Check for updates

SOFTWARE TOOL ARTICLE

REVISED Easing batch image processing from OMERO: a new toolbox for ImageJ [version 2; peer review: 2 approved]

Pierre Pouchin[1], Rayan Zoghlami[2], Rémi Valarcher[1], Maxence Delannoy[3], Manon Carvalho[3], Clémence Belle[3], Marc Mongy[4], Sophie Desset[1]*, Frédéric Brau[2]*

[1]GReD, CNRS, INSERM, Université Clermont Auvergne, Clermont-Ferrand, France
[2]Université Côte d'Azur, CNRS, IPMC, Valbonne, France
[3]Polytech Nice Sophia, Campus SophiaTech, Sophia Antipolis, France
[4]Univ. Lille, CNRS, Inserm, CHU Lille, Institut Pasteur de Lille, U1019 - UMS 9017 - CIIL - Center for Infection and Immunity of Lille, Lille, 59000, France

* Equal contributors

v2 **First published:** 05 Apr 2022, **11**:392
https://doi.org/10.12688/f1000research.110385.1

Latest published: 12 Sep 2022, **11**:392
https://doi.org/10.12688/f1000research.110385.2

Abstract
The Open Microscopy Environment Remote Objects (OMERO) is an open-source image manager used by many biologists to store, organize, view, and share microscopy images, while the open-source software ImageJ/Fiji is a very popular program used to analyse them. However, there is a lack of an easy-to-use generic tool to run a workflow on a batch of images without having to download them to local computers, and to automatically organize the results in OMERO. To offer this functionality, we have built (i) a library in Java: "Simple OMERO Client", to communicate with an OMERO database from Java software, (ii) an ImageJ/Fiji plugin to run a macro-program on a batch of images from OMERO and (iii) a new set of Macro Functions, "OMERO Macro extensions", dedicated to interact with OMERO in macro-programming. The latter is intended for developers, with additional possibilities using tag criteria, while the "Batch OMERO plugin" is more geared towards non-IT scientists and has a very easy to use interface. Each tool is illustrated with a use case.

Keywords
OMERO, Fiji, ImageJ, Java, Image Analysis, Image processing, Automation, Microscopy

This article is included in the NEUBIAS - the Bioimage Analysts Network gateway.

Open Peer Review

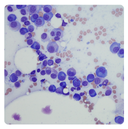
Approval Status ✓ ✓

	1	2
version 2 (revision) 12 Sep 2022		
version 1 05 Apr 2022	✓ view	✓ view

1. **Jason R. Swedlow**, University of Dundee, Dundee, UK

2. **Caterina Strambio-De-Castillia**, University of Massachusetts Medical School, Worcester, USA

Any reports and responses or comments on the article can be found at the end of the article.



This article is included in the Cell & Molecular Biology gateway.

Corresponding authors: Pierre Pouchin (pierre.pouchin@uca.fr), Sophie Desset (sophie.desset@uca.fr), Frédéric Brau (brau@ipmc.cnrs.fr)

Author roles: Pouchin P: Conceptualization, Investigation, Methodology, Resources, Software, Supervision, Validation, Writing – Original Draft Preparation; **Zoghlami R**: Investigation, Software, Validation; **Valarcher R**: Investigation, Software, Validation; **Delannoy M**: Investigation, Software, Validation; **Carvalho M**: Investigation, Software, Validation; **Belle C**: Investigation, Software, Validation; **Mongy M**: Validation; **Desset S**: Funding Acquisition, Supervision, Validation, Writing – Original Draft Preparation; **Brau F**: Conceptualization, Methodology, Project Administration, Resources, Software, Validation, Writing – Original Draft Preparation

Competing interests: No competing interests were disclosed.

Grant information: The Université Côte d'Azur and EMBRC-France OMERO Database deployment was financially supported by the "Complexity and diversity of living systems" Academie of Excellence established within the framework of the UCAJEDI IdEx program. This work has been supported by the French government, through the UCAJEDI Investments in the Future project managed by the National Research Agency (ANR) with the reference number ANR-15-IDEX-01. The work of PP and SD was supported by CNRS, INSERM, Université Clermont Auvergne (UCA), 16-IDEX-0001 CAP 20-25 challenge 1, and the COST-Action INDEPTH (CA16212), the work of RV was supported by Pack Ambition Recherche project Noyau-HD from the Region Auvergne Rhone Alpes.
The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

How to cite this article: Pouchin P, Zoghlami R, Valarcher R *et al.* **Easing batch image processing from OMERO: a new toolbox for ImageJ [version 2; peer review: 2 approved]** F1000Research 2022, **11**:392 https://doi.org/10.12688/f1000research.110385.2

First published: 05 Apr 2022, **11**:392 https://doi.org/10.12688/f1000research.110385.1

REVISED **Amendments from Version 1**

This revised version is a modified version giving complementary and missing information about previous works of the OMERO community and discussing our development choices regarding these approaches through the Introduction and Implementation chapters. We also attempted to clarify the terminology between the three developments mentioned as tools in this new toolbox, particularly the distinction between the library which is the support of the conjoint development of the OMERO Macro extensions and Batch OMERO plugins, which are new ImageJ/Fiji modules giving access respectively to a new set of new set of Macro Functions for macro programming; and a specific Graphical User Interface to ease non-IT batch image analysis on OMERO.

Any further responses from the reviewers can be found at the end of the article

Introduction

Cell biology research is a big provider of multidimensional image data through the use of multimodal microscopy approaches to decipher cellular processes. Photonic microscopes and associated areas of expertise in image analysis are often available in cellular imaging facilities which tend to propose unified tools to manage images and associated projects. Over the last decade many tools emerged which are in constant development: software platforms (QuPath,[1] Cell Profiler,[2] Icy,[3] KNIME,[4] napari[5]), browser-based and collaborative frameworks (ImJoy,[6] BIAFLOWS,[7] TissUUmaps[8]), and image databases (BisQUE,[9] Cytomine[10]). While some of these programs are focused essentially on the analysis of images and others on their management, the global trend for image analysis in the machine and deep learning era is to use complementary software or platforms integrating a combination of these tools. In this ecosystem, the Open Microscopy Environment (OME) Remote Objects (OMERO),[11] and ImageJ/Fiji[12–14] benefit from a high level of historical implementation in the microscopy landscape involving a large community of users and developers.

OMERO is a complete platform designed for managing (organizing, editing, analysing and sharing) images online through standalone clients (OMERO.insight) and dedicated web interfaces, including a viewer for full multi-dimensional image display, a figure editor, analysis with internal scripts and an integrated data mining tool. Over 150 proprietary microscopy image formats are supported. The organisation, browsing and searching of images is eased by multiple cataloguing tools, as data can be annotated with tags, comments, key-value pairs, tables, and supplementary files.

To analyse images, OMERO also provides Application Programming Interfaces (APIs) to let developers interact with a server from programs in Java, Python, MATLAB and C++ and thus many image analysis software packages can connect to OMERO via dedicated links or plugins.[1,2,15,16] Among these, the official OMERO plugin for ImageJ/Fiji allows access to the OMERO.insight interface to open and treat an image on local desktops before saving regions of interest (ROIs), results tables and images in the database after a manual or semi-manual sequence of treatment, and so does the bidirectional software bridge ImageJ-OMERO in ImageJ2. Although it is easy and common to automate such processing with Fiji via a macro for a folder of images, it is not yet possible to automate all the processes (import/export, saving images, ROIs, results, etc.) through a unique Macro program in ImageJ/Fiji for images hosted in the OMERO database. The only solutions available today are either to write and execute a script in ImageJ2/Fiji similar to the example provided in the OMERO guide for threshold segmentation on datasets, or using ImageJ-OMERO.

The goal of this work is to ease the access to image analysis for all users who are managing their projects and images in OMERO. Based on a new and exhaustive library for importing/exporting images and results from and to the OMERO database, named Simple OMERO Client, which mirrors similar efforts done in Python with *ezomero*, we propose two ways to interact with the OMERO database using the ImageJ Macro language. The first one is through a graphical user interface (GUI) on the same basis as the Batch Process module of ImageJ, which will loop a macro program written for one image on whole datasets. The other one is using new OMERO Macro functions to write a macro program that will loop the analysis on datasets. Both are developed using the aforementioned Java library, Simple OMERO client, which will also be described in this paper.

Methods

Implementation

Simple OMERO Client a library, and two plugins, OMERO Macro Extensions and batch OMERO plugin were built. These last two, are mentioned as "plugins" as they will be new ImageJ/Fiji modules/menus to get access to (i) a vocabulary extension in macro programming to interact with OMERO, or (ii) a specific GUI to batch image analysis from OMERO. All three were written in Java 8 and use Maven[17] to handle their dependencies. They all rely on *ImageJ*. The two plugins depend on the *Simple OMERO Client* library which was developed to wrap calls to the underlying *OMERO Java Gateway*, its main dependency.

Simple OMERO Client

Simple OMERO Client is a Java library that we developed to factor code that was often re-used when interacting with ImageJ1 in a few projects, such as methods to retrieve pixel values or ROI data. The ImageJ-OMERO plugin, which offers similar functionality, was not used as it was aimed at ImageJ2; and not compatible with OMERO versions greater than 5.4, until recently.

Our Maven project relies on *omero-gateway*, *omero-blitz* and *omero-model* to interact with an OMERO server, but it also depends on *formats-api* to handle microscope images locally. Finally, it uses *bio-formats_plugins*, *junit4* and *Jacoco* to run tests which should ensure that the library functions as intended. These tests, however, require a local OMERO server and are ideally run through *omero-test-infra*, as is the case during the continuous integration (CI) process on the project GitHub repository, where the sources and compiled JAR can be found. The CI also uses SonarCloud and Codecov for code analysis (coverage, quality).

This library often wraps simple calls to the underlying OMERO API, but does contain complex blocks that would otherwise need to be copied to every project, such as:

- handling key/value pairs or folders,
- retrieving pixels from OMERO,
- converting ROIs between OMERO and ImageJ,
- converting ImageJ results to OMERO tables.

Most OMERO data structures are simply wrapped (Facade pattern), although a Template method pattern was also used to reinstate inheritance like in the OME model and factor common methods shared by several classes when possible (such as shapes, annotations, or hierarchy objects).

OMERO Macro Extensions

The *OMERO Macro Extensions* set of macro functions only depends on *ImageJ* and *Simple OMERO Client* to run. It relies on *junit5* and *Jacoco* for unit tests, although some functions rely on graphical elements (such as the ROI manager) and are not currently automatically tested. The CI process is similar to what is done for *Simple OMERO Client*, except code analysis which is not performed through SonarCloud or Codecov. This plugin consists of a single class implementing the *MacroExtension* interface from ImageJ: it defines 22 macro functions to interact with OMERO and acts as a front for Simple OMERO Client. To do that, it essentially parses String and Long arguments before it calls the appropriate methods from the underlying library.

Batch OMERO plugin

The *batch OMERO plugin* depends on *ImageJ* and *Simple OMERO Client* as well as *formats-api* and *bio-formats_plugins* to open images from local files. It also, optionally, relies on *scijava-common* and *scijava-ui-swing* to handle script inputs and other script languages, if possible. Currently, no automatic testing is performed and there is no CI. The main plugin window handles the connection to OMERO, displays the objects and collects the input/output while a different class is responsible for effectively running the script on all the images from the selected source and saving the results. A specific class handles the execution of the script file and collects the possible arguments beforehand: if SciJava is available, it will be used, otherwise it will fall back on ImageJ1 functions.

Operation

To operate these tools, respective Maven dependencies need to be available, as well as a Java Virtual Machine (JVM). In practice, the OMERO dependencies can be provided by the OMERO.insight plugin for ImageJ, or better yet, by the OMERO-5.5-5.6 Fiji update site, while the bio-formats dependencies are provided by the corresponding plugin (included in Fiji).

Simple OMERO Client

The library will normally be used by developers: the easiest way is to add it as a Maven dependency to the project, as it was done by the ImageJ plugins presented in this paper. If the aim is to use an uber-JAR (including the dependencies) through another language (e.g. Python), the code just needs to be built with Maven: this will produce the desired file. This file can also be downloaded from the GitHub packages for this repository. When interacting with ImageJ, it is possible to create tables on OMERO from ImageJ results. The library also makes it possible to transfer ROIs between OMERO and ImageJ. However, as the latter only works with 2D shapes while the former handles 4D data, additional metadata are required to track which shapes belong to the same ROI. The library expects it to be done using a property in ImageJ: shapes that share the same local index for the specified key correspond to the same ROI in OMERO. Moreover, when ROIs are retrieved from OMERO, a second property, with "_ID" appended, is set with the OMERO ID as its value. Finally, tables created from ImageJ results can have a ROI column linking each line to an ROI if all the lines fulfil one of the following conditions:

- A column with the same name as the property key contains the corresponding value, and the ROI has an ID property.
- A column with the same name as the ID property contains the ROI ID.
- The label contains the name of an ImageJ ROI with those properties set.

OMERO Macro Extensions

Once the OMERO Macro Extensions plugin is installed in ImageJ, along with its dependencies, it can be used through the macro language. When writing a macro using these extensions, the first thing to do is to load the plugin with the following command: `run("OMERO Extensions");`

Connecting to OMERO is done using: `Ext.connectToOMERO("host", 4064, "username", "password");`

Then, switching group can be performed through: `Ext.switchGroup(groupId);`

Afterwards, interacting with OMERO only takes simple instructions, such as:

```
datasets = Ext.list("datasets");
```

or:

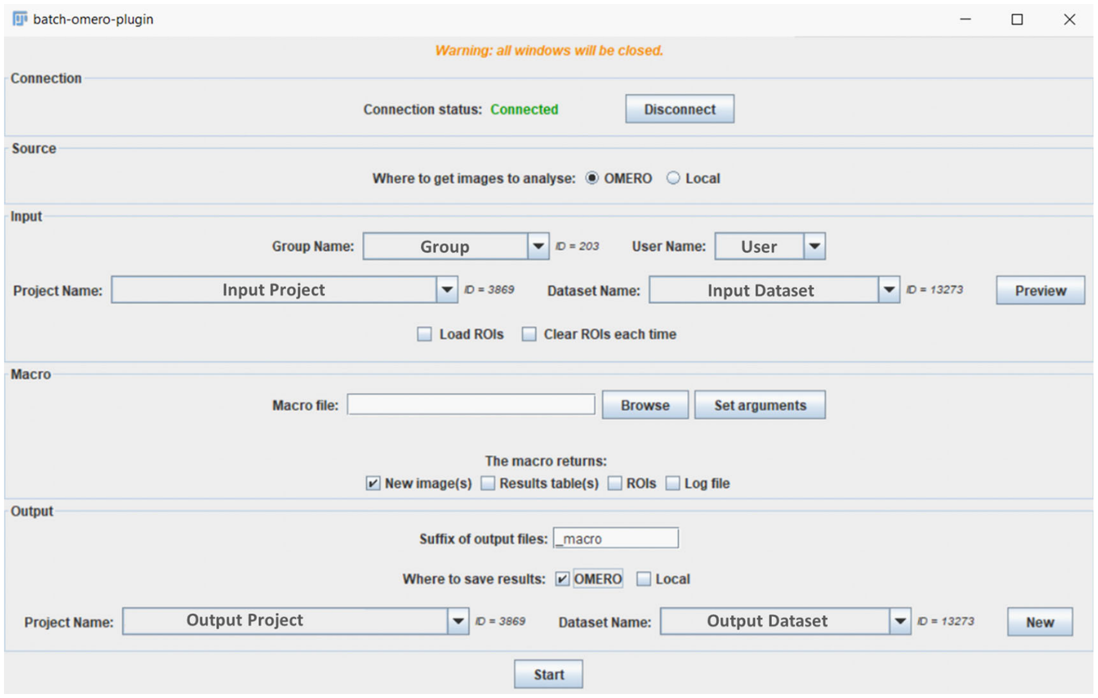
```
imageplusID = Ext.getImage(imageId);
```

When done, you can disconnect with: `Ext.disconnect();`

Batch OMERO plugin

In the same way, the batch OMERO plugin needs to be installed in the ImageJ plugins folder along with its dependencies. If this plugin is installed in ImageJ2/Fiji, it will make use of SciJava to run scripts using script parameters, otherwise it will only run ImageJ1 macro files, with arguments specified manually. When the plugin and the Simple OMERO Client are downloaded and installed, and once ImageJ/Fiji is launched, a new "Batch process…" item is added in the OMERO plugin menu. When chosen, the GUI (Figure 1) is opened. A connection window to OMERO appears by clicking « Connect » in this window. Once connected, the drop-down menus will be filled with information coming from the default group of the User (Group, User, Project, Dataset). Depending on the saving options (OMERO or local), the window will adapt its output choices.

The tool can process images retrieved remotely from OMERO or locally from a folder, using Bio-Formats. Conversely, it can save the output on OMERO and/or locally. The main possible outputs are new images, ROIs, results tables, or log windows and the user has to specify (Figure 1) what should be saved in accordance with the outputs of the macro.



batch-omero-plugin

Warning: all windows will be closed.

Connection

Connection status: Connected — Disconnect

Source

Where to get images to analyse: ● OMERO ○ Local

Input

Group Name: Group ID = 203 — User Name: User

Project Name: Input Project ID = 3869 — Dataset Name: Input Dataset ID = 13273 — Preview

☐ Load ROIs ☐ Clear ROIs each time

Macro

Macro file: — Browse — Set arguments

The macro returns:

☑ New image(s) ☐ Results table(s) ☐ ROIs ☐ Log file

Output

Suffix of output files: _macro

Where to save results: ☑ OMERO ☐ Local

Project Name: Output Project ID = 3869 — Dataset Name: Output Dataset ID = 13273 — New

Start

Figure 1. GUI window of the batch OMERO plugin.

ROIs can be loaded from OMERO: if this option is chosen (Figure 1), then OMERO ROIs will be exported to the ROI Manager using Simple OMERO Client. 3D/4D ROIs can thus be accessed from macros through two ROI properties: "ROI" and "ROI_ID", which contain, respectively, the local index and the ROI ID on OMERO for each 2D shape.

When saving to OMERO, the users have to choose an existing project or a dataset they own in the current group.

Furthermore, the following rules apply:

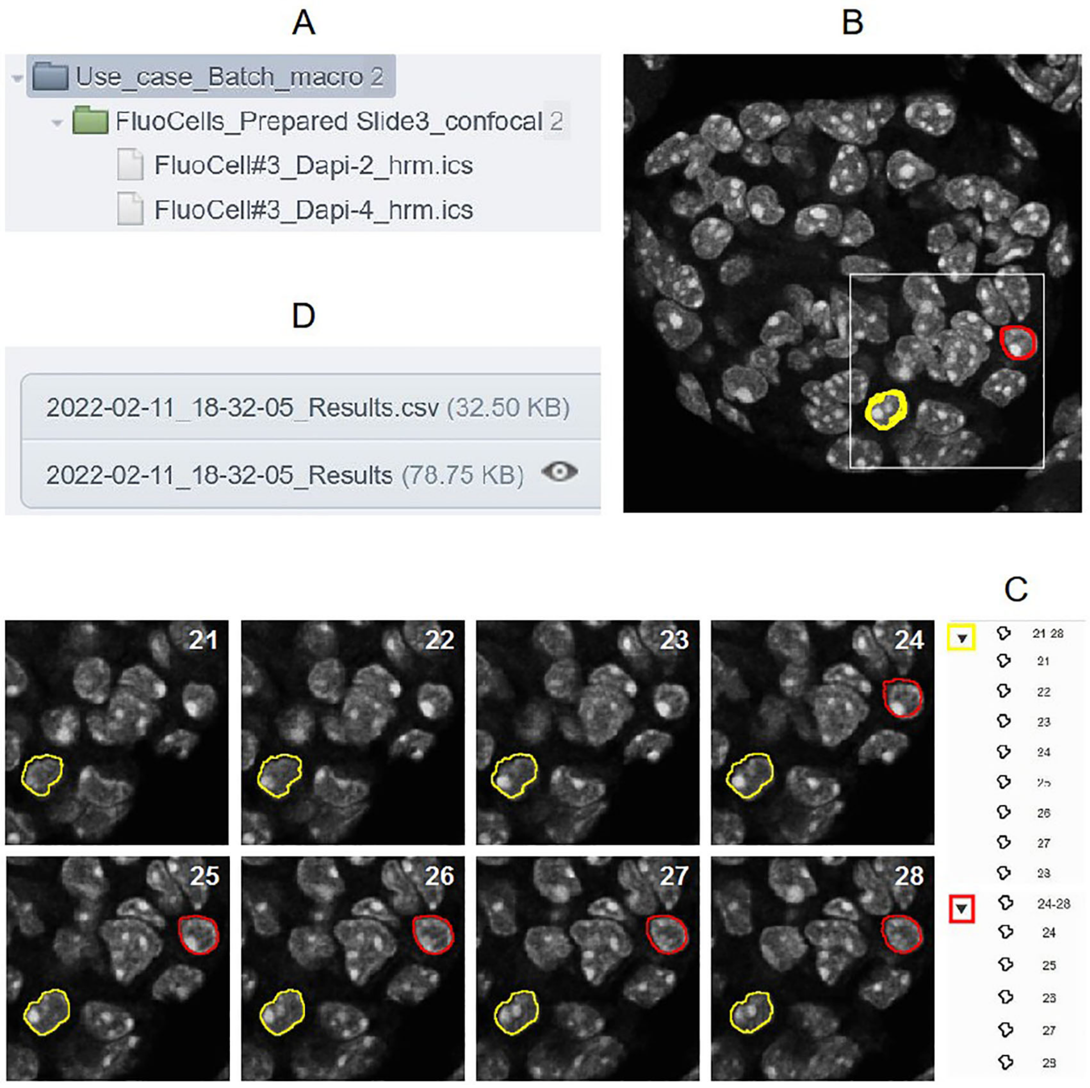
- If ROIs are saved but images are not, then they are saved to the input image on OMERO, which should be annotatable by the user.
- If images and ROIs are saved, then:
  - For each image, its overlay is imported as well.
  - The last active image obtained with the macro-processing gets the content of the ROI Manager, and the Results tables and log windows will be its associated files.
  - If the last active image is the same than the input image but it cannot be annotated by the current user, then the image is re-imported for the user.
- If tables are saved to a project, and ROIs were loaded or saved, the tables can have a ROI column, provided they fulfil the requirements from the simple-omero-client library mentioned previously.

Use cases

Batch OMERO plugin

As 2D or 3D segmentation is a general requirement to perform quantification in cellular biology, a use case based on this image analysis procedure is proposed to show all the possible outputs obtained with these tools. So, the batch OMERO plugin and OMERO Macro extensions were tested on a XYZ set of DAPI images obtained from a FluoCells Prepared Slide 3 (mouse kidney section with Alexa Fluor 488 WGA, Alexa Fluor 568 Phalloidin, and DAPI) from Thermo Fisher Scientific. Images were acquired on a LSM780 laser scanning confocal (Carl Zeiss, France) through a 63X/1.4 oil immersion objective (excitation 405 nm, emission 430-460 nm, voxel size 130×130×200 nm) and a blind deconvolution was applied using Huygens Remote Manager with a CMLE algorithm (Scientific Volume Imaging, Netherlands). Two macros are available to try the Batch OMERO plugin:

- "Macro_to_Batch_onOmero_3D": this program performs 3D segmentation on the image stack of nuclei using the 3D object counter plugin[18] after low pass filtering and Otsu thresholding. Then ROI groups are created by gathering objects which have the same label on different slices. At the end, the macro returns the image of labels overlaid with the ROIs, a results Table (Figure 2) and a log window. It can be run by ImageJ1 or ImageJ2/Fiji: in the former case default parameters will be used (predefined minimal sizes of objects, and all images kept at the end). In the latter, as it also uses the script parameters of ImageJ2, the minimal size of the objects can be defined by the user who can decide if images are kept or not at the end of the execution of the macro. These inputs can be displayed and modified through the GUI of the plugin with the "Set Arguments" button. This macro is the generic one called by the next one and the use case macro of OMERO Macro extensions.



Use_case_Batch_macro 2
FluoCells_Prepared Slide3_confocal 2
FluoCell#3_Dapi-2_hrm.ics
FluoCell#3_Dapi-4_hrm.ics

2022-02-11_18-32-05_Results.csv (32.50 KB)
2022-02-11_18-32-05_Results (78.75 KB)

Figure 2. A) Project and Dataset processed by the "Macro_to_Batch_onOmero_3D" B) First image with 2 groups of ROIs displayed (red and yellow) C) View of each ROI of these groups on the different z positions of the 3D stack D) csv file and Table attached to the Project containing all the ROI measurements for all the images of the dataset processed.

- "Macro_to_Batch_onOmero_3D_IJ1_Arguments": this macro is only coded in ImageJ1 macro language to get the parameters and calls the previous one. Its goal is to show another way to get input parameters. Indeed, the batch OMERO plugin calls the macro each time an image is opened from a dataset. If it is an ImageJ1 macro containing a dialog box to get input parameters, this one will be displayed at each execution of the macro, by default. This problem can be circumvented by getting the number of times the macro is called by the plugin with the `getArguments()` command. The dialog box will be displayed at the first call and parameters stored in a text file. They will be retrieved from the file at the next calls.

OMERO Macro Extensions

In this use case, the macro language extension for OMERO is used in the macro program "Test_langage_extensions_runMacro_alltags" to show another way to access and process the images through their tags. Two dialog boxes are displayed during the execution. The first one allows the user to log in and define a specific signature keyword to tag the images that will be processed ("IJ_Processed" by default). The second one contains a drop-down menu to choose the images that will be processed for 3D segmentation according to their tag, among the tags used in the default user group. If the signature tag already exists, the macro tests if it is linked to the image. If this condition is fulfilled, the image will not be processed, otherwise it will be linked to the image after processing. When the signature tag is unknown in the user group, it is created and linked to the processed image.

The plugin also includes macro templates that are conversions of some of the Groovy scripts provided in the *omero-guide-fiji* repository and illustrate the possibilities offered by the plugin.

Simple OMERO Client

The expected use case for the library is to use it as a Maven dependency in a Java project, as demonstrated by the two ImageJ plugins presented in this paper. Another use case would be to call it from an ImageJ2 script to access advanced OMERO functions directly from there. It is, for example, short and easy to retrieve the maximum value from images inside a dataset and tag them if the value is between two set thresholds (script available in OMERO toolbox examples).

Conclusion

In this paper we provide new tools to facilitate automatic image processing with ImageJ/Fiji on images managed through an OMERO database. One of the main advantages of these tools is to simplify the macro-programming and give the opportunity to quickly analyse multiple images. Among these advantages, avoiding the tedious image format management when coding Macro programs in ImageJ/Fiji, will accelerate Macro development for end-users and help focus on the essential goal: image analysis. The spirit of the plugin could be considered close to the Batch Process plugin integrated in ImageJ1, as people without any knowledge with Macro language could record their image analysis process and apply it after on all their images. Consequently, we should highlight, that our plugin potentiates the original Batch Process by offering the choice to alternatively work with local folders, like the original one, or the files could be imported/exported from local directories or the OMERO database. The plugin and language extension approaches can be considered complementary, as the language extension will allow more complex processing, including a faster connection without GUI, a tag management of the images for example, allowing selective analyses. Another big advantage will be the delivery of CSV files and tables associated with the Project, which allows the use of the OMERO.parade data mining tool, and particularly the recent parade-crossfilter development.

Data availability

Underlying data

No underlying data are associated with this article.

Extended data

The Dataset of images dedicated to these treatments and processed with the "Macro_to_Batch_onOmero_3D", through the Batch OMERO Plugin are available, alongside their results, on the public webpage of the OMERO database from Université Côte d'Azur and EMBRC-France, managed by the "Microscopie Imagerie Côte d'Azur" (MICA) Facility and housed by "Institut Français de Bioinformatique" from: https://bioimage.france-bioinformatique.fr/omero-mica/webclient/?show=project-3006.

Software availability

- Simple OMERO Client:
    - Software and source code available from: https://github.com/GReD-Clermont/simple-omero-client
    - Archived source code at time of publication: https://doi.org/10.5281/zenodo.6320867
    - License: GPLv2+
- OMERO Macro Extensions:
    - Software and source code available from: https://github.com/GReD-Clermont/omero_macro-extensions
    - Archived source code at time of publication: https://doi.org/10.5281/zenodo.6320876
    - License: GPLv2+
- OMERO Batch plugin:
    - Software and source code available from: https://github.com/GReD-Clermont/omero_batch-plugin
    - Archived source code at time of publication: https://doi.org/10.5281/zenodo.6367840
    - License: GPLv2+
- OMERO toolbox examples:
    - Software and source code available from: https://github.com/GReD-Clermont/omero-toolbox-examples/tree/1.0.1
    - Archived source code at time of publication: https://doi.org/10.5281/zenodo.6367851
    - License: MIT

Acknowledgements
We thank Carole Rovere for providing the initial use case of this project in the context of a co-mentored internship. We thank Christophe Blanchet from Institut Français de Bioinformatique for the Université Côte d'Azur and EMBRC-France OMERO Database administration. We are grateful to the Mésocentre Clermont Auvergne University and AuBi platform for providing computing and storage resources. We also thank Sophie Abélanet for her expertise with OMERO.Figure, helpful for this manuscript.

References

1. Bankhead P, Loughrey MB, Fernández JA, *et al.*: **QuPath: Open source software for digital pathology image analysis.** *Sci. Rep.* 2017; **7**(1): 1–7.
2. McQuin C, Goodman A, Chernyshev V, *et al.*: **CellProfiler 3.0: Next-generation image processing for biology.** *PLoS Biol.* 2018; **16**(7): e2005970.
PubMed Abstract | Publisher Full Text
3. De Chaumont F, Dallongeville S, Chenouard N, *et al.*: **Icy: an open bioimage informatics platform for extended reproducible research.** *Nat. Methods.* 2012; **9**(7): 690–696.
PubMed Abstract | Publisher Full Text
4. Dietz C, Berthold MR: **Knime for Open-Source bioimage analysis: a tutorial.** *Focus on Bio-Image Informatics.* 2016: 179–197.
PubMed Abstract | Publisher Full Text
5. Napari Contributors: **napari: a multi-dimensional image viewer for python. Zenodo 10.5281/zenodo.** 2019; 3555620.
6. Ouyang W, Mueller F, Hjelmare M, *et al.*: **Imjoy: an open-source computational platform for the deep learning era.** *Nat. Methods.* 2019; **16**(12): 1199–1200.
PubMed Abstract | Publisher Full Text
7. Rubens U, Mormont R, Paavolainen L, *et al.*: **Biaflows: A collaborative framework to reproducibly deploy and benchmark bioimage analysis workflows.** *Patterns.* 2020; **1**(3):

100040.
PubMed Abstract | Publisher Full Text

8. Solorzano L, Partel G, Wählby C: **Tissuumaps: Interactive visualization of large-scale spatial gene expression and tissue morphology data.** *Bioinformatics.* 2020; **36**(15): 4363–4365.
PubMed Abstract | Publisher Full Text

9. Kvilekval K, Fedorov D, Obara B, *et al.*: **Bisque: a platform for bioimage analysis and management.** *Bioinformatics.* 2010; **26**(4): 544–552.
PubMed Abstract | Publisher Full Text

10. Rubens U, Hoyoux R, Vanosmael L, *et al.*: **Cytomine: toward an open and collaborative software platform for digital pathology bridged to molecular investigations.** *Proteomics Clin. Appl.* 2019; **13**(1): 1800057.
PubMed Abstract | Publisher Full Text

11. Allan C, Burel J-M, Moore J, *et al.*: **William J Moore, Carlos Neves, Andrew Patterson, et al. Omero: flexible, model-driven data management for experimental biology.** *Nat. Methods.* 2012; **9**(3): 245–253.
PubMed Abstract | Publisher Full Text

12. Schneider CA, Rasband WS, Eliceiri KW: **Nih image to imagej: 25 years of image analysis.** *Nat. Methods.* 2012; **9**(7): 671–675.
PubMed Abstract | Publisher Full Text

13. Schindelin J, Arganda-Carreras I, Frise E, *et al.*: **Fiji: an open-source platform for biological-image analysis.** *Nat. Methods.* 2012; **9**(7): 676–682.
PubMed Abstract | Publisher Full Text

14. Schroeder AB, Dobson ETA, Rueden CT, *et al.*: **The imagej ecosystem: Open-source software for image visualization, processing, and analysis.** *Protein Sci.* 2021; **30**(1): 234–249.
PubMed Abstract | Publisher Full Text

15. Stritt M, Stalder AK, Vezzali E: **Orbit image analysis: an open-source whole slide image analysis tool.** *PLoS Comput. Biol.* 2020; **16**(2): e1007313.
PubMed Abstract | Publisher Full Text

16. Berg S, Kutra D, Kroeger T, *et al.*: **Ilastik: interactive machine learning for (bio) image analysis.** *Nat. Methods.* 2019; **16**(12): 1226–1232.
PubMed Abstract | Publisher Full Text

17. Miller FP, Vandome AF, McBrewster J: *Apache Maven.* Alpha Press; 2010.

18. Bolte S, Cordelières FP: **A guided tour into subcellular colocalization analysis in light microscopy.** *J. Microsc.* 2006; **224**(3): 213–232.
PubMed Abstract | Publisher Full Text

Open Peer Review

Current Peer Review Status: ✓ ✓

Version 1

Reviewer Report 06 June 2022

https://doi.org/10.5256/f1000research.121985.r130101

✓

Caterina Strambio-De-Castillia
Program in Molecular Medicine, University of Massachusetts Medical School, Worcester, MA, USA

GENERAL NOTES

The authors describe three software tools they have developed to facilitate the batch analysis of images that are stored and managed using the popular OMERO data management platform.

The explicit goal described by the authors "to ease the access to image analysis for all users who are managing their projects and images in OMERO." is definitely worthwhile and the work is timely especially because similar work to connect ImageJ with OMERO (https://github.com/imagej/imagej-omero) has not been maintained and cannot be used with recent versions of OMERO.

SPECIFIC COMMENTS

<u>LANDSCAPE ANALYSIS</u>:

Given the potential impact and general interest of the topic and the existence of similar ongoing efforts in the community, in order to avoid confusion it is important that the authors compare their work with other tools and address the following questions:

1. Were previous efforts utilized as a starting point?

2. If not why not?

3. How are their tools better than those produced by other efforts?

<u>CLARITY:</u>

We suggest that the authors might consider revising the text to increase clarity and the potential impact of their work on non-IT users.

Specifically, the authors should consider that terms that are routinely used by software engineers such as "application", "tool", "plugin" "API" and "library" are often very confusing to biomedical researchers who are often non-computer experts.

For example, the text in the Abstract and Introduction is a bit confusing regarding the number of tools that were developed. Below I list the primary sources of confusion in the text:

1. Line 8 of the Abstract states "we have built three tools".

2. The description in the Abstract describes three tools, called: Simple OMERO Client (a library); OMERO Macro extensions (a plugin); and Batch OMERO plugin (a plugin).

3. The last line of the Abstract states "Both tools are illustrated..."

4. The last paragraph of the Introduction describes two methods to interact with OMERO.

5. The Methods section states "Three tools were built..."

While I understand that some of the confusion arises from the fact that one of the tools is a library and the others are plugins, the authors might want to consider revising the text to ensure that biomedical researchers who are not fluent in "computerese" can understand the importance of their work.

Is the rationale for developing the new software tool clearly explained?

Partly

Is the description of the software tool technically sound?

Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?

Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?

Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?

Partly

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Cell Biology, Microscopy, Metadata Modelling, Data Management, Imaging Science

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Author Response 08 Aug 2022

Frédéric BRAU, Université Côte d'Azur, CNRS, IPMC, Valbonne, France

LANDSCAPE ANALYSIS:
Given the potential impact and general interest of the topic and the existence of similar ongoing efforts in the community, in order to avoid confusion it is important that the authors compare their work with other tools and address the following questions:
Were previous efforts utilized as a starting point?
This work mixes two different axes of development: from the language part, a dedicated library developed to ease Java interaction with OMERO commands on a specific image analysis problem (NucleusJ); and from the GUI part of the batch OMERO plugin and modified groovy script from OMERO guide integrating GUI interfaces. The former library was then used to ease the Java plugins development, including the migration from groovy to Java for the batch OMERO plugin.

If not why not?
How are their tools better than those produced by other efforts?
This tool addresses the problem from the ImageJ/Fiji point of view, to link OMERO which is more detailed in the introduction. Previous links between this database and third-party software and languages exists, as also mentioned in the manuscript. The closest approaches are dedicated groovy scripts available in OMERO guide or using the ImageJ-OMERO plugin.

CLARITY:
We suggest that the authors might consider revising the text to increase clarity and the potential impact of their work on non-IT users. Specifically, the authors should consider that terms that are routinely used by software engineers such as "application", "tool", "plugin" "API" and "library" are often very confusing to biomedical researchers who are often non-computer experts.

For example, the text in the Abstract and Introduction is a bit confusing regarding the number of tools that were developed. Below I list the primary sources of confusion in the text:
Line 8 of the Abstract states "we have built three tools".
The description in the Abstract describes three tools, called: Simple OMERO Client (a library); OMERO Macro extensions (a plugin); and Batch OMERO plugin (a plugin).
The last line of the Abstract states "Both tools are illustrated..."
The last paragraph of the Introduction describes two methods to interact with OMERO.
The Methods section states "Three tools were built..."
While I understand that some of the confusion arises from the fact that one of the tools is a library and the others are plugins, the authors might want to consider revising the text to ensure that biomedical researchers who are not fluent in "computerese" can understand the importance of their work.
We tempted to clarify the terminology through the updated version submitted, and these points to be more accurate with the denominations between the tools mentioning that there is a library, a new set of Macro Functions through the OMERO Macro extensions plugin and a Batch OMERO plugin. These last two are mentioned as plugins as they are

modules added to ImageJ/Fiji to get their functionalities, and they were developed using the library.

Competing Interests: No competing interests were disclosed.

Reviewer Report 17 May 2022

https://doi.org/10.5256/f1000research.121985.r129917

✓

Jason R. Swedlow
Division of Computational Biology, Centre for Gene Regulation and Expression, School of Life Sciences, University of Dundee, Dundee, UK

The authors introduce a set of tools to simplify access to an OMERO server from the ImageJ ecosystem (Java). Similar efforts are on-going in other programming languages e.g. https://github.com/TheJacksonLaboratory/ezomero (Python). This could maybe be mentioned to highlight the fact that the work of the authors is part of a general community effort and should be well received by scientists.

Other groups previously released work connecting OMERO and ImageJ, see https://github.com/imagej/imagej-omero. This work has not been updated in recent years to be compatible with newer versions of the OMERO.server. Have the authors considered such work based on ImageJ2 as a starting point? If not, it will be interesting to highlight why since scientists and other software e.g. KNIME used it in some versions of their applications.

From a technical point of view:

The examples provided have been tested and the various GitHub repositories checked.

- Simple OMERO client. The jar associated with the tag e.g. https://github.com/GReD-Clermont/simple-omero-client/releases/tag/5.9.0 will not allow the user to run the examples. There is no indication in the README of the dependencies required. They are mentioned in the paper but it is not indicated where they can be found. The name of the omero dependencies is omero-gateway-java and not omero-gateway. Installing the uber jar simple-omero-client-with-dependencies.jar, built from source, will allow us to run the examples exposed in the README. Authors should consider uploading the uber jar to the GitHub tag and should review the installation instructions in the paper and README.

- The Authors indicate that the simple-omero-client is built and compatible with Java 8. The GitHub action only uses Java 11. Java 8 should be added to the matrix of the GitHub action to ensure compatibility.

- https://github.com/GReD-Clermont/omero_macro-extensions and https://github.com/GReD-Clermont/omero_batch-plugin, the authors should consider depending on simple-omero-client-with-dependencies.jar and not the OMERO.insight plugin. The instructions written in the paper and the ones in the README in the GitHub repository do not match. The instructions in the paper will not work since the omero-* libraries used to connect to the OMERO.server are not mentioned. The OMERO.insight plugin jar listed in the README will install several dependencies not required by the plugins or macro-extensions e.g. UI dependencies. This should be considered for multiple reasons: size required, conflict of dependencies etc. The authors should consider mentioning simple-omero-client-with-dependencies.jar or (simple-omero-client.jar +omero-* dependencies). It will also allow the authors to control the exact version of the omero libraries e.g. the version of omero-gateway-java they want their application to depend on.

- Have the authors considered an ImageJ update site?

- Minor point:

  - The term Facade should be used in this context and not Façade

  - The authors should indicate that the macro extensions examples are a conversion of some of the groovy scripts, used for training scientists, in https://github.com/ome/omero-guide-fiji/tree/master/scripts/groovy. Some examples in the omero-guide-fiji repository have been requested by the scientists themselves.

  - Code example: imageplusID = Ext.getImage (imageId);. Remove space between getImage and (

  - The term fluidize in the conclusion could be replaced with facilitate

Suggestion for follow-up work. If the package https://github.com/GReD-Clermont/simple-omero-client is going to supersede https://github.com/imagej/imagej-omero, the authors should consider implementing methods from https://github.com/imagej/imagej-omero to the new proposed package.

Jean-Marie Burel
Jason Swedlow

Is the rationale for developing the new software tool clearly explained?
Yes

Is the description of the software tool technically sound?
Partly

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?
Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?
Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?
Partly

Competing Interests: The reviewers are part of the OME Consortium which builds and releases OMERO. JRS founded and runs Glencoe Software which commercialises OME's tools, including OMERO.

Reviewer Expertise: microscopy, cell biology, mitosis, image informatics, public data resources

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Author Response 08 Aug 2022

Frédéric BRAU, Université Côte d'Azur, CNRS, IPMC, Valbonne, France

The authors introduce a set of tools to simplify access to an OMERO server from the ImageJ ecosystem (Java). Similar efforts are on-going in other programming languages e.g. https://github.com/TheJacksonLaboratory/ezomero (Python). This could maybe be mentioned to highlight the fact that the work of the authors is part of a general community effort and should be well received by scientists.
We haven taken into account these remarks and mentioned the ezomero and ImageJ-OMERO works done before, with a similar goal, in the Introduction.

Other groups previously released work connecting OMERO and ImageJ, see https://github.com/imagej/imagej-omero. This work has not been updated in recent years to be compatible with newer versions of the OMERO.server. Have the authors considered such work based on ImageJ2 as a starting point? If not, it will be interesting to highlight why since scientists and other software e.g. KNIME used it in some versions of their applications.
The library project started as a small project to wrap API calls when accessing and organizing data on OMERO from a Java program. However, as this software depended on ImageJ 1.x to open images, it soon became apparent that it would be easier to directly convert that data from OMERO to ImageJ than to create and populate basic Java objects from the server and then convert them to ImageJ. This was not the initial goal though, which is why the ImageJ-OMERO project was not considered during the project genesis. Moreover, the project focus was on making small and simple re-usable objects/methods for an IJ1 plugin: the ImageJ2 library looked impressive and using it as a starting point seemed too time-consuming considering the human resources at our disposal. In hindsight, it might have been a better choice for the community though. Merging both may still be an option, however.

Simple OMERO client. The jar associated with the tag e.g. https://github.com/GReD-Clermont/simple-omero-client/releases/tag/5.9.0 will not allow the user to run the examples. There is no indication in the README of the dependencies required. They are mentioned in the paper but it is not indicated where they can be found. The name of the omero dependencies is omero-gateway-java and not omero-gateway. Installing the uber jar simple-omero-client-with-dependencies.jar, built from source, will allow us to run the examples exposed in the README. Authors should consider uploading the uber jar to the GitHub tag and should review the installation instructions in the paper and README.
Indeed, the library has only been thought as a Maven dependency, which would be available to users if they installed the other plugins following the provided steps, so instructions to install only the library to ImageJ specifically are currently missing. The uber-jar exists because when the project started, before ezomero, the library was also used to interact with OMERO from Python using JPype, as well as other Java programs outside of ImageJ. However, using it in ImageJ may cause conflicts if the official OMERO.insight plugin is installed, unless dependencies are “shaded” and “relocated”, which is why it was not readily available. The aim will be to rely on the ImageJ-OMERO-5.5 update site in Fiji to provide the dependencies and, alternatively, the uber-jar for ImageJ1. Instructions will be updated when this solution will be operational.

The Authors indicate that the simple-omero-client is built and compatible with Java 8. The GitHub action only uses Java 11. Java 8 should be added to the matrix of the GitHub action to ensure compatibility.
The paper states the language level in which the library is written is 8, it is true though that the inference would be that it is therefore built and compatible with Java 8, when it only uses Maven properties to convey this information to the Java 11 compiler (source = 8, target = 8), which could fail in theory. However, as the build is done through omero-test-infra, it is the Dockerfile that defines which JDK is used for the build. It will be changed to 8 in 5.9.2. The GitHub action JDK will remain 11 though, as it is required for SonarCloud and not used for compilation.

https://github.com/GReD-Clermont/omero_macro-extensions and https://github.com/GReD-Clermont/omero_batch-plugin, the authors should consider depending on simple-omero-client-with-dependencies.jar and not the OMERO.insight plugin. The instructions written in the paper and the ones in the README in the GitHub repository do not match. The instructions in the paper will not work since the omero-* libraries used to connect to the OMERO.server are not mentioned. The OMERO.insight plugin jar listed in the README will install several dependencies not required by the plugins or macro-extensions e.g. UI dependencies. This should be considered for multiple reasons: size required, conflict of dependencies etc. The authors should consider mentioning simple-omero-client-with-dependencies.jar or (simple-omero-client.jar +omero-* dependencies). It will also allow the authors to control the exact version of the omero libraries e.g. the version of omero-gateway-java they want their application to depend on.
Instructions in the paper should indeed reflect what is in the README. Moreover, what is in the README may need adjustments, depending on the possible use of the library uber-jar. As mentioned previously, the procedure in the README (OMERO.insight plugin + library jar + plugin jars) was chosen to prevent conflicts between OMERO.insight and the library when it comes to dependencies.

Have the authors considered an ImageJ update site?
An update site has been considered but OMERO dependencies need to be properly handled beforehand. If the plugins can rely on the ImageJ-OMERO update site (and offer to activate it), then an update site for these JARs will be set up.

The Minor points have been considered and the corresponding corrections were applied in the new version of the manuscript.

Competing Interests: No competing interests were disclosed.