[i]);

  // INSERT MACRO HERE

}
```