
OMERO guide Documentation

Release latest

Open Microscopy Environment

Jan 08, 2026

CONTENTS

1	Getting started	3
2	General	5
3	Prepare an OMERO server for training	7

The OMERO guide offers step-by-step instructions on how to use components of the OMERO platform.

To learn more about OMERO, we recommend that you first visit the [Open Microscopy Environment Website](#).

You can also watch videos of OMERO features or workshop recordings on the [Open Microscopy Environment YouTube channel](#).

Following the guides/videos linked below will help both with answering the “Why OMERO?” question from wider perspective and with organization of trainings, see [I3D:bio’s OMERO training materials](#).

GETTING STARTED

OMERO walkthrough example presents a step-by-step example of a typical OMERO user workflow from import to figure creation.

OMERO walkthrough for facility managers gives a typical workflow of a facility manager helping and guiding their users to use the local OMERO ecosystem.

GENERAL

The OMERO guide is divided into several parts:

General concepts covers how to get started, import, manage data, users, groups and run server-side OMERO scripts.

OMERO.web extensions to view data includes instructions for installing Web plugins and how to use those extension points e.g. OMERO.iviewer, OMERO.figure, OMERO.parade.

Add-ons for OMERO includes instructions for installing plugins for OMERO.insight and the Command Line Interface and how to use those extension points.

OMERO Application Programming Interfaces allows clients to be written in Java, Python, R, C++ or MATLAB. This section includes installation instructions and an exhaustive list of examples.

External Software and OMERO introduces how to analyze data using third party tools e.g. Fiji, CellProfiler, ilastik.

PREPARE AN OMERO SERVER FOR TRAINING

Prepare an OMERO server for training presents a step-by-step example of a setup needed to get a training server similar to the one used by the OME Team.

All the guides are hosted on GitHub. If you wish to create a new guide, check the instructions on *How to write a guide*. Click on the icon to go the relevant part of the guide.



Powered by





3.1 General concepts

Here we introduce general concepts about OMERO: image import, metadata upload, management, annotation, search and server-side scripts.

- *Upload data*

This section shows how to import data using the Desktop client, the Command Line Interface (CLI) and OMERO.dropbox. It also covers advanced import features and post-import processing steps used mainly in the context of the Image Data Resource (IDR).

- [download/docs/index](#)

The section shows the possibilities of getting images from OMERO. These can be described as “export” (new image, usually a jpeg, tiff or png is created from the original image saved in OMERO) or “download” (the original image, in the original format is downloaded from OMERO).

- *General Introduction*

This section covers the general concepts of OMERO. Those concepts are demonstrated using the Web client. This section introduces the data management functionalities of OMERO, shows how to annotate data and demonstrates how to search for data and metadata. Also, the management of groups and users is presented, both in the Web client as well as on the command line.

- *Server-side scripts*

This section describes how to run, write and manage the OMERO server-side scripts. Server-side scripts can be written in different programming languages but you will need to invoke the scripts using a Python wrapper.

3.1.1 Upload data

This section shows how to import data using the Desktop client, the Command Line Interface (CLI) and OMERO.dropbox. It also covers advanced import features and post-import processing steps used mainly in the context of the Image Data Resource (IDR).

Contents:

Import Image data

This section shows how to import data either using the Java Desktop client or the Command Line Interface (CLI).

Contents:

Import data using the Desktop Client

Description

In the first part, we first show how to import data by yourself and for yourself into OMERO using various import strategies. This will be mainly done using the OMERO.insight desktop client.

Second part of this import section will show how to import data for another user, using OMERO.insight. The user importing the data needs to have some admin or restricted-admin privileges. More information about restricted privileges can be found at <https://docs.openmicroscopy.org/latest/omero/sysadmins/restricted-admins.html>

The import for another user requires that the user doing the import has specific privileges. We will use the user importer1, this could be, for example, a facility manager.

We will show:

- How to install the OMERO.insight desktop client on Windows, Mac and Linux.
- How to import data for the user logged in using OMERO.insight.
- How to select a target Project and Dataset or create a new ones in OMERO for the imports.
- How to add Tags to imported images at the import stage, to facilitate the management of these images later in OMERO.server.
- How to import data for other users in OMERO.insight.

Setup

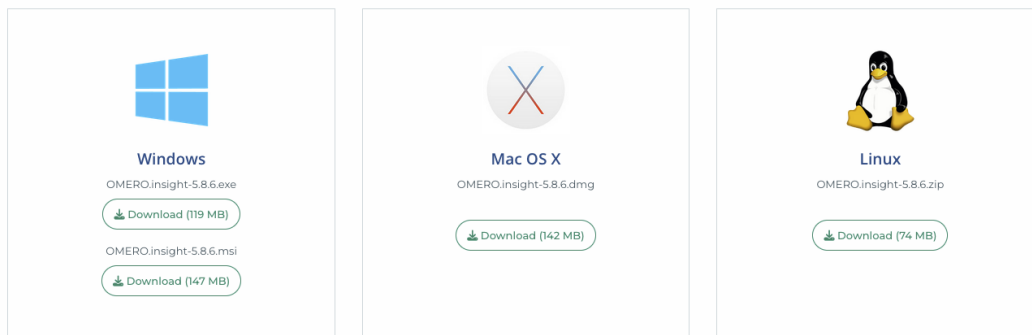
OMERO.insight desktop client installation instructions:.

Download the OMERO.insight client corresponding to your operating system at:
<https://www.openmicroscopy.org/omero/downloads>



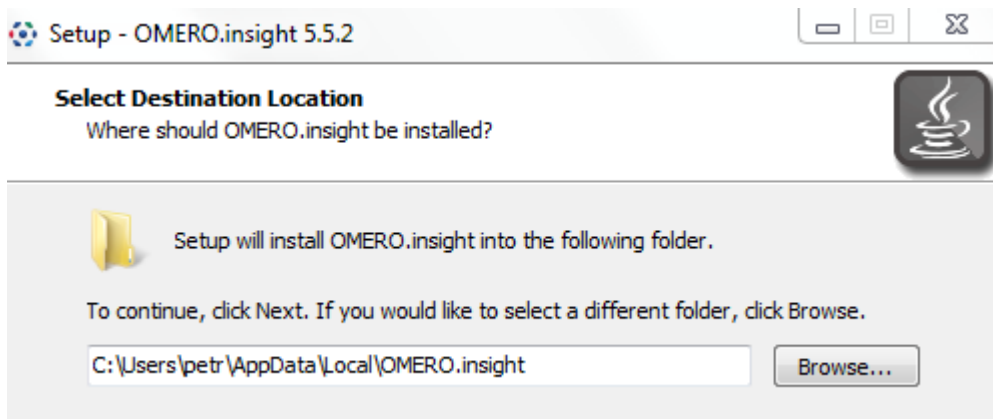
A standard OMERO user just needs to download the client package with a compatible version for their institutional server (e.g. 5.5.8 clients will connect to 5.6.0 servers but not to 5.4.0 servers). Full instructions for installation are on the [guides](#).

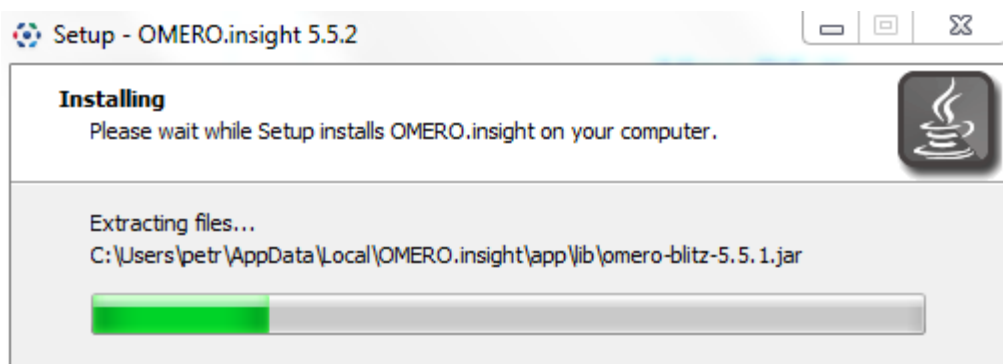
If you do not have an institutional server, you can take advantage of the OMERO platform hosted by the OME team in Dundee and sign up for a free demo account at <https://demo.openmicroscopy.org/signup/>. You will receive an email with your username and password allowing you to connect to the server in Scotland.



Windows

- From version 5.5.0, OMERO.insight comes with two installers: `.msi` and `.exe`.
- Click onto the downloaded `.exe` or `.msi` file.
- This will run the installer. The `.exe` file installs by default in the userspace apps folder. This can be changed during the installation process.

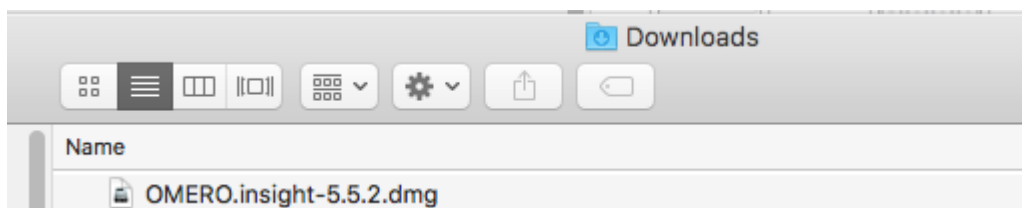




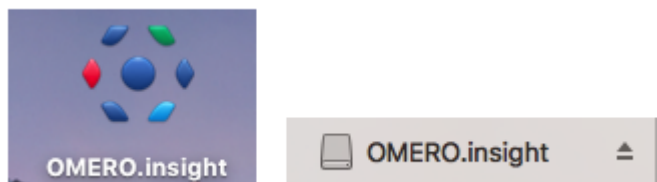
- The .msi installer will deploy the application in the Program Files folder of Windows. *OMERO.insight* is then available to all the users of the target machine. *OMERO.insight* installed in the userspace is only available to the user who did the installation. A Desktop icon and a new *OMERO.insight* Start menu item will be created.

Mac

- From version 5.5.0, OMERO.insight can be installed using an Apple DMG (.dmg) file.
- Mac OS X: Click onto the downloaded .dmg installer to start the installation.



- This will mount the DMG to your Mac. The DMG mounts in two places: on your Desktop and in the Finder sidebar under your hard drive.

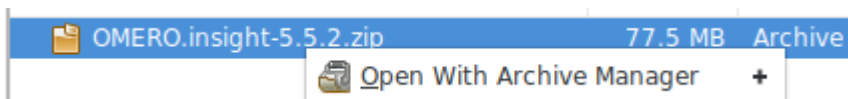


- Clicking either one of these opens the DMG file. When you open a DMG file, you will usually see two things:
 - the app itself.
 - a link to your applications folder.
- Depending on your settings, the Applications folder icon might not appear. In such case, drop the app icon into the Applications folder.

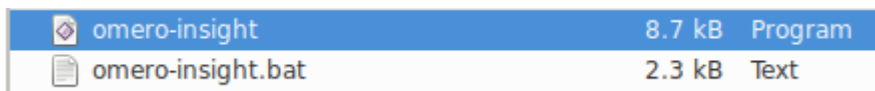


Linux

- Unzip the downloaded .zip file.

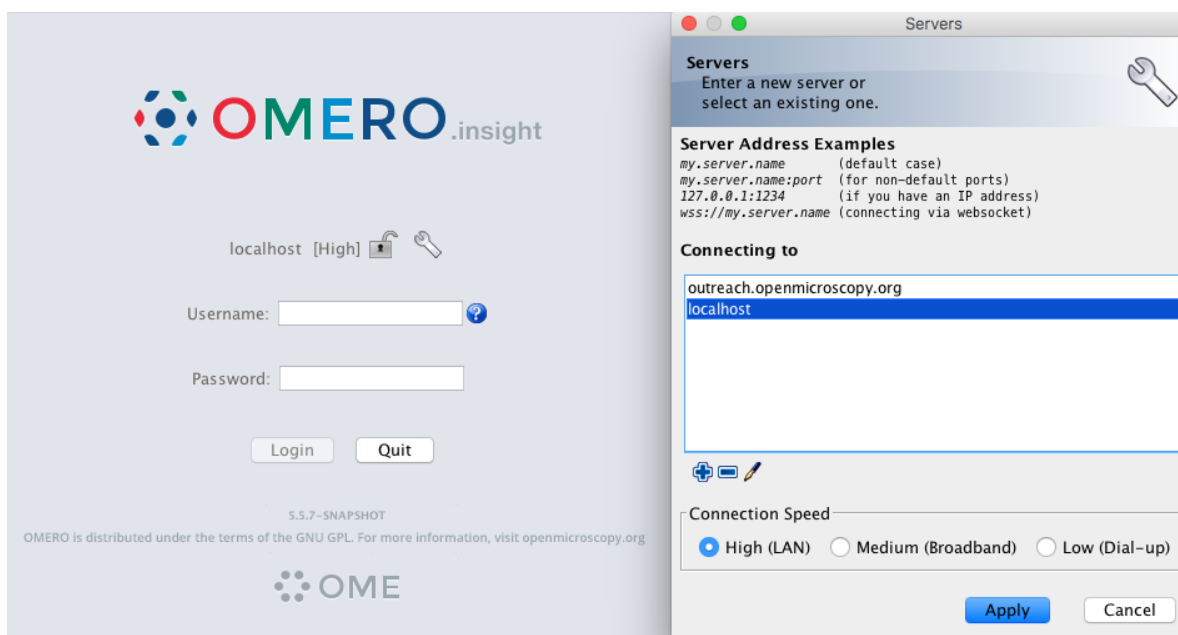


- Click on the omero-insight file to start the application.



Step-by-step

1. Open OMERO.insight and in the login dialog, click on the wrench icon .
2. This will open a list of servers to which you can connect. By default, only “localhost” is listed. Click on the plus icon to add a new line to the list and type into the line the server address.



Note: If your server is running on a default port (4064), as is usually the case, then you can simply just type in the server name such as `my.server.name`. For servers running on a non-default port, e.g. 54064, type the address of the server as `my.server.name:54064` into the above dialog. Alternatively, you can also type in the IP address of your server, or connect using websockets. The websockets connection has typically the form `wss://my.server.name/omero-ws` and can be used if your OMERO.server has been [configured to use websockets](#). The websockets can help in case the 4064 port access is blocked.

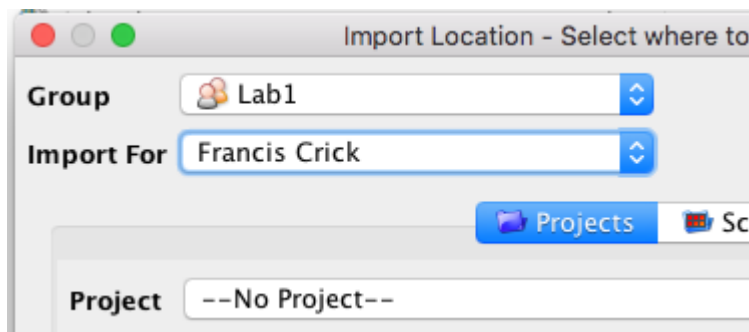
3. When done, click *Apply*.
4. Log in using the username and password provided.
5. In OMERO.insight, click on the Importer Icon  in the toolbar.

6. Browse your local hierarchy in the left-hand pane of the importer, select single images or whole folders and add these to the Queue by clicking on the arrow  icon.
7. In the Import Location window, select the target Project and Dataset (existing or create a new one) to import to.
8. Note: If no Dataset is selected or created, a new Dataset will be automatically created and named after the folder containing the images to be imported.
9. Optional: Go to the *Options* tab
 - Click on **Add tag to images**  : to bring the Tag selection dialog.
 - Select the tag(s) on the left-hand side or create a new one.
 - Click  to move the tag(s) to the right-hand side.
 - Click Save.
10. Click on the Import button in the bottom-right corner of the Importer window. You should see two progress bars for every image imported, Upload and Processing.
11. Note: The import of the next image in the queue starts immediately after the Upload of the previous one is finished. The Processing phase of the import is done on the OMERO.server only, and can be finished while the next image(s) is/are being uploaded to the server.
12. Once imports are finished, go back to the OMERO.insight main window and click the Refresh button  above the right-hand pane. This will display the imported images inside the Dataset and/or Project you specified previously in the Import Location window.
13. (demo only step) Now, the demonstrator will log out from OMERO.insight and log in again, this time as some other user and will show the import process again, this time importing a different set of images. After this import, the two sets of images (belonging to two different users) will be shown in the webclient.

Import for another user

In this example, we show how to import data for another user. A facility manager importer1 with restricted admin privileges imports the data for user-1. The facility manager has been given the ability to import for others.

The steps are the same as before for the normal import, but as importer1 has the permission to import for another user there are two drop down menus for selecting the user and group to import for:



- Select Group: 'Lab1'
- Select User: 'Francis Crick'

- Continue the import workflow as usual.

A restriction of OMERO.insight is that the user `importer1` needs to be a member of the groups he wants to import for. This restriction does not hold when importing the the Command Line Interface (CLI) (link to CLI import g.doc).

Import data using the Command Line Interface (CLI)

Description

This chapter will show how to import data for another user, using Command Line Interface (CLI).

The import can be done by any user as long as they import the data for themselves.

In case of the import for others, the user importing the data needs to have some admin (or restricted-admin) privileges. More information about restricted privileges can be found at [restricted-admins](#).

In the example workflow below, a user with restricted administrator privileges is used with login name `importer1`. This could be in real life e.g. a facility manager.

We will show:

- How to import data using the CLI for myself and for others into a remote OMERO.server
- How to import data using the CLI “in-place”, which means not copying the imported data into OMERO. Instead, OMERO will point to the original location of “in-place” imported files, thus preventing data duplication.
- How to deal with imports of large amounts of data in CLI, using the `–bulk` option and helper `csv` and `yml` files which define what is to be imported and how.

Resources

- Documentation:
 - <https://docs.openmicroscopy.org/latest/omero/users/cli/installation.html>
 - <https://docs.openmicroscopy.org/omero/latest/users/cli/index.html>
 - <https://docs.openmicroscopy.org/omero/latest/users/cli/import-target.html>
 - <https://docs.openmicroscopy.org/omero/latest/sysadmins/in-place-import.html>
 - <https://docs.openmicroscopy.org/omero/latest/users/cli/import-bulk.html>
- Data: example images from
 - <https://downloads.openmicroscopy.org/images/DV/will/FRAP/>
- Bash script for performing in-place imports:
 - https://github.com/ome/training-scripts/blob/master/maintenance/scripts/in_place_import_as.sh
- Example files for bulk import
 - `bulk.yml`

Setup

CLI Importer installation

Client libraries from the OMERO.server have to be installed on the client to import images using CLI. Please read the [installation instructions](#).

Note: When importing for another user using the CLI, the `importer1` does not have to be a member of the target group.

Step-by-step

1. If you did not do so already, open a terminal window on your local machine and activate the conda environment where your `omero-py` is installed (see the Setup section above):

```
$ conda activate myenv
```

2. Set the `OMERODIR` variable to point to the downloaded and unzipped OMERO server dir (see the Setup section above):

```
$ export OMERODIR=/path/to/OMERO.server-x.x.x-ice36-bxx
```

3. Login to the OMERO.server you wish to import to. This can be a remote OMERO.server. In the below example we log in as the user `importer1`, who will import for themselves:

```
$ omero -s your-server-address -u importer1 login
```

4. Go to the directory where there are some images you wish to import:

```
$ cd /path/to/images/directory/
```

5. Create an OMERO Dataset to import to:

```
$ DID=$(omero obj new Dataset name=import_for_myself)
```

6. Import the images as `importer1` into the newly created Dataset. You can import a single image as in the example below, or a whole directory of images:

```
$ omero import -d $DID <your-image-name>
```

7. Log in to OMERO.web as `importer1` and verify the imported image in the newly created dataset `import_for_myself`.

8. Import an image for another user, for example `user-1`. For that, your `importer1` user logs in as `user-1`:

```
$ omero --sudo importer1 -u user-1 login
```

9. Create a Dataset `import_for_user_one` as `user-1`:

```
$ DID=$(omero obj new Dataset name=import_for_user_one)
```

10. Import the data in the newly created Dataset:

```
$ omero import -d $DID <path-to-image-or-directory-with-images>
```

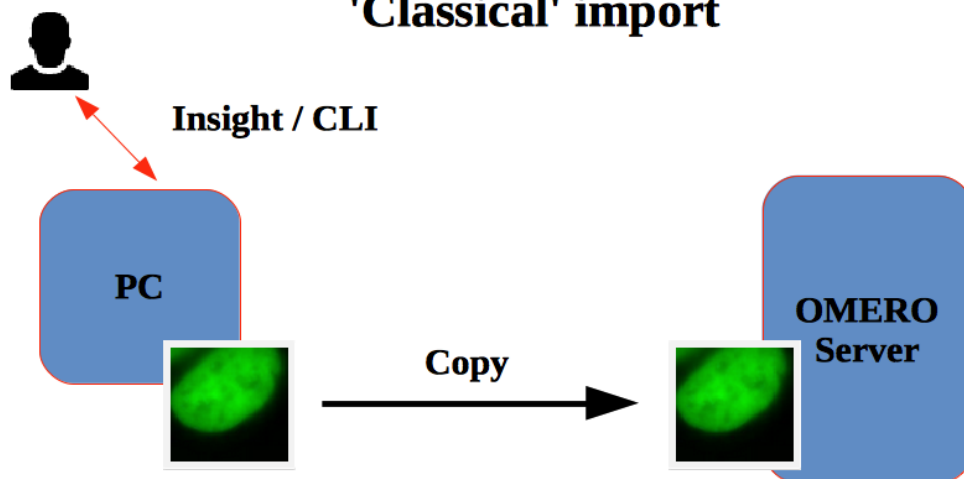
11. Check that the image(s) are successfully imported and that the image(s) and the containing Dataset both belong to `user-1` (not `importer1`).

In-place Import using the CLI

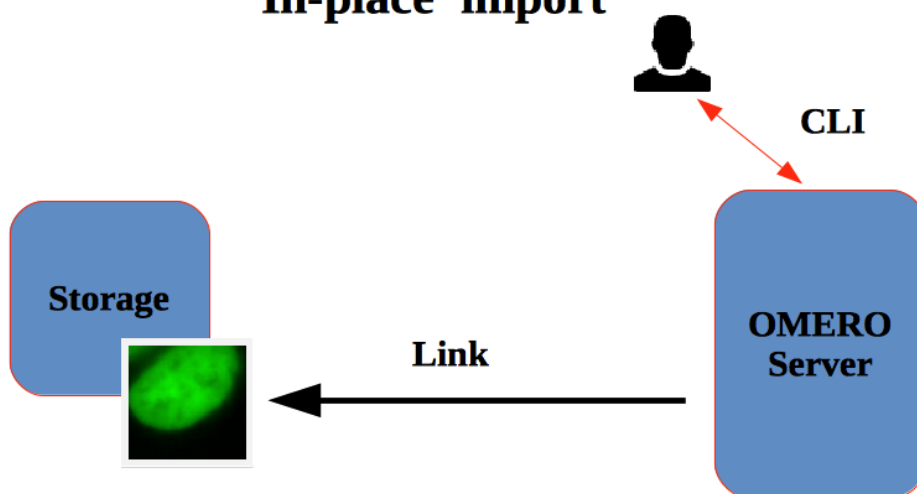
Instead of being copied into OMERO's managed repository, the image files stay at their original place and are just linked into the repository.

It is only available for the CLI importer, using the argument `--transfer=ln_s`.

'Classical' import



'In-place' import



Advantages:

- All in-place import scenarios provide non-copying benefit. Data that is too large to exist in multiple places, or which is accessed too frequently in its original form to be renamed, remains where it was originally acquired.

Limitations:

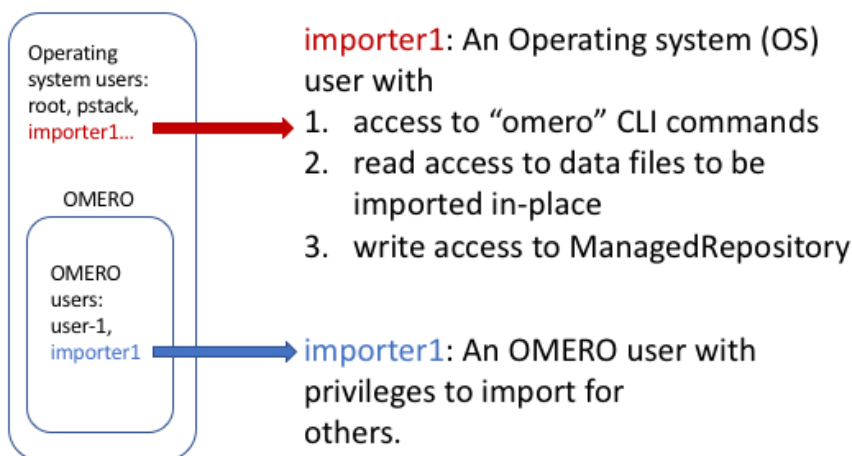
- Only available on the OMERO server system itself.
- Do not edit or move the files after an in-place import. OMERO may no longer be able to access them if you do.

Important:

A user performing an in-place import MUST have:

- a regular OMERO account
- an OS-account with ability to run omero commands on server machine
- read access to the location of the data
- write access to the ManagedRepository or one of its subdirectories. Please check the [ManagedRepository](#) documentation.

Unix machine with installed OMERO.server

**Step-by-step:**

1. Connect to the machine on which the OMERO.server is running as OS user `importer1` using `ssh`.
2. The aim is to import an image from `/OMERO/in-place-import/FRAP`:

```
$ ls /OMERO/in-place-import/FRAP
```

3. Activate the virtual environment where `omero-py` is installed or add it to `PATH`. In the example below, the path to the OMERO.server is `/opt/omero/server`:

```
$ export PATH=/opt/omero/server/venv3/bin:$PATH
```

4. Point `OMERODIR` to the location where the OMERO server is installed e.g.:

```
$ export OMERODIR=/opt/omero/server/OMERO.server
```

- Import now data for another user, this time a large image where the advantage of not copying the image file onto the server is most visible. The `importer1` user logs in as `user-1`:

```
$ omero --sudo importer1 -u user-1 login
```

- Create a Dataset `import_for_user_one`:

```
$ DID=$(omero obj new Dataset name=import_for_user_one)
```

- ‘In place’ import a large SVS file into the `import_for_user_one` dataset:

```
$ omero import -d $DID --transfer=ln_s /OMERO/in-place-import/svs/77917.svs
```

- Check that the image is successfully imported.
- Click on the paths icon  to show the difference between the normal and in-place (`ln_s`) imported images. Validate that In-place import is indicated Imported with --transfer=ln_s.
- Note: The script `in_place_import_as.sh` shows how to perform the in-place import steps described above in one single command.

Bulk Import using the CLI

In this example, we show how to combine several import strategies using a configuration file. This is a strategy heavily used to import data to IDR.

We import two folders named *siRNA-HeLa* and *condensation*.

Note: Connecting over SSH is necessary only if you intend to import in-place. If you do not wish to perform the bulk import in “in-place” manner, you can connect to the server remotely using locally installed OMERO.cli and adjust the `bulk.yml` file by commenting out the `transfer...` line, then follow the steps as described below.

- Open a terminal and connect to the server (for example as `importer1`) over SSH. Alternatively, use your local terminal with installed OMERO.cli if not importing “in-place”.
- Description of the files used to set up the import (see `bulk.yml`, `import-paths.csv` and [import-bulk.html#bulk-imports](#) for further details).
 - `import-paths.csv`: (.csv, comma-separated values) this file has at least two columns. In this case the columns are separated by commas. The first column is the name of the target Dataset and the second one is the path to the folder to import. We will import two folders (the `import-paths.csv` has two rows).

Example csv (note the comma between the “HeLa” and “/OMERO...”):

```
*Dataset:name:Experiment1-HeLa,/OMERO/in-place-import/siRNAi-HeLa*
*Dataset:name:Experiment2-condensation,/OMERO/in-place-import/condensation*
```

- `bulk.yml`: this file defines the various import options: transfer option, checksum algorithm, format of the .csv file, etc. Note that setting the `dry_run` option to true allows to first run an import in `dry_run` mode and copy the output to an external file. This is useful when running an import in parallel. Comment out the `transfer` "ln_s" if not importing “in-place”.

Example `bulk.yml`:

```
*continue: "true"*

*transfer: "ln_s"*

*# exclude: "clientpath"*

*checksum_algorithm: "File-Size-64"*

*logprefix: "logs"*

*output: "yaml"*

*path: "import-paths.csv"*

*columns:*
  - *target*
  - *path*
```

3. Activate the virtual environment where `omero-py` is installed or add it to `PATH`. In the example below, the path to the OMERO.server is `/opt/omero/server`:

```
$ export PATH=/opt/omero/server/venv3/bin:$PATH
```

4. Point `OMERODIR` to the location where the OMERO server is installed e.g.:

```
$ export OMERODIR=/opt/omero/server/OMERO.server
```

5. Find the place where the `bulk.yml` file is located, for example `/OMERO/in-place-import`:

```
$ cd /OMERO/in-place-import
```

6. The importer1 (Facility Manager with ability to import for others) OMERO user logs in as `user-1`:

```
$ omero --sudo importer1 -u user-1 login
```

7. Import the data using the `--bulk` command:

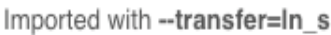
```
$ omero import --bulk bulk.yml
```

8. Go to the webclient during the import process to show the newly created dataset. The new datasets in OMERO are named `Experiment1-HeLa` and `Experiment2-condensation`. This was specified in the first column of the `import-paths.csv` file.

9. Select an image.

10. In the right-hand panel, select the General tab to validate:

- Click on  to show the import details.

- Validate that In-place import is indicated  in case you imported “in-place”.

Advantages:

- Large amount of data imported using one import command.
- Heterogeneous data for multiple users can be imported using bulk import in combination with bash scripting, e.g. `in_place_import_as.sh`
- Reproducible import.

Limitations:

- Preparation of the .csv or .tsv file.

Combine the CLI imports with post-import steps

The following example shows how to do the import on CLI and follow-up operations like rendering and metadata import in one step.

In many cases, the rendering and metadata import is best done separately, as the visual checking of the imported images might be crucial for further rendering and metadata import, see *Change image rendering settings and channel names using the Command Line Interface (CLI)* and *Import metadata using the Command Line Interface (CLI)* for details on this.

Further, the images you are importing might need a range of different rendering settings, not just one set of settings for all of them. Also for this case, the step-by-step approach, first importing the images, only then deciding on the rendering strategy and preparing the `renderingdef.yml` files, is preferable.

Nevertheless, there are cases which do not need visual checks and use a single rendering for all images, for which a streamlined sequence of commands is offered below which will perform all three steps (import, rendering and metadata import) in one single session on the CLI.

Resources

Additionally to the *Resources mentioned in the import-cli section* and in the *Setup* you will also need the rendering and metadata plugins as mentioned in *Change image rendering settings and channel names using the Command Line Interface (CLI)* and *Import metadata using the Command Line Interface (CLI)*, and possibly the following files:

- <https://github.com/ome/training-scripts/blob/master/maintenance/preparation/renderingdef.yml>
- `simple-annotation.csv`
- `simple-annotation-bulkmap-config.yml`

Step-by-step

1. If you did not do so already, open a terminal window on your local machine and activate the conda environment where your `omero-py` is installed (see the *Setup*):

```
$ conda activate myenv
```

2. Set the `OMERODIR` variable to point to the downloaded and unzipped OMERO server dir (see the *Setup*):

```
$ export OMERODIR=/path/to/OMERO.server-x.x.x-ice36-bxx
```

3. Prepare an `renderingdef.yml` file, by either creating a new one or downloading <https://raw.githubusercontent.com/ome/training-scripts/master/maintenance/preparation/renderingdef.yml>.

4. Prepare an *annotation.csv* file, by creating a new file or downloading the provided example file. In the example below, we use the file *simple-annotation.csv*. The Dataset names in this CSV file must match the Dataset names in OMERO as created in the *DID* variable definition line in the command below. The Image names in the CSV file must match the file names in your imported folder.
5. Prepare a *bulkmap-config.yml* file. In the example below, we use the file *simple-annotation-bulkmap-config.yml*.
6. Log in to the OMERO.server you wish to import to. This can be a remote server if you do not wish to import *in-place*.
7. Import, render and annotate in a single command sequence below:

```
$ PID=$(omero obj new Project name='Project_import_concatenate')
$ DID=$(omero obj new Dataset name='siRNAi-HeLa')
$ omero obj new ProjectDatasetLink parent=$PID child=$DID
$ omero import -d $DID /path/to/data/folder/or/image/siRNAi-HeLa --file import.out
$ omero render set $DID renderingdef.yml
$ omero metadata populate --report --batch 1000 --file /path/to/downloaded/simple-
↪annotation.csv $PID
$ omero metadata populate --context bulkmap --cfg simple-annotation-bulkmap-config.
↪yml --batch 100 $PID
```

8. Log in to OMERO.web and check that the images are imported, have the expected rendering settings and also the annotations in the form of Key-Value pairs on each imported image.

For more information about CLI import options, go to [import.html](#).

Import data using OMERO.dropbox

Description

OMERO.dropbox allows to import files into OMERO automatically, by means of offline import from a watched directory. Typically, each user has a folder into which they “drop” their images as these are being acquired. The folder can be directly on the machine where OMERO is installed, or, more commonly, OMERO can watch directories mounted on the machine where OMERO is installed.

Alternatively, the files to import can be copied into the folders watched by OMERO.dropbox by cron jobs <https://en.wikipedia.org/wiki/Cron> or systems developed by the community users such as <https://github.com/imcf/auto-tx>, which automatically harvest the files from acquisition computers and transfer them onto shared network drives.

We will show

- How to set up OMERO.dropbox on your OMERO.server.
- How to set up the directories/folders into which the users or the automatic systems described above will copy the files to import.
- How to import several files for two different users using OMERO.dropbox. The image files are manually copied into the prepared folders watched by OMERO.dropbox.

Resources

- Documentation:
 - <https://docs.openmicroscopy.org/latest/omero/sysadmins/dropbox.html>
 - <https://docs.openmicroscopy.org/omero/latest/users/cli/import-target.html>
- Data: example images from <https://downloads.openmicroscopy.org/images/DV/alexia/cajal-bodies/>

Setup

This example setup will help you to understand OMERO.dropbox functionality. Later, you can choose the setup you like for your OMERO.server studying detailed instructions at <https://docs.openmicroscopy.org/latest/omero/sysadmins/dropbox.html#advanced-use>

Below are the installation instructions.

- First connect to the OMERO.server over SSH using your administrator account.
- In the OMERO.server directory set the following lines to enable and configure Dropbox:
 - `$ bin/omero config set omero.fs.watchDir "/home/DropBox"`
 - `$ bin/omero config set omero.fs.importArgs "-T \\"regex:^(?<Container1>.*?)\\""`

Note: The last line makes sure the created Dataset into which the images will be imported will have the same name as the deepest folder in the DropBox folder for that particular user. Every image in any Experiment-1 directory will end up in the same Dataset for that user, however the superstructure of folders have been.

To set up different import options, go to <https://docs.openmicroscopy.org/omero/latest/users/cli/import-target.html>.

For example, it is possible to create a Dataset with a fixed name for every user or to create a Dataset whose name is picked from the first subfolder under images folder.

- On the OMERO.server machine, create a folder called DropBox under /home.
- `$ cd /home`
- `$ mkdir DropBox`
- Create under the /home/DropBox directory two subdirectories for two users e.g. user-1 and user-2. The names of these directories should match the login names of those users in OMERO.

Note: the omero system user must be able to read from those directories. Also, the users or the system which will drop files into those directories must have write permissions there.

- `$ cd /home/DropBox`
- `$ mkdir user-1`
- `$ mkdir user-2`

Step-by-step

1. Open a browser window.
2. Enter the URL provided.
3. Login as user-1 or any user with the right to see user-1's data.
4. Leave the browser window open.
5. On your computer, open a new terminal window.
6. Connect over SSH to the machine where the OMERO.server is running.
7. Open the logfile DropBox.log under var/log/ e.g.
 - `$ tail -f /path/to/OMERO.server/var/log/DropBox.log`
8. Open a new terminal window.
9. Connect over SSH to the machine where the OMERO.server is running.
10. In any folder on that machine, create a directory Experiment-1 into which you copy an image. Drop the Experiment-1 directory into the user-1 folder you created during the setup above:
 - `$ cd /path/to/other/dir`
 - `$ mkdir Experiment-1`
 - `$ scp 090829_5_HeLa_siCTL_coilin_ATUB01_05_R3D_D3D.dv ./Experiment-1`
 - `$ cd /home/DropBox`
 - `$ scp -r /path/to/other/dir/Experiment-1 ./user-1` #copy the whole directory "Experiment-1" into the directory watched by OMERO
11. The OMERO.dropbox will intentionally wait for 60 seconds between registration of the new drop into the folder and the actual import, in anticipation of further file drops of files into the DropBox folder.
12. After you have copied the directory into the user-1 folder, you should see in the DropBox.log lines like as follows.

```
2019-08-12 17:09:23,644 INFO [ fsclient.fsDropBoxMonitorClient]
(Thread-3 ) New entry
/home/DropBox/user-1/Experiment-1/090829_5_HeLa_siCTL_coilin_ATUB01_05_R3D_D3D.dv
contains 1 file(s). Files=1 Timers=1

2019-08-12 17:10:23,644 INFO [ fsclient.fsDropBoxMonitorClient]
(Thread-17 ) Removed key
/home/DropBox/user-1/Experiment-1/090829_5_HeLa_siCTL_coilin_ATUB01_05_R3D_D3D.dv

2019-08-12 17:10:23,666 INFO [ fsclient.fsDropBoxMonitorClient]
(Thread-17 ) Importing
/home/DropBox/user-1/Experiment-1/090829_5_HeLa_siCTL_coilin_ATUB01_05_R3D_D3D.dv
(session=2155e6d0-445a-496b-a6bc-8afeb93ac58d)

2019-08-12 17:10:23,955 INFO [ omero.util.Resources] (Thread-18 )
Starting

2019-08-12 17:10:23,961 WARNI [ stderr] (Thread-17 ) Joined session
for user-2@localhost:4064. Idle timeout: 10 min. Current group: Lab1
```

(continues on next page)

(continued from previous page)

```

2019-08-12 17:10:31,989 INFO [ fsclient.fsDropBoxMonitorClient]
(Thread-17 ) Import of
/home/DropBox/user-1/Experiment-1/090829_5_HeLa_siCTL_coilin_ATUB01_05_R3D_D3D.dv
completed (session=2155e6d0-445a-496b-a6bc-8afeb93ac58d)

2019-08-12 17:10:32,001 INFO [ omero.util.Resources] (Thread-18 )
Halted

```

1. Go back to OMERO.web. Refresh the tree.
2. Observe that a new Dataset was created, with the name Experiment-1. The image is imported into that Dataset.
3. The image is always imported into the default group of the user.
4. Repeat the workflow for user-2. First, go to your browser, logout and login again as user-2.
5. Connect over SSH to the machine where the OMERO.server is running if required.
6. Create again a folder, Experiment-2.
7. Copy an image into it
8. Copy the whole Experiment-2 folder under /home/DropBox/user-2.
9. Go back to the browser, refresh and verify that you can see a Dataset Experiment-2 under user-2's data with the image inside.
10. Note: Even if user-2's folder in the previous workflow uses the same name for their dataset as user-1 (Experiment-1), the data of user-2 would not be imported into user-1's Dataset. Instead, a new Dataset Experiment-1 would be created under user-2's data, belonging to user-2, into which the image would be imported.

Change image rendering settings and channel names using the Command Line Interface (CLI)

Description

This chapter will show how to change rendering settings on images using the Command Line Interface (CLI).

This action is typically done after a successful import of images.

We will show:

- How to change rendering settings of large amount of images on the CLI in a repeatable manner.

Resources

- Documentation:
 - <https://docs.openmicroscopy.org/latest/omero/users/cli/installation.html>
 - <https://docs.openmicroscopy.org/omero/latest/users/cli/index.html>
- Data: example images from
 - IDR data `idr0021`.
- Rendering plugin for OMERO
 - <https://pypi.org/project/omero-cli-render/>

- Rendering yml files defining the various rendering parameters, such as the channel color, channel name and minimum and maximum values.
 - <https://github.com/ome/training-scripts/blob/master/maintenance/preparation/renderingdef.yml>
 - <https://github.com/ome/training-scripts/blob/master/maintenance/preparation/renderingdef2.yml>
- Rendering mapping file has two columns, in the left-hand one there are the images to be rendered and in the right-hand side column points to the appropriate renderingdef.yml for that image.
 - <https://github.com/ome/training-scripts/blob/master/maintenance/preparation/renderingMapping.tsv>
- Shellscript for batch modification of rendering settings of images
 - https://github.com/ome/training-scripts/blob/master/maintenance/scripts/apply_rnd_settings_as.sh

Setup

Rendering plugin installation

- Go to the environment where you installed your OMERO.cli as specified under - <https://docs.openmicroscopy.org/latest/omero/users/cli/installation.html>.
- Activate the virtual environment.
- Run:

```
$ pip install omero-cli-render
```

Step-by-step

1. On your local machine, open a terminal
2. Activate the virtual environment where omero-py is installed or add it to PATH e.g.:

```
$ export PATH=/opt/omero/server/venv3/bin:$PATH
```

3. The variable \$ID below is the ID of the selected Dataset. To change the rendering of images in one Dataset, run:

```
$ omero render set Dataset:$ID local_path/to/renderingdef.yml
```

4. Verify the change in the browser.
5. To change the rendering of images in two Datasets, run:

```
$ omero render set Dataset:$ID1 Dataset:$ID2 renderingdef2.yml
```

6. Modify the rendering settings in batch using a shellscript such as [apply_rnd_settings_as.sh](#) which uses HQL to find the Images IDs in OMERO and deliver them to the omero-cli-render plugin. The script reads the Datasets in which the Images are located in OMERO are listed from a renderingMapping.tsv file, such as [renderingMapping.tsv](#). Go to the folder where the script is located. Then run the bash script with the default parameters:

```
$ sh apply_rnd_settings.sh
```

The script could be run by a facility manager on behalf of other users.

Import metadata using the Command Line Interface (CLI)

Description

This chapter will show how to import metadata starting from a local CSV file and ending with OMERO.tables on images or Key-Value pairs on images using the Command Line Interface (CLI). For a more user-friendly way of uploading metadata using graphical interface see the *Import metadata using the Populate Metadata script in OMERO.web* chapter.

This action is typically done after a successful import of images.

We will show:

- How to import metadata from local CSV file in a bulk manner and turn them into OMERO.tables on images using CLI
- How to turn the OMERO.tables on images into Key-Value pairs on images in bulk manner using CLI
- How to import metadata from local CSV file and use a server-side script in OMERO to turn these into OMERO.tables on images
- How to construct a simple file to turn the metadata stored in OMERO.tables into Key-Value pairs on images using CLI

Resources

- Documentation:
 - [CLI installation](#)
 - [CLI](#)
- Data: example images from
 - IDR data [idr0021](#)
 - [siRNAi-HeLa dataset](#)
- Metadata plugin for OMERO
 - <https://pypi.org/project/omero-metadata/>
- Bulkmap config yaml files defining the various Key-Value pairs parameters, such as the groups and other parameters.
 - [idr0021-experimentA-bulkmap-config.yml](#)
 - [simple-annotation-bulkmap-config.yml](#)
- Annotation CSV files define the content of OMERO.tables for each image.
 - [idr0021-experimentA-annotation.csv](#)
 - [simple-annotation.csv](#)
 - [four-images.csv](#)

Setup

Metadata plugin installation

- Go to the environment where you installed your OMERO.cli as specified under - [CLI installation](#).
- Activate the virtual environment.
- Run:

```
$ pip install omero-metadata
```

Step-by-step

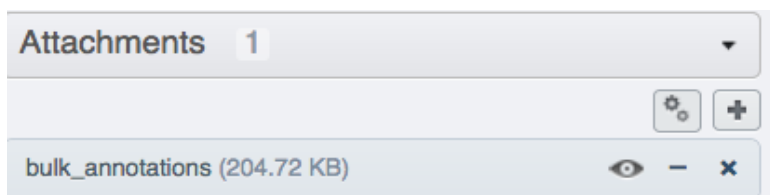
1. On your local machine, open a terminal
2. If you did not do so already, activate the virtual environment where omero-py is installed or add it to PATH e.g.:

```
$ export PATH=/opt/omero/server/venv3/bin:$PATH
```

3. Download the CSV from [idr0021-experimentA-annotation.csv](#) if you have access to the [idr0021](#) data in your OMERO.server. Alternatively, download [simple-annotation.csv](#), which will allow you to work with the [siRNAi-HeLa](#) dataset.
4. The variable \$ID below is the ID of the Project, in this example case it is the Project containing the idr0021 study. If you are working with the siRNAi-HeLa data, replace in the following example the “Project” with a “Dataset” and the [idr0021-experimentA-annotation.csv](#) with [simple-annotation.csv](#). To add annotations from a local CSV file to the images in the said Project or Dataset in the form of OMERO.tables, run:

```
$ omero metadata populate --report --batch 1000 --file local/path/to/idr0021-  
↪experimentA-annotation.csv Project:$ID
```

5. Open your browser and login to the OMERO.web. Navigate to the Project or Dataset you just worked with, expand the “Attachments” harmonica in the right-hand pane and verify that a new attachment is on that Project named `bulk_annotations`.



6. You can inspect its content by clicking on the “eye” icon  inside the attachment.
7. Select an image inside the Project/Dataset and expand the “Tables” harmonica in the right-hand pane. These tables contain the appropriate line from the `bulk_annotations` attachment you just created for that particular image.

Tables	
INFO	
Dataset Name:	CDK5RAP2-C
Image Name:	Centrin_PCNT_Cep215_20110506_Fri-1620_0_SIR_PRJ.dv
Characteristics [Organism]:	Homo sapiens
Characteristics [Cell Line]:	HeLa
Characteristics [Cell Cycle Phase]:	interphase
Experimental Condition [Antibody Target]:	CDK5RAP2-C

8. Go back to your terminal. Download the `idr0021-experimentA-bulkmap-config.yml` file . Alternatively, in case you are working with the siRNAi-HeLa Dataset, download `simple-annotation-bulkmap-config.yml`.
9. If you are working with the IDR data, open the downloaded `idr0021-experimentA-bulkmap-config.yml` file in a text editor and delete the `Advanced options...` section. Save the file and run:

```
$ omero metadata populate --context bulkmap --cfg local/path/to/idr0021-experimentA-
↪bulkmap-config.yml --batch 100 Project:$ID
```

10. If you work with the siRNAi-HeLa data, open the downloaded `simple-annotation-bulkmap-config.yml` and study the comments in the file itself, which will give you hints about how to manipulate the file to fit your particular needs with respect to the resulting Key-Value pairs layout. Make your changes (no need to change anything if you do not want), save the file locally and run:

```
$ omero metadata populate --context bulkmap --cfg local/path/to/simple-annotation-
↪bulkmap-config.yml --batch 100 Dataset:$ID
```

11. Go to your browser and in OMERO.web, select the images in the Project or Dataset you targeted and verify that they have now new Key-Value pairs displayed in the right-hand pane.

The screenshot shows a web interface titled "Key-Value Pairs" with a dropdown menu showing "7". Below the title, there are three sections of metadata:

- openmicroscopy.org/mapr/cell_line**
Added by: Maurice Wilkins
Cell Line: HeLa
- Gene**
Added by: Maurice Wilkins
Gene Identifier: ENST00000394818.8 (with a small square icon to its right)
Gene Symbol: INCENP
Gene Symbol Synonyms: FLJ31633
- openmicroscopy.org/mapr/organism**
Added by: Maurice Wilkins
Organism: Homo sapiens

Import metadata using the Populate Metadata script in Omero.web

Description

This chapter will show how to import metadata starting from a local CSV file and ending with Omero.tables on Images or wells using the [server-side script](#) Populate Metadata in Omero.web. See also the workflow described in [Import metadata using the Command Line Interface \(CLI\)](#) which will give you more possibilities, such as Key-Value Pairs creation, but which is using Command Line Interface (CLI). The workflow described here is using only graphical user interface elements.

This action is typically done after a successful import of Images.

We will show:

- **How to import metadata from local CSV file in a bulk manner and turn them into Omero.tables on**
 - *Images contained in Projects/Datasets*
 - *Wells contained in Screens/Plates*
- How to *create or adjust a local CSV file containing metadata* for creation of Omero.tables containing numbers and text.

Resources

- Annotation CSV files define the content of Omero.tables for each image or each well.
 - `four-images.csv`
 - `simple-screen.csv`
- `omero-metadata` plugin (necessary for having full set of features in Populate Metadata script).
 - <https://pypi.org/project/omero-metadata/>

Setup

omero-metadata plugin installation

Note: For the best experience, the omero-metadata plugin should be installed on your OMERO.server. The Populate Metadata script tries to reuse the code of the omero-metadata plugin. If the plugin is not found on the server, the Populate Metadata script falls back on a deprecated code, with limited set of features. The omero-metadata plugin installation is typically done by the administrator of the OMERO.server.

- In your OMERO.server environment, go to the environment where you installed your OMERO.cli as specified under - [CLI installation](#).
- Activate the virtual environment.
- Run:

```
$ pip install omero-metadata
```

Populate Metadata script

No explicit installation necessary, shipped with the OMERO.server.

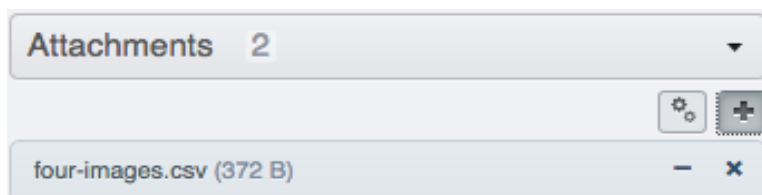
Step-by-step

Project/Dataset/Image

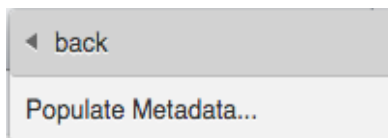
1. Log in to OMERO.web, create a new Dataset and copy into it four Images, preferably Images which have no OMERO.tables on them. Note the name of the Images you are copying in.



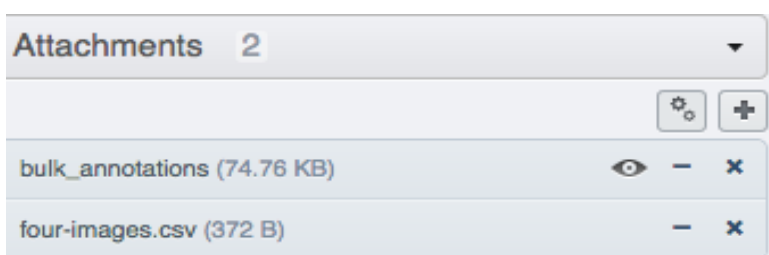
2. Download `four-images.csv`. Open the CSV file in Excel and edit the name of the Images in the first column to match the names of the Images you copied into your Dataset in the previous step. Also, edit the name of the Dataset in the second column to match the name of your Dataset in OMERO.web. Save the file locally as CSV.
3. (Optional) In your OMERO.web, upload the CSV file you just saved and attach it onto the Dataset you created previously. Alternatively, you can skip this step, and point the Populate Metadata script to the local CSV, as explained below.



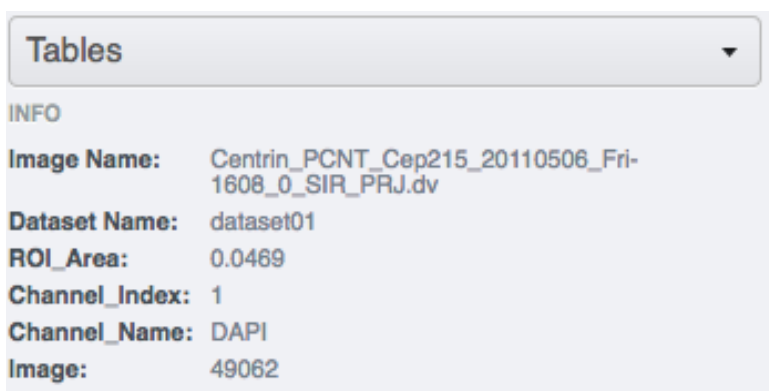
4. Select the Dataset you created. Find the script icon  above the central pane, expand it and find the **Import scripts** section. In there, select the **Populate metadata** script which will launch the script dialog.



5. If you did not attach the CSV to the Dataset, you can now click on the **Browse** button and select the CSV from your local machine.
6. Click **OK** to run the script, and wait for it to show as complete in the **Activities** panel in the top-right corner above the central pane.
7. Click again onto the Dataset in the left-hand pane to refresh and observe that there is a new Attachment in the right hand pane under “Attachments” harmonica, named **bulk_annotations**.

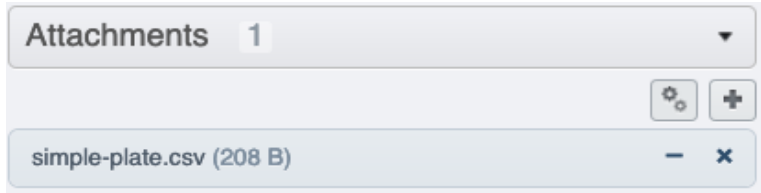


8. Click on single Images inside the Dataset and observe that in the “Tables” harmonica in the right-hand pane there are new values coming originally from your edited CSV.

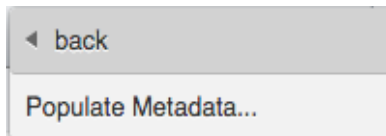


Screen/Plate/Well

1. Find a Plate inside a Screen in OMERO.web which has no OMERO.tables on its Wells.
2. Download `simple-screen.csv`. Open the CSV file in Excel and edit the name of the wells in the first column to match the names of the wells in your Plate from the previous step. Also, edit the name of the Plate in the second column to match the name of your Plate in OMERO.web. Save the file locally as CSV.
3. (Optional) In your OMERO.web, upload the CSV file you just saved and attach it onto the Screen containing the Plate you created previously. Alternatively, you can skip this step, and point the **Populate Metadata** script to the local CSV, as explained below.



4. Select the Screen you identified above. Find the script icon  above the central pane, expand it and find the **Import scripts** section. In there, select the **Populate metadata** script which will launch the script dialog.



5. If you did not attach the CSV to the Screen, you can now click on the **Browse** button and select the CSV from your local machine.
6. Click **OK** to run the script, and wait for it to show as complete in the **Activities** panel in the top-right corner above the central pane.
7. Click again onto the Screen in the left-hand pane to refresh and observe that there is a new Attachment in the right hand pane under **Attachments** harmonica, named **bulk_annotations**.
8. Click on single Wells inside the Plate under the Screen and observe that in the **Tables** harmonica in the right-hand pane there are new values coming originally from your edited CSV.

Create a metadata CSV

1. Download the `four-images.csv` (for Images in Projects/Datasets) or `simple-screen.csv` (for Wells in Screens/Plates) as templates to create your own CSV.
2. Open the downloaded CSV file in Microsoft EXcel, but do not use **Import** command in Excel, instead, either double-click on the file or use the **Open** command in Excel. Populate the values in the CSV using Microsoft Excel with your own numbers or text, possibly expanding the number of rows or columns as appropriate.
3. Replace the **# header** ... column types inside the templates with your own column types according to the content of your CSV: Follow the Note below for guidelines. Save the file as CSV in Microsoft Excel.

Note: The **# header** row is optional. If **# header** is not used, all column types are treated as String (i.e. text, not numbers) in OMERO. The header abbreviations have following meaning:

d: DoubleColumn, for floating point numbers

l: LongColumn, for integer numbers

s: StringColumn, for text

b: BoolColumn, for true/false

plate, well, image, dataset, roi: to specify objects

If the target is a Project, the CSV file needs to specify Dataset Name and Image Name. If the target is a Dataset instead of a Project, the Dataset Name column is not needed.

If the target is a Screen, the CSV file needs to specify Plate name and Well. If a **# header** is specified, column types must be **well** and **plate**. If the target is a Plate, the CSV file **must not** specify a Plate column, but it must specify the Well column.

Column names should not contain spaces if you want to be able to query by these columns.

Contribute

Changes to the documentation or the materials should be made directly in the [omero-guide-upload repository](#).

3.1.2 General Introduction

This section covers the general concepts of OMERO. Those concepts are demonstrated using the Web client. This section introduces the data management functionalities of OMERO, shows how to annotate data and demonstrates how to search for data and metadata. Also, the management of groups and users is presented, both in the Web client as well as on the command line.

Contents:

Data management and cooperation

In this document, we introduce the basic concepts of data management, such as browsing, navigating to others' data, and changing the display of the images in OMERO. The example here uses OMERO.web, but majority of the features described here are also present in OMERO.insight.

Further, we show how to use the Command Line Interface (CLI) for data management, introducing mainly the features which are not present in the OMERO.web.

Description

We will show:

- How to browse data in OMERO.web, navigating to yours and other users' Images.
- How to use the basic layout of OMERO.web for Images organized in Projects and Datasets.
- How to use OMERO.web for viewing of High-Content Screening (HCS) data.
- How to use the Preview panel.
- How to adjust the rendering settings of your and other users' images from the Preview panel.
- How to organize Images in Projects and Datasets.
- How to *move the data between groups if you are data owner*.
- How to move the data between groups if you are an administrator working on behalf of others.
- How to *change the ownership of objects*.
- How to use *Command Line for duplicating objects* such as Images, Datasets or Projects.

Setup

OMERO.server has been installed and provisioned using an [Ansible playbook](#).

Resources

- All data have been pre-imported. For more details, look at [data.md](#).
- For the HCS data screenshot, the IDR data <https://idr.openmicroscopy.org/webclient/?show=run-5403> were used.
- To import Images and metadata, see the [maintenance scripts](#) for more details.
- For import of Images, we use [in_place_import_as.sh](#).
- The cooperation in OMERO is described in [Groups and permissions system](#).
- See also the documentation for [Moving objects between groups](#) and [Changing ownership of objects](#) using the Command Line Interface.

Step-by-Step

Data layout and ownership, usernames (when running a workshop)

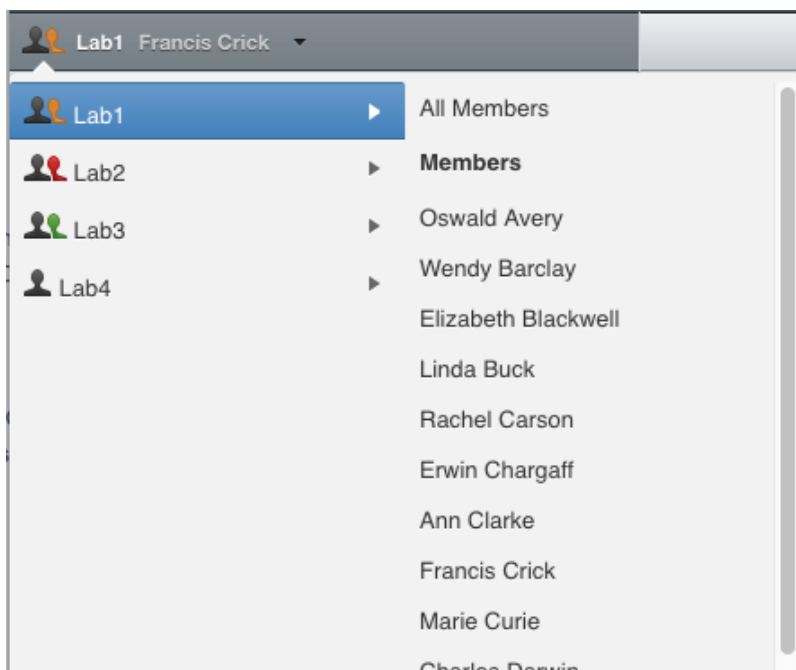
All Images have been pre-imported into the OMERO.server. For training purposes, we prepared 50 users on the OMERO.server. Each of these 50 users has their own set of Images. These sets consist of Images of the same name, size, shape, form and quality for each user. Thus, the data of each user appears the same but, in actual fact, are different and totally independent sets. Thus, small discrepancies and differences between the users are completely possible. Further, if one user deletes their own data in OMERO (please do not delete anything on our server), this will not have any bearing on the other 49 sets of Images belonging to the 49 other users.

The login names of the 50 users are “user-x” where x goes from 1 to 50. We have given to each of the 50 users in OMERO a unique first name and surname which we picked from a list of 50 famous scientists e.g. Ada Lovelace or Francis Crick. This is the name (“your” name) which you will see in the top-right corner of the OMERO webclient after you log in with your loginname.

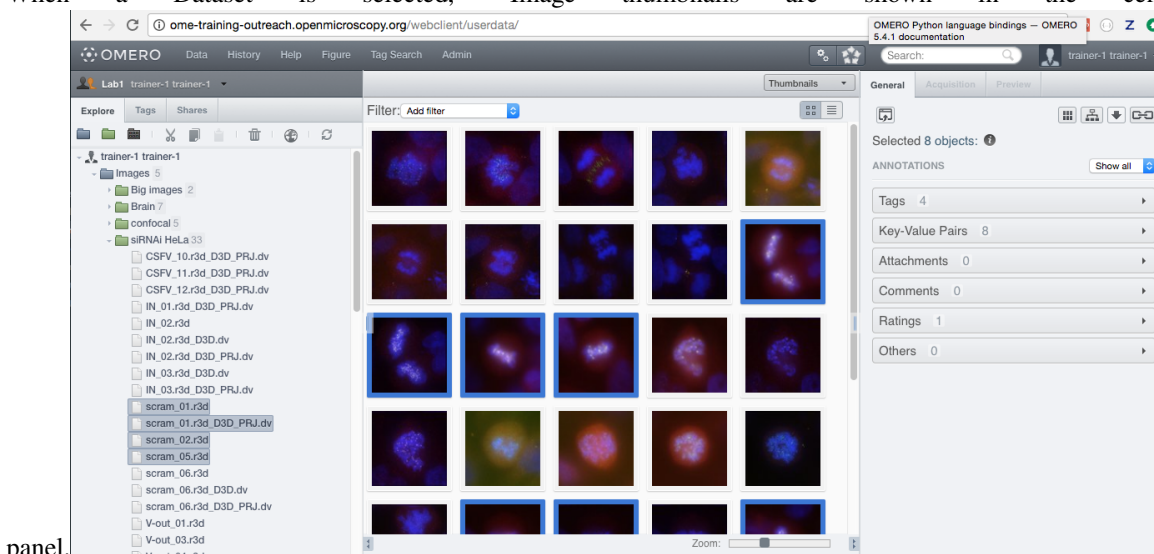
In the OMERO webclient the default view shows only your own Images.

Browsing and rendering

1. In your web browser, go to the server address provided.
2. Log in using the username and password provided.
3. OMERO offers various levels of permissions for groups and users. Depending on the permissions level of each group, a given user might be able to view, annotate or edit data belonging to other users. Permissions are managed by users with admin privileges. To highlight some of the collaborative aspects of OMERO, we will use a “Read-Annotate” group. This implies that users **can** view and annotate each other’s data but **cannot** delete other people’s data.
4. To see other people’s data, click on the (for example Lab1) group name in the top-left corner of the webclient. Note that in case you are in a differently named group, the name in the top-left corner will accordingly reflect this. Then use the menu to select the name of the group you wish to browse and then the user inside that group whose data you wish to see.

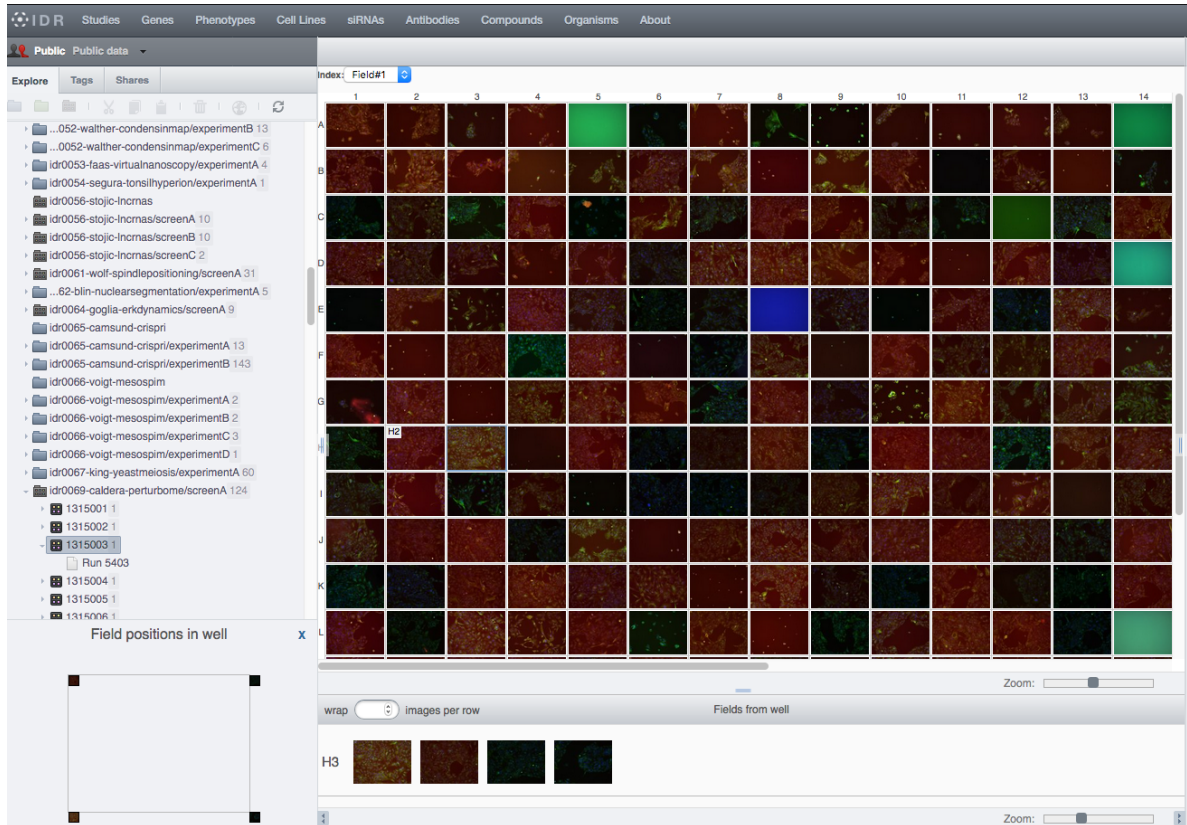


5. You can browse 'folders' in the left-hand pane: Image folders are called Datasets and they are within Projects.
6. When a Dataset is selected, Image thumbnails are shown in the centre



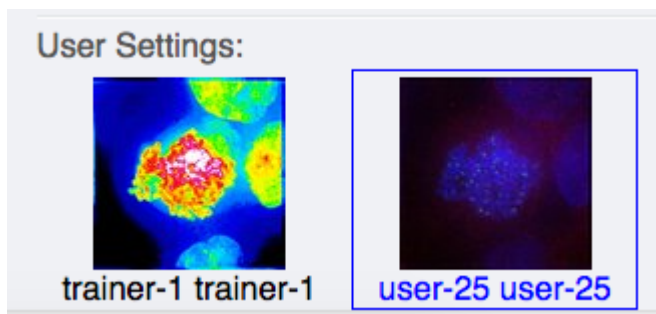
7. These represent imported Images. The original Images are stored on the server and the generated thumbnails allow us to browse them.
8. **Bio-Formats** is used to read the pixel-data and metadata from over 150 different Image formats, including multi-z timelapse Images with many channels, they are referenced as 5D Images. Large pathology and medical Images are also supported.
9. For HCS data, the layout of the OMERO.web is a bit different. The HCS data are usually organized in following manner:
 - Images are contained in Wells.
 - Wells are contained in Plates.

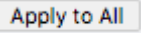
- Plates are organized in Screens.
- A Plate may or may not contain several Runs.
- The screenshot below shows the typical layout of a Plate in OMERO.web, where the Wells are organized in rows and columns. The Plate contains one Run. One Well is selected in the central pane and it contains four Images whose thumbnails are displayed below the central pane. The bottom-left corner shows the positions of the Images (called Fields in this context) inside that Well.



10. Select an Image. In the right-hand pane, metadata read by Bio-Formats and stored in a relational database is displayed:
 - Core metadata in the General tab.
 - Additional metadata in the Acquisition tab.
 - All the metadata read by Bio-Formats can be downloaded at any time.
11. In the Preview tab in the right-hand panel, you can also view the Image.
12. For multi-plane Images, sliders allow you to move through Z or Time dimensions.
13. Viewing Images **does not** download the whole Image to the client. Only the viewed Image plane is rendered from the original Image file on the server and sent back to the OMERO.web client.
14. You can adjust the rendering settings for each channel e.g. turn on/off the channels, adjust color settings, look-up tables, etc.
15. The rendering settings can be saved to the server. This **never** changes the original Image data and can be reverted at any time.
16. The rendering settings can also be copied and pasted between Images. To modify the rendering settings in batch, click on the Save to All button to apply the same settings to, for example, all Images in a given Dataset.

17. You can use the settings which other users saved on your Images and apply them for your own Image. These settings are highlighted as thumbnails in the lower part of the Preview pane.

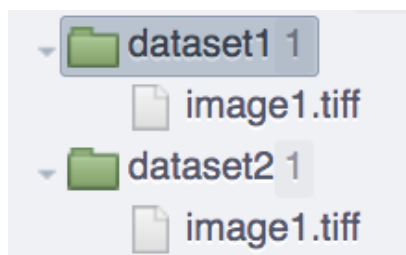


18. Your own settings are highlighted in blue.
19. You can revert to the original settings for an Image or Dataset. For example, using the context menu for a Dataset in the tree, select **Rendering Settings > Set Imported and Save**.
20. Stay in **General** tab of the right-hand pane and adjust the channel names:
- Select any image inside that Dataset and click on the pencil  icon in the right-hand pane next to Channels.
 - Input “DAPI” instead of channel “457” and “GFP”, “Aurora-B” and “CY-6” for the other channels.
 - Click the **Apply to All** button  and confirm by clicking **Continue**. This will change the channel names of all the images in that Dataset.

Manage Images in Datasets/Projects

You can organize the data in the left-hand side tree by creating new Projects and Datasets. You can link the Images to the new or existing Datasets and Datasets to new or existing Projects. For HCS data, you can create new Screens and link Plates to these Screens.

1. Use the Project  and Dataset  icons above the left-hand side tree or the right-click contextual menu to create new Datasets or Projects.
2. Drag and drop Images between Datasets and Datasets between Projects. For HCS data, drag and drop Plates between Screens.
3. **Copy Images using the right-click context menu:**
 - Select the Images to be copied, then right-click and click **Edit > Copy Link**.
 - Select the Dataset you want to copy the Images to, right-click and click on **Edit > Paste Link**.



Warning: The Copy Link feature will only create new links between an Image and a Dataset, so that one Image becomes linked to multiple Datasets. **This does not create a new independent copy of the Image.** The only way to create a fully independent copy in OMERO is to use the *Duplicate feature*.

If you delete one of the Datasets, any Images within it that are linked to other Datasets will be retained. Nevertheless, **if you directly select and delete an Image that has been copied from another Dataset it will be deleted and lost from both Datasets.** There is a clear warning in the OMERO.web when you try to delete such a doubly linked Image, see screenshot below.

Delete



Warning: One or more Images are linked to multiple Datasets. This will DELETE them from ALL of those Datasets. If you only wish to remove Images from one Dataset, use the "Cut Link" action.

Note: Organizing data of other users as an administrator, restricted administrator or group owner is made easier in OMERO.web in the following manner.

If you are an administrator or administrator with restricted privileges working in a group you are not a member of, except for private groups where this workflow is not possible, all containers (Projects, Datasets, Screens) created in OMERO.web in such a group will belong to the user whose portfolio you are working with.

Changed in version 5.8.0: Also the links between the containers and their content will belong to that user. In case there are different owners of the container and of the linked content, then the created link will belong to the owner of the linked content.

This helps to retain the possibility for this user to manipulate their containers even though you created them.

Nevertheless if the workflow is executed by a group owner (i.e. not an administrator), the new links will belong to the group owner and the user will not be able to unlink the objects later. This is a current limitation.

Move data between groups

In OMERO, Users are organized in Groups. The Groups allow a level of viewing and cooperation between the members of the group which can be adjusted by changing the permissions level on that group. A User can be a member and have their data in one or more Groups. Thus it is sometimes necessary to move the data between groups. This action can be done by the owners of the data themselves or by an administrator or an administrator with restricted privileges.

Note that caution has to be taken in case the data are linked to other users' containers (Datasets, Projects). If you move only the contents of those containers (Datasets or Images) and not the containers themselves (Projects or Datasets), the links between such containers and the Images or Datasets which are moved will be deleted.

Further, if any objects are moved, the links to any annotations such as Tags or attached File annotations linked to these objects will be deleted in case these annotations belong to others or in case these annotations belong to you but are also linked to some other objects in the original group which are not being moved.

If you want to retain a copy of your data in its current position, then you should *Duplicate* it first. The *Duplication* also bypasses the unlinked annotations problem highlighted in the previous paragraph. Because the *Duplicate feature*

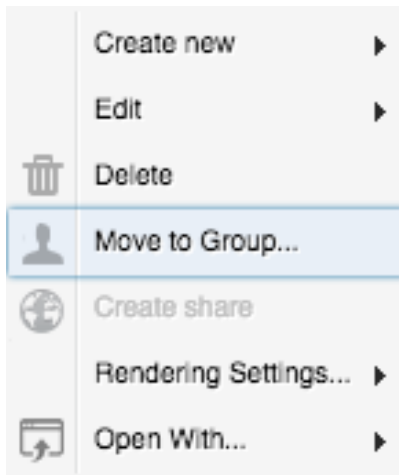
creates new objects, which are not dependent on the originals, the relationships between these duplicates are then preserved even during a subsequent move to another group.

Note that except for using OMERO.web described below, it might be worth in some situations to consider moving data between groups using the Command Line Interface see [CLI Moving Objects between Groups](#).

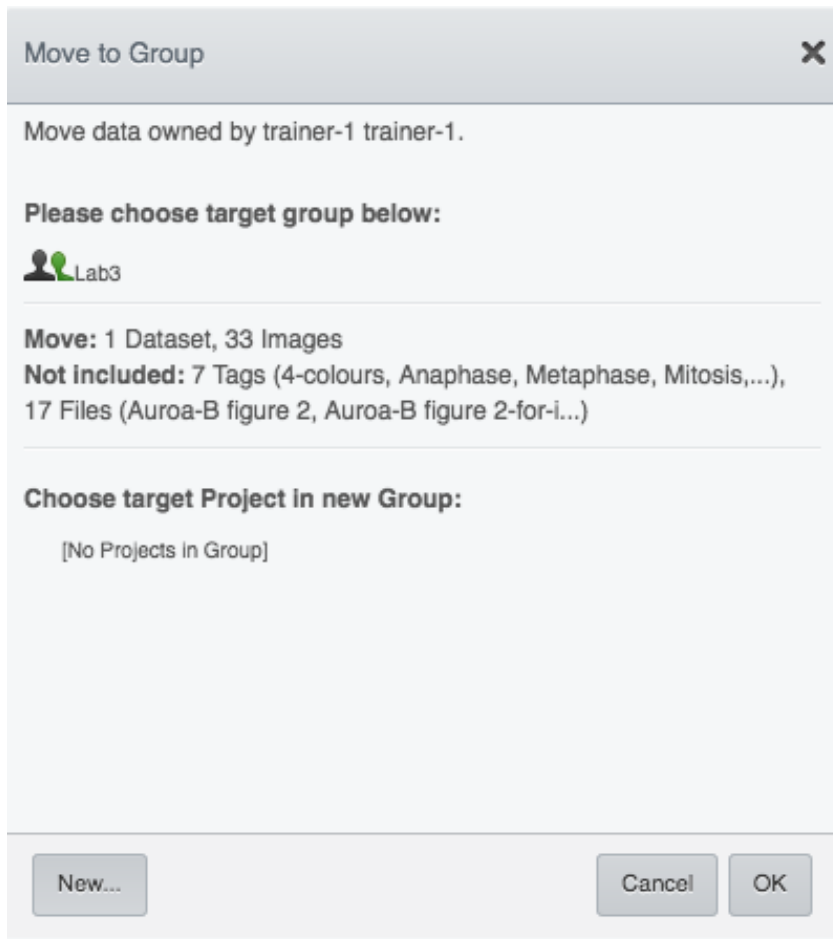
Move data between groups: owners of data

If you are an owner of the data, you can move the data between the groups you are a member of.

1. In OMERO.web, select the data to be moved in the left-hand side tree.
2. Right-click and select **Move to Group...**



3. Select the group you want to move the data to.
4. A message **Checking which linked objects will be moved** will appear and a spinner to the left of it. Wait until the spinner vanishes and a list of objects to be moved and a list of objects which are not included in the move appears.



5. Check both lists. Please read the note above about which objects are typically not included and reconsider the Move action. The not included objects will not be linked to the Moved objects anymore if you go ahead with the move, the linkage will be lost.
6. In case you are happy with the Move action to go ahead, select a target Dataset or Project or create a new one and click OK.

Move data between groups: administrators

The administrators can move the data to any group, not only to the group where the owner of the data is a member. Note though that it is not desirable to create a situation where the data belong to someone who is not a member of the group where the data reside.

Typically an administrator works on behalf of other users in a group where the administrator is not a member. For these cases, some features of OMERO.web help to facilitate the moving of data for others (note that these features are not yet available in the Command Line Interface).

1. Navigate to the data of a user in a group that you are not a member of.
2. Select the data in the left-hand tree.
3. Right-click and select Move to Group...
4. Follow further the steps described in the section *Move data between groups: owners of data*, taking note of the Not included objects.

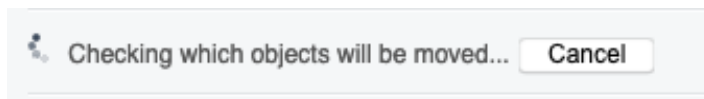
5. When creating new Datasets or Projects during the move, note that these containers will belong to the owner of the data, not yourself. Also the links between the new containers and the moved data will belong to the owner of the data. This should help to facilitate a smooth workflow, retaining data-handling possibilities such as reorganizing the data, renaming the containers you created for them etc. for the owner of the data.

Change ownership of data

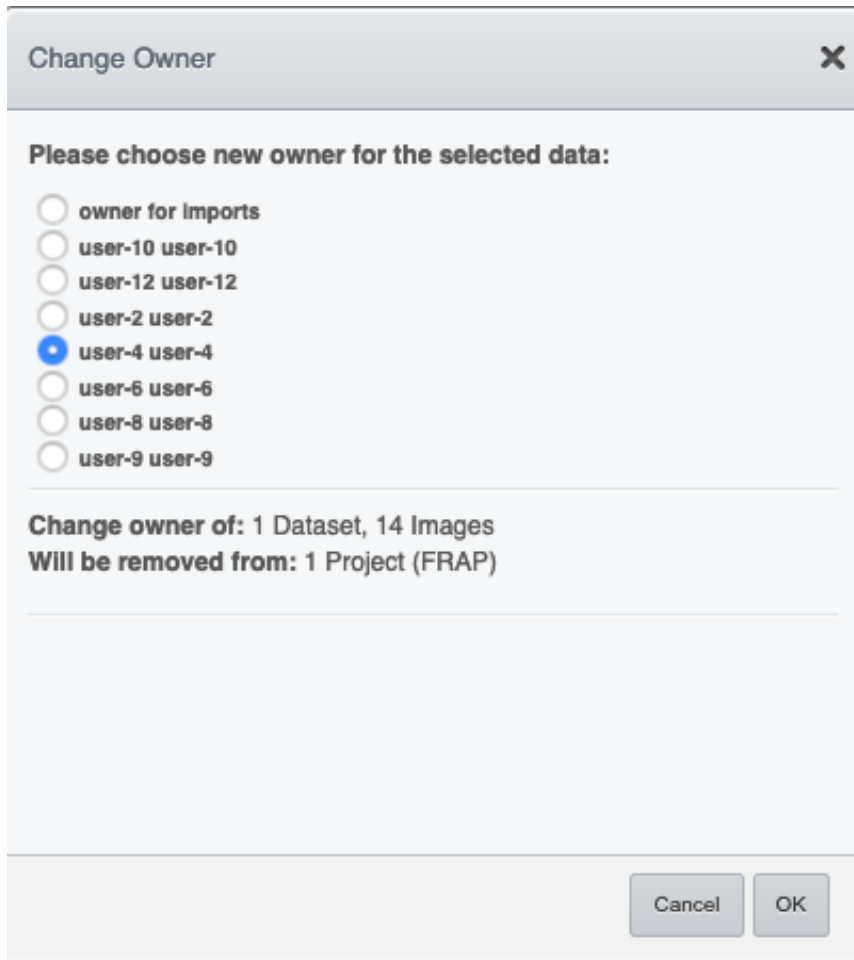
Every object in OMERO has a single, particular user as an owner. The ownership of objects can be changed but only if you are an Administrator, Administrator with restricted privileges or a Group owner.

There is also a possibility of [changing ownership of objects](#) using the Command Line Interface. The Command Line Interface implementation contains some features not present in OMERO.web.

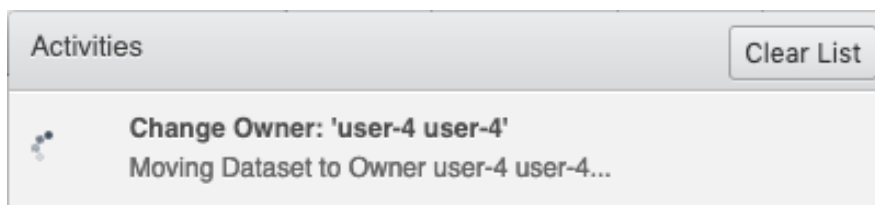
1. Select a Project, Dataset or Images in the left-hand side tree of OMERO.web.
2. Right-click and in the context menu, select **Change owner**.
3. In the new dialog, select the new owner of the data.
4. Wait until the **Checking which objects will be moved** is done and the spinner vanishes. This might take time depending on the number of objects you are attempting to change owner of. For example, a large number of ROIs on the images can be a cause of longer waiting times.



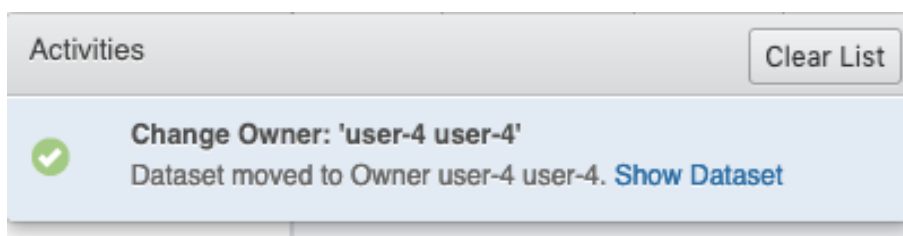
5. Check the list of objects which are supposed to be transferred to the new owner. Also consider the possible loss of linkage between objects listed in the **Will be removed from** line.



6. Click OK.
7. Observe the Activities item above the central pane of OMERO.web and wait until the change of ownership is finished.



8. The Activities item will stop showing a spinner and a link to the data you have just changed the owner of will appear. This link will help you to find the data you just changed the ownership of. This data will be shown as a part of the tree of the new owner in the OMERO.web interface, and are now removed from the old owner tree.



9. Click on the link Show . . . in the Activities and inspect the data you just transferred the ownership of.

Command Line: Duplicating objects

You can duplicate objects in OMERO. The functionality is available only on the CLI for now. The duplication creates a full copy of the objects as if they were created independently. When desired, the newly duplicated objects can additionally be linked to other objects. When the duplicate is later deleted, it will not have an influence on the original, which stays preserved. Similarly, when the original is later deleted, the duplicate stays preserved as well.

In case of Image objects, which have image files linked, the duplication creates a new image file which is linked to the original image file by a hard link when possible. This means every duplication of an Image increases the number of hard links on the image file in OMERO.server's Managed Repository, but does not duplicate the image file itself, and thus does not increase the storage demands too much, except for rare cases where the linking is not possible. If creation of the hard link is not possible, the Image duplication will still proceed, creating a full new image file copy.

In case of File Attachment objects though, which also have files to them, each duplication will duplicate the linked file, thus doubling the storage space necessary for these File Attachment files.

Use this Duplicate feature to *replace the discontinued Shares feature*.

Resources

- Necessary software versions:
 - **OMERO.server 5.6.3 or later**
 - **omero-cli-duplicate plugin 0.4.0 or later**
- Documentation:
 - [README of the omero-cli-duplicate repository](#)
 - [Application Programming Interface documentation](#)
- Plugin for duplication on Command Line:
 - [omero-cli-duplicate on PyPI](#)

Setup

Duplicate plugin installation

- Go to the environment where you installed your OMERO.cli as specified under [Installation](#).
- Activate the virtual environment where omero-py is installed or add it to PATH e.g.:

```
$ export PATH=/opt/omero/server/venv3/bin:$PATH
```

- Run:

```
$ pip install omero-cli-duplicate
```

Step-by-Step

1. On your local machine, open a terminal
2. Activate the virtual environment as indicated in the Setup section above.
3. The variables \$ID1 and \$ID2 below are the IDs of the selected Datasets. To duplicate two Datasets with their Images and annotations on both the Images and the Datasets, run:

```
$ omero duplicate Dataset:$ID1,$ID2 --report
```

4. The duplicated Datasets will not be linked to any Project, even if the originals were linked to some Project.
5. Duplicate two Images with many ROIs on them. The ROIs duplication might take a long time. To exclude the duplication of the ROIs, run:

```
$ omero duplicate Image:$ID1,$ID2 --ignore Roi --report
```

6. Find the duplicated Images in the Orphaned Images and Drag and Drop them into a Dataset or create a new Dataset for them.
7. Duplicate two Projects of another user in read-annotate group type. This will duplicate also the Datasets linked to those Projects as well as Images linked to those Datasets. Here we specify some classes of objects that we do not want to duplicate (*reference-classes*), even though they are linked to the objects we are duplicating. Instead, the duplicated objects will be linked to these *reference-classes* of objects. We can also specify classes that we explicitly do want to duplicate using the *duplicate* argument. In this example, we specify all Annotations as *reference-classes*, but a subset of these (Comments and LongAnnotations such as Ratings) are *duplicate-classes*. This means that the duplicated Projects will be linked to the original annotations such as Tags, Key-Value pairs and FileAnnotations, but Comments and Ratings will be duplicated. This can be useful to ensure that a single Comment is not attached to multiple objects which might cause confusion when the Comment is edited. Also it prevents a loss of link between the Comment and an object in case that object gets moved into another group after the duplication. The duplicated Annotations and the two Projects with the linked Datasets and Images will belong to the current user (*user-1*), even if all the original Annotations may belong to other users in the read-annotate group. Run:

```
$ omero logout
$ omero login -u user-1 -g read-annotate-group
$ omero duplicate Project:$ID1,$ID2 --reference Annotation --duplicate_
↪ CommentAnnotation,LongAnnotation --report
```

8. Duplicate two Projects of another user in read-only group type. The group name in our example below is *read-only-group*. If the duplicator is not an administrator, administrator with restricted privileges or group owner, they cannot link the annotations of another user to their duplicate in a read-only group. They might duplicate all the annotations (this is the default behaviour) or exclude the duplication of all the annotations by excluding the Link duplication to the relevant objects as shown below. Run:

```
$ omero logout
$ omero login -u user-1 -g read-only-group
$ omero duplicate Project:$ID1,$ID2 --ignore IAnnotationLink,Roi --report
```

Note: You must log in to the group where the data are, either by virtue of this group being your default group or by using the -g flag as shown in the examples above, otherwise the *omero-cli-duplicate* plugin will not find the data. This is a current limitation.

If you intend to move the duplicate into a different group, it is recommended that you duplicate the annotations as well. If you link the annotations from other objects to your duplicates, the link might be deleted during the subsequent move

of that duplicate to another group.

Shares (discontinued feature)



Previously created Shares can still be viewed in the Shares tab above the left-hand side pane of Omero.web. Nevertheless, the Shares feature is discontinued. **It is not possible to use the World**



icon above the left-hand pane in Omero.web to create new Shares anymore. Shares are discontinued in Omero.web 5.9.0 and later. You need to use the Omero standard permissions to share Images, by *moving the data* into the appropriate group. If you also want to retain the Images in their current group, you can first duplicate them as described in *Duplicate feature*. Nevertheless, this *Duplicate* and *Move* workflow has following limitations:

- *Duplicate* is only available on the Command Line Interface (CLI)
- You need to already be in a non-private group with the user you wish to share with
- You will also be sharing the data with all the other members of that group

Note: Shares were never supported by Omero.iviewer. If Omero.iviewer or other viewer which does not support Shares is installed on your server and is set as a default viewer, then users with permissions to access the data in the Share cannot view images in any Full viewer.

Annotate Data and Filter using Annotations

There are several ways to add annotations to objects in Omero. Here, addition and filtering of annotations using Omero.web is described. You can add annotations using the Omero.web interface to any object(s) that you can select in the left-hand-side tree or central pane, this means Project, Dataset, Image, Screen, Plate and Well.

Description

We will show:

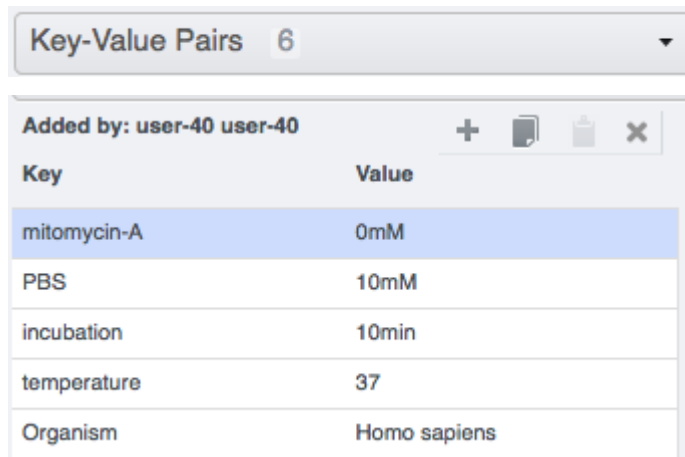
- How to annotate Images, Datasets and Projects with:
 - Tags
 - Key-Value Pairs
 - File Attachments
 - Ratings
- How to filter thumbnails in the central pane of Omero.web for:
 - Tags
 - Key-Value Pairs
 - Ratings

Setup

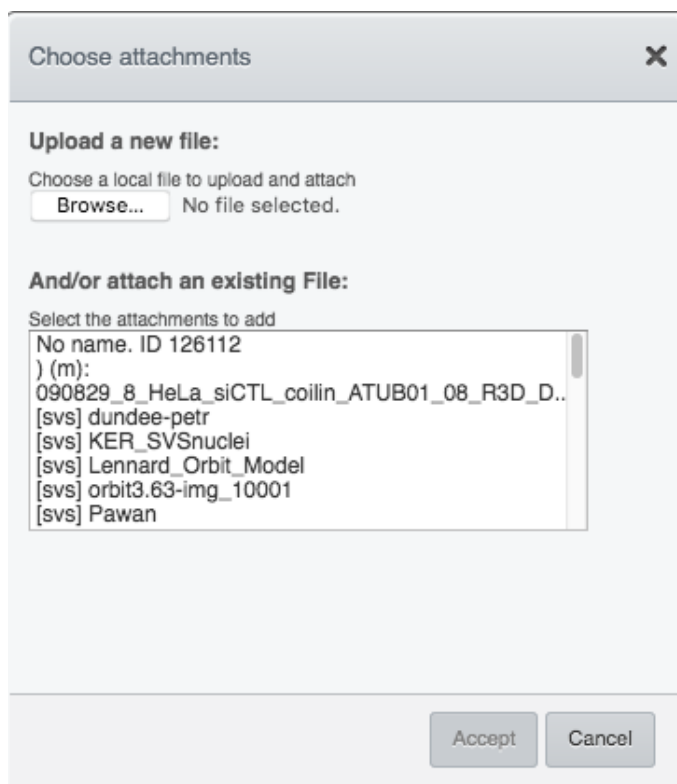
- The data used are from the [siRNAi-HeLa](#) folder.
- The tags and ratings were added manually, then propagated for all users on the OMERO.server using the script [copy_tags_ratings.py](#).
- The Key-Value Pairs were added to the images in the siRNAi-HeLa Dataset for all users using the script [key_value_pairs.py](#).

Step-by-step

1. Open a browser and enter the provided URL
2. Connect using the provided credentials
3. First, we will add Tags to indicate Metaphase stages of these cells.
 - Select one or more Images in the siRNAi-HeLa Dataset of cells which appear to be in metaphase.
 - Choose the Tag harmonica in the right-hand General tab and click [+] to launch the Tag dialog.
 - Choose the existing Metaphase tag from the list of Tags (to filter, type above the list).
 - Click > to move it to the right column, then click Save.
4. Let us now add Key-Value Pairs
 - Select an Image from the Dataset and in the right-hand pane in General tab, click the harmonica Key-Value Pairs.



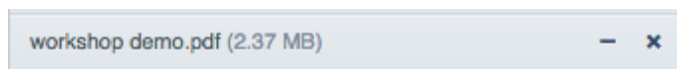
- The Key-Value Pairs allow you to add lab-book-like additional metadata for the Image. These Key-Value Pairs are also specifically searchable. See [Search for Data](#).
5. For adding of File Attachments:
 - Select one or more objects in the left-hand side tree, such as Dataset or Image.
 - Expand the Attachments harmonica in the right-hand pane.
 - Click the plus button.



- You can attach any type of file using this function. If you select a file from your local filesystem using the **Browse** button, the feature will upload that file to the OMERO.server and save it there. The content of .pdf, CSV and plain text files is also searchable in OMERO.

6. Remove a File Attachment.

- Find the File Attachment you have just added.
- Click on the minus sign to the right of it.

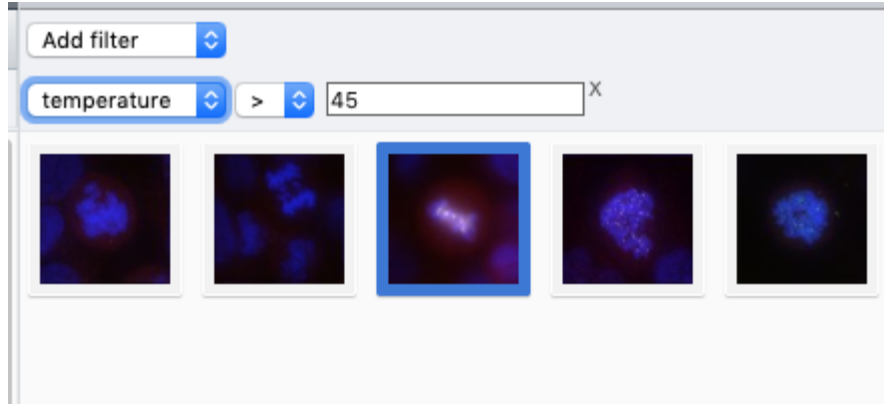


- The removal action just unlinks the File Attachment from the selected object(s). The File Attachment is not deleted from the server. If deletion is needed, click in the workflow above on the cross icon instead of the minus icon.

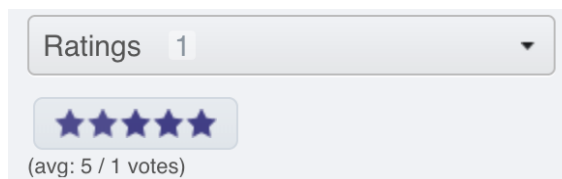
7. You can also add Comments and Rating to selected objects - follow analogous steps to the ones described above for Tags, Key-Value pairs and File Attachments.

8. Filter using annotations

- Images can also be filtered by Name, Tag, Key-Value pairs or Rating in the centre pane, using the **Add filter** chooser above the thumbnails.
- For example, choose **Tag** and then select **Metaphase** from the list of Tags to show the images we tagged earlier.
- Or choose to filter by Key-Values. You can then filter by a particular Key. If you select a Key where all the values are numbers, you can filter for those that are greater than, less than or equal to a threshold value.



- Review the filtered Images, choose a favourite Image and under the Ratings section in the right-hand pane, click on the 5th star to add a rating of 5



- Now we can remove the filtering by Tag and instead filter by Rating of 5 to show only our favourite images.

Search for Data

In this section, we present the server-wide search in OMERO. This is an additional option for data management, which complements filtering, mining and using annotations to organize your data, which are described elsewhere in [Annotate Data and Filter using Annotations](#).

Description

We will show:

- How to start a search in OMERO.web.
- How to search for objects annotated with a specific Key-Value Pair.
- How to use Advanced Search.
 - How to search for terms in specific fields e.g. “name”.
 - How to combine search terms using AND.
 - How to combine search terms using AND NOT.

Resources

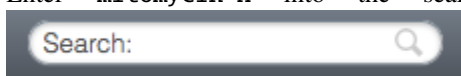
- Documentation:
 - [Search](#)

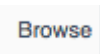
Setup

- The data used are from the [siRNAi-HeLa](#) folder.
- The Key-Value Pairs were added to the images in the siRNAi-HeLa Dataset for all users using the script [key_value_pairs.py](#).

Step-by-Step

1. Open a browser and enter the provided URL
2. Connect using the provided credentials
3. Enter `mitomycin-A` into the search box in the top right corner of the webclient



4. Press Enter.
5. The search results will show any objects e.g. Images or Datasets, which have anywhere the string `mitomycin-A`.
6. Several images should be found.
7. Refine the search now for only Key-Value Pairs which have the key `mitomycin-A` and value `0mM` by entering `mitomycin-A:0mM` into the search box and pressing Enter.
8. This should narrow down your search and find less results compared with the previous case.
9. Click on the Browse link  in one search result line of the last image (in the right-hand part of the centre pane) to navigate back to the main webclient.

Advanced search

The Advanced search in OMERO.web gives the opportunity to construct queries with Lucene syntax. These queries will be sent into the OMERO search (which is based on Lucene) unparsed. The possibilities include using logical operators (AND, OR, NOT, see workflow below) or fuzzy search (see Search Examples below).

1. Still on the search results page, click on the Advanced tab in the upper part of the left-hand side pane.



2. Enter `mitomycin-A:0mM AND name:VRAQ` which will narrow down your previous search for `mitomycin-A:0mM` to objects which also have VRAQ in their name.

3. Enter `mitomycin-A:0mM AND NOT name:VRAQ` which will reject all objects which have VRAQ in their name and find only the ones which are named differently.

Note: Spaces, stars, question marks and ^ characters are to be avoided in the Keys. If you already have these characters in Keys in your OMERO server, then try to replace them with underscores if you want to use the Search functionality on them. Spaces, stars, question marks and ^ characters in Values are acceptable, but should be avoided if possible, see Search Examples below.

Search examples

Considering the following setup of 13 separate images:

Table 1: Images with Key-Value pairs

Image ID	Image Name	Key	Value
1	Aurora1	GFP H2B	2 uM
2	Aurora2	GFP^H2B	2 uM
3	Aurora3	H2B	2
4	Aurora4	H2B	4
5	Aurora5	GFP-H2B	2-uM
6	Aurora6	GFP-H2B	2 uM
7	Aurora7	GFP_H2B	2_uM
8	Aurora8	GFP^H2B	2^uM
9	GFP	none	none
10	uM	none	none
11	H2B	none	none
12	2	none	none
13	Aurora13	H2B	2 uM

Basic Search tab:

- `GFP H2B:2 uM` finds images 1,2,3,5,6,7,8,9,10. In that case, the query is interpreted as `GFP OR H2B:2 OR uM`.
- `"GFP H2B":2 uM` throws an error. Do not use quotes around Keys!
- `GFP H2B:"2 uM"` finds images 1,2,5,6,7,8,9. In that case, the query is interpreted as `GFP OR H2B:2 uM` which prevents finding of image 3 with Value 2.
- `GFP^H2B:2 uM` finds images 1,2,3,5,6,7,8,9,10. In that case, the query is interpreted as `GFP OR H2B:2 OR uM`.
- `H2B:2` finds image 3.
- `H2B:4` finds image 4.
- `GFP-H2B:2 uM` finds images 1,2,5,6,7,8,10.
- `GFP-H2B:2-uM` finds images 5,6.
- `GFP-H2B:"2-uM"` finds images 5,6.
- `GFP-H2B:"2 uM"` finds images 5,6.
- `GFP-H2B:"2_uM"` finds images 5,6.
- `GFP_H2B:2_uM` finds image 7.

- `GFP^H2B:2^uM` finds images 1,2,3,5,6,7,8,9,10.
- `GFP` finds images 1,2,5,6,7,8,9.
- `GFP` with checkbox `Name` under `Restricted by Field` section checked finds image 9.
- `uM` with checkbox `Name` under `Restricted by Field` section checked finds image 10.
- `H2B` with checkbox `Name` under `Restricted by Field` section checked finds image 11.
- `2` with checkbox `Name` under `Restricted by Field` section checked finds image 12.
- `GFP*:2 uM` throws an error. Do not use wildcards in Keys!
- `H2B:*` finds images 3,4,13. The wildcard can be used in Values.
- `H2B:2*` finds images 3,13.

Advanced tab:

- `GFP^H2B:2^uM` and `GFP^H2B:2 uM` throw an error in Advanced tab. This is due to the different interpretation of the `^` character between the basic Search and Advanced tabs.
- As there is no `Name` checkbox in the Advanced tab, use `name:GFP` instead, which finds image 9.
- `Aurora2~0.85` finds 1,2,3,4,5,6,7,8. The `~` denotes a fuzzy search, which is possible only in Advanced tab. The number behind the `~` indicates the precision with which the result must match the query.
- `Aurora2~0.86` finds image 2.

The behaviour for the rest of the query examples in Advanced tab is the same as listed above for the basic Search tab.

Administrate Groups and Users

Description

This chapter will show how to manage groups and users using the graphical interface in OMERO.web and the command-line interface. Most of the following tasks below can only be done by users with some administrator privileges. We will show:

- How to manage groups, creating and editing a new/existing group.
- How to manage users, creating and editing a new/existing user.
- How to set up the OMERO server to be able to email all users.

Resources

- Documentation:
 - <https://docs.openmicroscopy.org/latest/omero/sysadmins/server-permissions.html>
 - <https://docs.openmicroscopy.org/latest/omero/sysadmins/restricted-admins.html>
 - <https://docs.openmicroscopy.org/omero/latest/sysadmins/cli/usergroup.html>
- Script for Command Line User/Group management
 - `create_groups_users.sh`
- File defining the User/Group setup used by the script
 - `create_groups_users_setup`

Setup

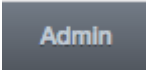
No setup needed for OMERO.web administration panel (see [Web Interface](#) chapter below) except working OMERO.web.

Command Line interface installation

The installation instructions can be found at [CLI installation](#).

Step-by-step

Administrate using the Web Interface

1. In your web browser, go to the server address provided.
2. Log in using the username and password provided.
3. In the top toolbar, click the **Admin** button . Note that the Admin button is only available for users with certain privileges: administrators and administrators with restricted privileges. If you are a user or a group owner, navigate to the section [Web Interface: Users change their own settings below](#).

Web Interface: Managing Groups

1. Click on the **Groups** tab. You can search for groups if desired.
2. To create a new Group, click on the **Add new Group** button. Note that the **Name** and **Permissions** fields are mandatory.
3. Click **Save**.
4. The new group will be shown in the list of Groups.
5. To edit a Group, click on the **Pencil** button .
6. You can add or remove members or group's owners or change group permissions.
7. Before removing a user from a group, it is preferable to move their data to another group or transfer ownership of their data to another user. Having a data owned by someone who is not a member of the group is not desirable.
8. Click **Save**.

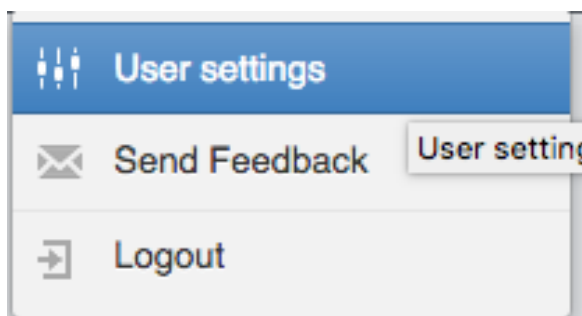
Web Interface: Managing Users

1. Click on the **Users** tab.
2. You can search for users if you wish.
3. OMERO.web denotes the user categories using small helpful icons:
 - Users with administrator privileges have a **tools icon** .
 - Active users have an icon with **blue circle** .
 - Inactive users have a **lock icon** .
 - LDAP users have a **red hexagon** .

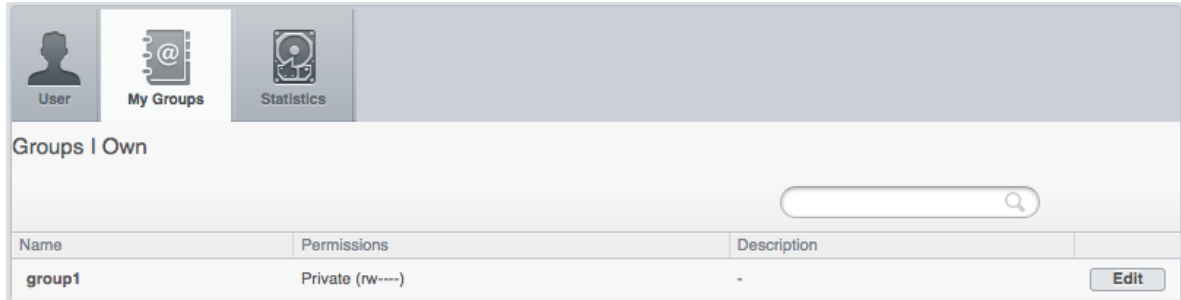
4. To create a new user, click on the `Add new User` button.
5. Mandatory fields are highlighted in red.
6. You can select the role of the user to be:
 - `User` (no special privileges).
 - `Administrator` (this means full administrator).
 - `Administrator with restricted privileges`.
7. If you choose the role to be `Administrator with restricted privileges`, you must also select the privileges in a subsequent menu. Hover with mouse over the checkboxes to see short descriptions of the privileges. Creating an administrator with restricted privileges allows to give some limited rights to some trusted users e.g. to allow a facility manager to import data for other users. It is currently preferable to create users with such roles via the OMERO.web Interface. More about `Administrator with restricted privileges` can be found in [this OMERO documentation section](#).
8. Click `Save`.
9. To edit a `User`, click on the `Pencil` button  to the right of the line with the name of the user. You can add/remove the `User` to/from a group or modify the roles.
10. Click `Save`.

Web Interface: Users change their own settings

1. Note that these features are not limited to administrators, **any** user can change their settings in the manner described here. Furthermore, this is the preferred way for Group Owners to manage their groups.
2. In OMERO.web, click in the top-right corner of the webclient, click on your name, then, in the dropdown menu, click on `User settings`.



3. In the interface that appears, you can change your password and default group. Default group is the group you log in to by default when logging to OMERO. Your data in your default group is what you typically see immediately after logging in for example to OMERO.web, whereas your data in your other (non-default) groups have to be explicitly navigated to.
4. **For group owners only:** You can now navigate to the group(s) you own by clicking onto `My Groups` tab.



5. Identify the group you want to edit in your group list and click on Edit button.
6. You can now add or remove group members, add members as group owners (a group can have many owners, besides yourself). When removing users from the group, make sure that the data owned by a user is moved or transferred to another user before removing the user from the group.
7. You can also change the permissions level of your group. Note though that this is an action which needs careful thinking, especially if you are going from more permissive group types towards less permissive ones.

Administrate using the Command Line Interface (CLI)

Typically, the administration of Groups and Users in OMERO is done in OMERO.web (see section above), as it is more user friendly. The Command Line Interface (CLI) cannot offer the easy quick overview, filtering and searching and intuitively named buttons and tabs. For creation of administrators with restricted privileges, there are several key features missing from the CLI which are present in OMERO.web. Nevertheless, some features for handling LDAP users are implemented only in the CLI. Further, the CLI offers an environment in which custom bash scripts for user/group creation and maintenance can be executed. One example of such script can be taken from [create_groups_users.sh](#). The script consumes a file [create_groups_users_setup](#) in which a certain user-group setup is defined.

Command Line: Managing Groups

1. By default when creating a group, its permissions level is set to private. To create a new read-annotate group Lab1, run:

```
$ omero group add Lab1 --type=read-annotate
```

2. Or, you can define the permissions of the new group in a different way:

```
$ omero group add Lab1 --perms='rwra--'
```

3. To list all the groups and save the output for example in a CSV file:

```
$ omero group list --style csv > groups.csv
```

4. To add an existing user user-1 to the Lab1 group and make that user a group owner (the option `--as-owner` is not needed when adding a member), run:

```
$ omero group adduser user-1 --name=Lab1 --as-owner
```

5. Let us add trainer-1 as an owner of the group too:

```
$ omero group adduser trainer-1 --name=Lab1 --as-owner
```

6. To remove user-1 from the list of owners (user-1 will still be a member of the Lab1 group):

```
$ omero user leavegroup Lab1 --name=user-1 --as-owner
```

7. Note that the previous command when run without the `--as-owner` flag would remove the `user-1` from the group completely. Thus, it is an alternative to the following command.

8. To remove `user-1` from the `Lab1` group, you can also run:

```
$ omero group removeuser user-1 --name=Lab1
```

9. To edit the `Lab1` group, first determine its ID:

```
$ omero group info --group-name Lab1

id \ | name \ | perms \ | ldap \ | # of owners \ | # of members
-----+-----+-----+-----+-----+-----
653 \ | Lab1 \ | rwra-- \ | False \ | 0 \ | 0
```

10. Change the group name to `LabN`:

```
$ omero obj update ExperimenterGroup:653 name='LabN'
```

11. Let us reset the name back to `Lab1` to simplify the rest of the workflow.

12. Change the group's permissions to read-write:

```
$ omero group perms --perms='rwrw--' --name='Lab1'
```

Command Line: Managing Users

1. Create a new user with login name `lpasteur` and at the same time add this user (with first and last name `Louis Pasteur`) to the `Lab1` group:

```
$ omero user add lpasteur Louis Pasteur --group-name Lab1
```

2. Let us now add the user to another group:

```
$ omero user joingroup Lab2 --name=lpasteur
```

3. To edit the user and for example add an email address, first determine the user's ID:

```
$ omero user info --user-name lpasteur
```

4. Add an email address (supposing the ID of the user is 123):

```
$ omero obj update Experimenter:123 email='lpasteur@demo.co.uk'
```

5. Make a user inactive. User **cannot** be deleted but it is possible to prevent a user from logging in. For that, we need to remove the user from the user group (an internal OMERO group):

```
$ omero user leavegroup user --name=lpasteur
```

6. To reactivate the user:

```
$ omero user joingroup user --name=lpasteur
```

Command Line: Managing LDAP Users

If LDAP authentication is configured on your OMERO.server, the OMERO.server synchronizes the user list with an LDAP server, thus enabling an easy user creation and maintenance. It is possible to convert non-LDAP OMERO users to LDAP authentication using the command `omero ldap setdn`. See further information in the links under the Resources section of this guide. See [LDAP authentication](#) and [LDAP plugin design](#).

Typically, it is impractical to synchronize the OMERO groups with LDAP groups. In such case, the OMERO.server can be configured in such a way that LDAP users when they first log in to OMERO will be added to a specific private OMERO group (let us call this group `My Data`). This situation is further explored in the example below.

The administrator or administrator with restricted privileges can add an LDAP user to OMERO even before the user have ever logged in to OMERO:

1. First create the existing LDAP user as OMERO user. In the example below the user name is `enoether`:

```
$ omero ldap create enoether
```

2. The user is now a member of the `My Data` group in OMERO. Then, if needed, add the user to the `Lab1` group:

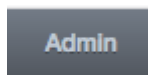
```
$ omero group adduser enoether --name=Lab1
```

3. Note that it is advisable to clarify the OMERO group membership situation of the LDAP users soon after they joined OMERO. This can be done for example by adding the new user to their lab group (e.g. `Lab1`) in OMERO as well and by changing the default group of such user in OMERO to be their lab group. See above for how to change the default group of a user. Otherwise, the new LDAP&OMERO users might be importing their data into the `My Data` group for some period of time, without realizing the data are not accessible to their colleagues in the lab group for cooperative purposes because `My Data` is a private group.

Set up OMERO server to email users

If you are a full administrator or an [administrator with restricted privileges](#) with any or no privileges, you can email OMERO users. This can be helpful for example to inform users about downtimes, new features or imminent changes regarding OMERO.

1. In cooperation with you OMERO.server system administrator, consult the [documentation on email in OMERO](#).
2. Once the OMERO.server is configured, log in to OMERO.web and in the top toolbar, click the **Admin** button



3. Click on the **Email** tab.
4. Choose the appropriate options, enter the email subject and message. Note that depending on the number of users you are choosing to email, the action might take a long time to finish. You **must** keep the session of OMERO.web alive (i.e. doing actions still being logged in OMERO.web) until the **Activities** dropdown menu (icon to the left of the **Search** in the top bar of OMERO.web) reports that all emails were sent.






Email Announcement

All Users: ☐

Users:

Groups:

Cc:

Subject:

Message:

Include inactive users: ☐

Note: Checking 'Include inactive users' enables sending email to inactive users. If this is not checked, any selection above will ignore inactive users. E.g. If a group selected above has an inactive user, they will not be emailed unless this option is checked. Selecting an inactive user above and not checking this option will result in an error.

Send

Activities

Clear List

✓

Send email

Sent 1 email of 1

5. Click Send button.

Prepare data for publication using OMERO

Users can publish Image data to the world using OMERO. Here we describe some steps to facilitate that. The steps are to be done by an administrator or restricted administrator in OMERO.

Description

We will show:

- How to set up configurations in OMERO.web for establishing of a `public` user.
- How to set up a `public` group.
- How to get the data into the `public` group.
- How to open the `public` group to the outer world.

Setup

OMERO.server has been installed and provisioned using an [Ansible playbook](#).

OMERO.server configurations for publication are described in the following chapter of the [publication in OMERO documentation](#):

- [Configuring public user](#).

Resources

- The publication in OMERO is described in [the documentation](#).

Step-by-Step

1. Consult with the system administrator of your OMERO.server the [configuring of public user on the server](#). Read the possibilities about how to restrict the access to the data in public group using url filtering in the [publication with OMERO documentation](#)
2. Establish a `public` group with read-only permissions. See the [publication example documentation](#) and the *Administrate Groups and Users* for how to do it. Note: The `public` group is just an ordinary read-only group, only a subsequent addition of the `public` user to this group makes it public.
3. Think about the best strategy of data layout in the `public` group, see suggestions about it in the [Group setup part of publication example documentation](#).
4. Duplicate the data to be published as described in *Data management and cooperation* either yourself, or instruct the lead scientist on the publication to do this. It is recommended to duplicate the annotations as well, which is done by the duplication plugin by the default, as this will facilitate the preservation of the linkage between the data and annotations during the subsequent move to the public group.
5. Consult with the scientists intending to publish their data about how to Move the data into the `public` group, or move them yourself. Some helpful hints about the data Move can be found in the [Data migration chapter of the documentation](#). The move between groups is made easier by making duplicates of the data prior to the move and then moving these duplicates (see previous step). Also see the chapter *Move data between groups* in *Data management and cooperation* for how to Move the data between groups. The other option is to import the data afresh into the `public` group. Note: there is no possibility to copy or link the data in a single step into public group from another group in OMERO at the moment. The duplication of the data and subsequent move of the duplicate into the public group is the recommended workflow.
6. Once you are happy with the setup of the `public` group, add the `public` user into the `public` group as described in [the documentation](#). See *Administrate Groups and Users* for how to add users into groups in OMERO. The addition of the `public` user will make the data in the `public` group visible to the world.

3.1.3 Server-side scripts

This section describes how to run, write and manage the OMERO server-side scripts. Server-side scripts can be written in different programming languages but you will need to invoke the scripts using a Python wrapper.

Contents:

Run server-side scripts

The server-side Python scripts on OMERO give an opportunity to run the analysis close to the data.

Description

The server-side Python scripts on OMERO can be accessed via both the OMERO.web and OMERO.insight user interfaces. The scripts are typically uploaded by an administrator or a restricted administrator of the OMERO.server and can be run by any user on the server.

First, we show an example of a server-side script with a customized UI.

Secondly, two examples are presented showing the experience from the user interface of OMERO.web (Batch ROI export and Kymograph).

Setup

No specific setup needed.

Resources

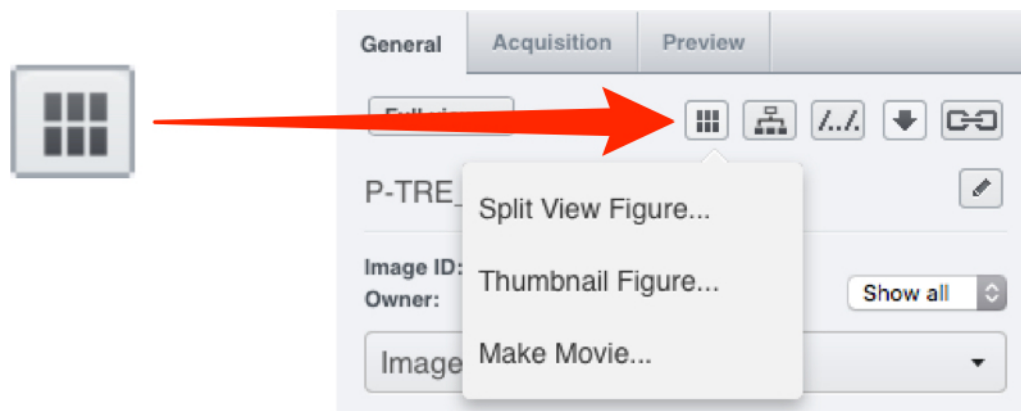
- [siRNAi-HeLa images](#) for the Batch ROI Export.py script.
- Images for Kymograph script were downloaded from [Septin GTPases spatially guide microtubule organization and plus end dynamics in polarizing epithelia](#) (Bowen et. al. Journal of Cell Biology 194 (2): 187).

Step-by-Step

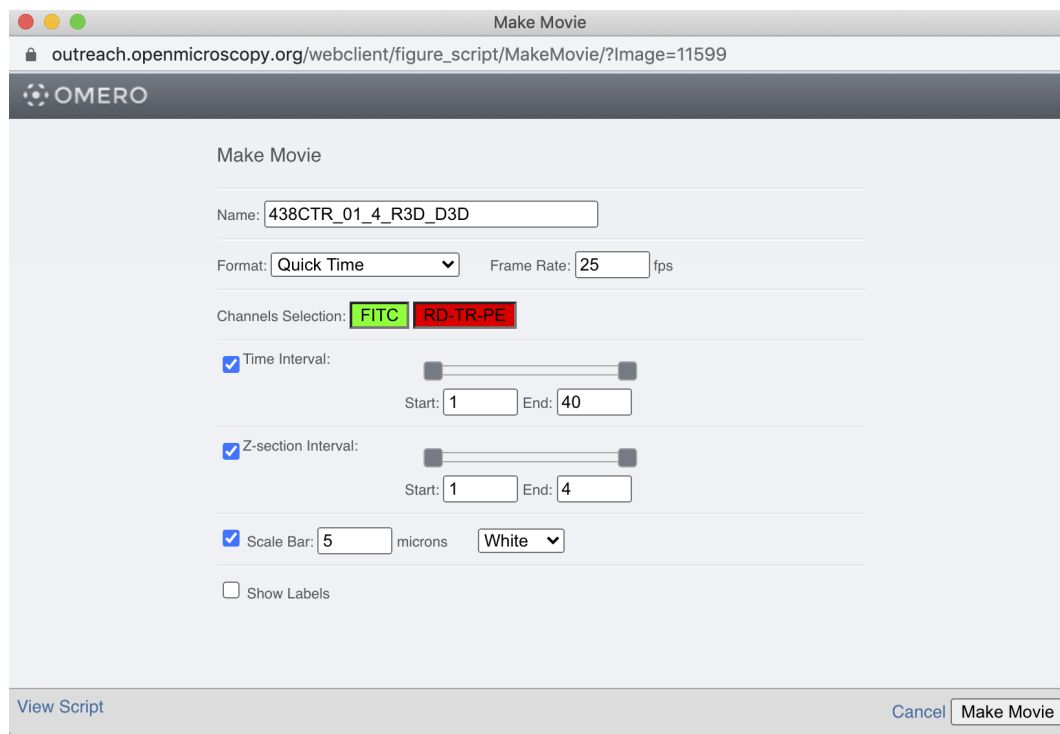
Example 1

In this example, we create a movie server-side using the script [Make_Movie.py](#). For this script, there is a customized UI for the script that will override the default UI available from the script itself.

1. Select a time-lapse Image in the OMERO.web client and click on the Publish icon in the toolbar.

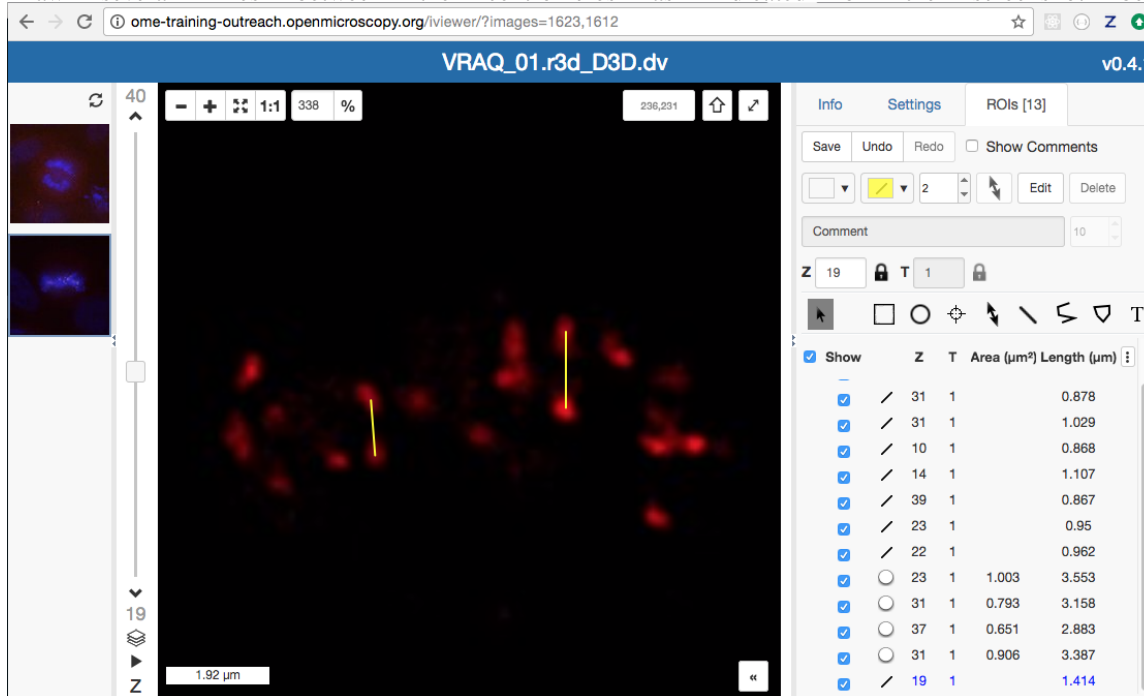


2. Select **Make Movie...** from the drop-down menu to create a movie from an Image.
3. Select the format from the **Format** drop-down.
4. Use the **Frame Rate** drop-down to select the frame rate.
5. Set the desired time interval if appropriate.
6. Set the desired Z-section interval if appropriate.
7. Click the button **Make Movie** to start the movie creation server side.

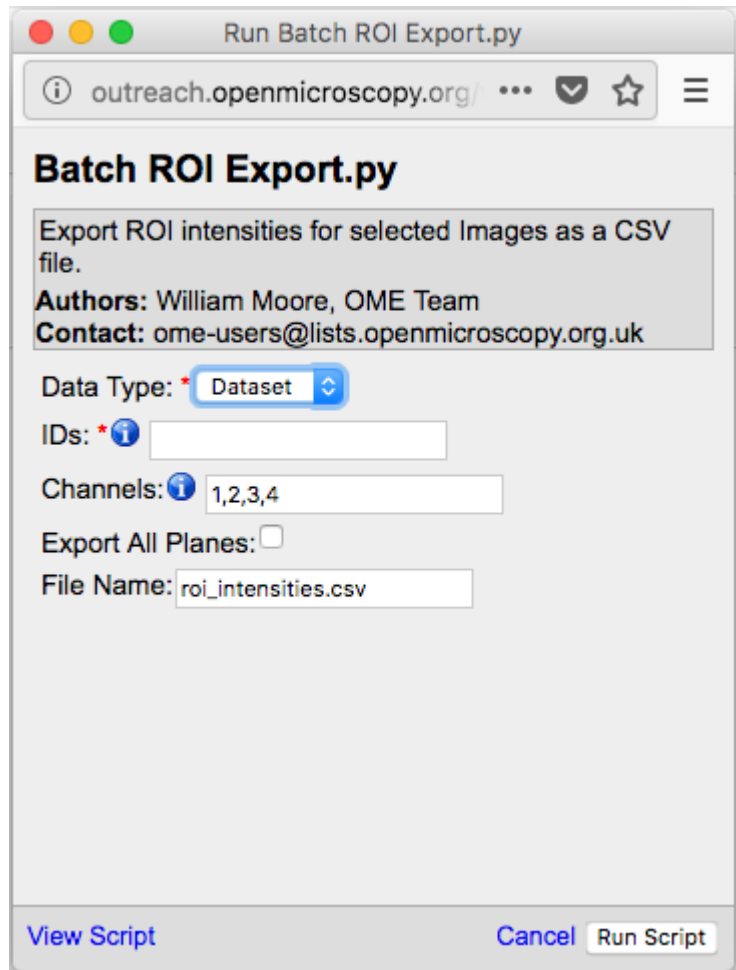


Example 2

1. We will now analyse the ROIs created in OMERO.iviewer using a server-side script.
2. Go to the siRNA-HeLa Dataset and open several images whose name start with VRAQ... in OMERO.iviewer.
3. We want to measure the distance between Centromeres, stained with ACA in the 4th Channel. Turn on ONLY the 4th channel and open the ROIs tab to on the right-hand pane.
4. Draw several lines between the centromeres as indicated on the screenshot below.



5. Try to identify centromere pairs:
 - Select the Line tool and draw a line between the centres of the centromeres.
 - In the ROIs table, in the Comments column, click the 3 dots in the column header and choose to Show Area/Length.
6. Do the same now on several of the Images whose names start with IN..., which are in the Metaphase state.
7. Select Dataset siRNA-HeLa.
8. Click the Script button in the top-right of the page .
9. Select *export_scripts* > *Batch_ROI_Export...*
10. In the dialog that pops up, click on View Script to view the Python code.
11. Search for “idr”# to find the code block that selects the filter_channel based on Dataset and Channel names.
12. Scroll to the bottom of the script to see where the input parameters are defined, such as Data_Type and IDs. Note how these appear in the script dialog and are auto-populated with the currently-selected Datasets or Images.

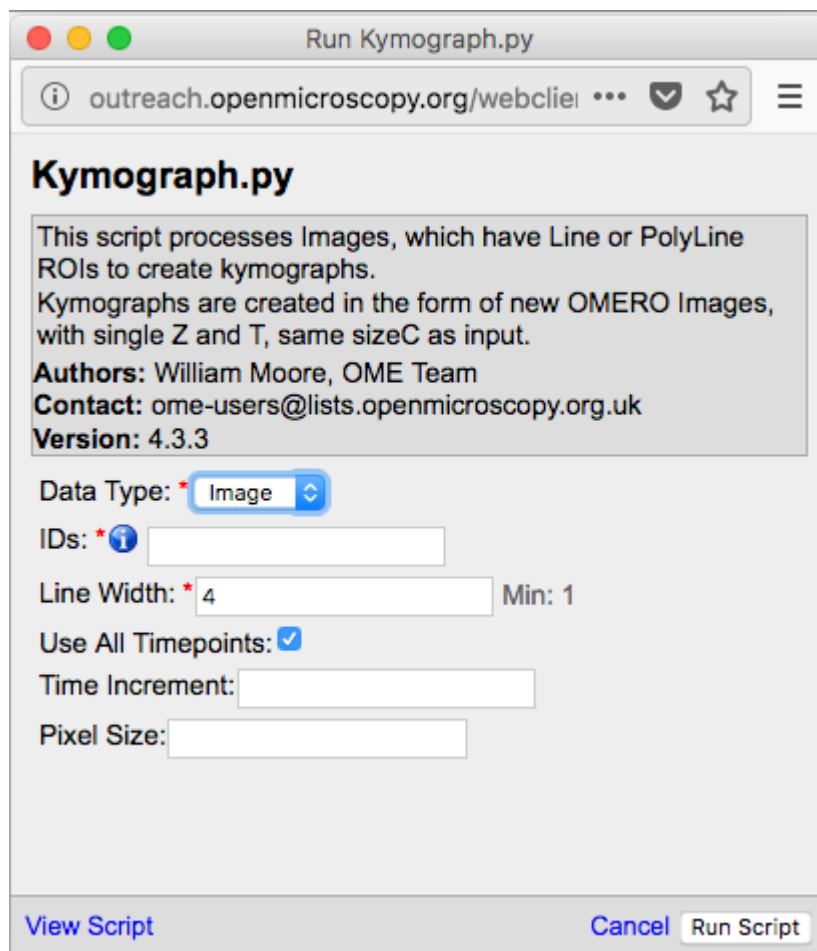


13. The script will process all the Images in the selected Dataset and data can be exported as CSV file, with one table row per Shape/Channel.
14. Click Run Script.
15. The status of the processing is displayed in the Activities dialog .
16. When the script has completed, it will show in the Activities dialog and allow you to download the CSV file. Download this and open it, e.g. in Excel, to see the output data for all the shapes.

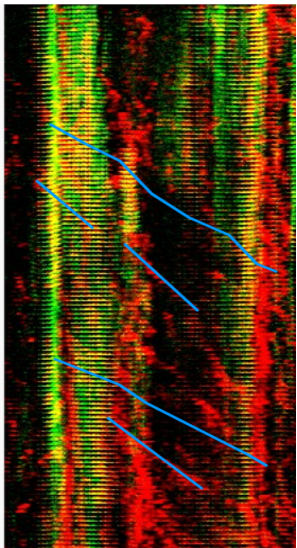
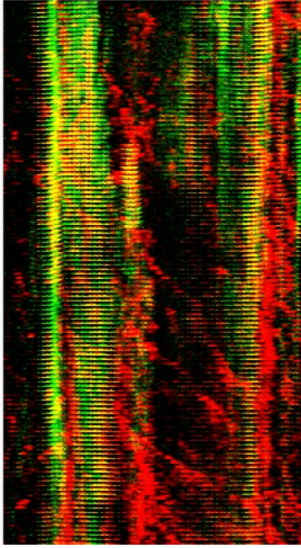
Example 3

We will now use another server-side script for creating a Kymograph from an Image in OMERO.

1. Go to the Dataset Kymograph.
2. Select the Image inside the Dataset.
3. Double-click to open the Image in OMERO.iviewer and draw one or more lines along microtubules which seems to have the most persistent trafficking of objects along them.
4. Save the line(s).
5. Go back to the webclient. Click the Script button in the top-right of the page .



6. Select workshop_scripts > Kymograph...
7. The script will create a new image (=Kymograph) where the pixels under the line region you have drawn previously will be collated into this image timepoint by timepoint. The row of pixels from the first timepoint will be on the top of the new Kymograph Image.
8. Note: the direction in which you have drawn the line ROI on the original image matters with respect to the orientation of the stripes composing the Kymograph image. The start of the original line is on the left of the Kymograph Image, the end on the right.
9. Open the new Kymograph image in OMERO.iviewer.
10. Find some tracks (typically red stripes going under angles across the image, see screenshot below).



11. Draw some lines over these tracks and save them.
12. Go back to the webclient, select the Kymograph Image and select the script analysis > Kymograph analysis...
13. Run this script. The Kymograph analysis script will produce a CSV file attachment on the Kymograph Image.
14. Open the CSV in Excel for example and verify the speeds of the observed particles in the original image.

How to write a server-side script

The server side script follows some simple steps so that a simple UI can be generated automatically.

In this section we will show how to write a simple script and upload it to the OMERO.server: The user will specify a Dataset and the script will be uploaded by the OMERO.server administrator (or administrator with restricted privileges) to the OMERO.server. If the script is uploaded as “official”, it can be run by all users on the OMERO.server after upload. The script demonstrated here is very simple, it just loads the images contained in a dataset.

See https://raw.githubusercontent.com/ome/omero-guide-python/master/scripts/hello_world_server.py.

1. Click on the link above and copy and paste the script into a text editor of your choice.

2. Study the composition of the script - the script is taking a Dataset ID and produces an output of all the images contained in that dataset. This is of course just a springboard for further work with the images in a more advanced script.
3. (demo only) The script can be immediately uploaded to the OMERO.server in this present state, using

- OMERO.insight, the sixth icon from the left, top-left of the UI  (Note that only admins and restricted admins will see this icon in OMERO.insight).
- Command line interface (CLI) using the command.

```
$ bin/omero script upload test-script1.py --official
```

4. After the demonstrator uploaded the script, you can
 - Go to OMERO.web and select any dataset in the left-hand tree
 - Above the central pane, find the “cogs” icon with scripts
 - Find the newly uploaded script.
 - Click on the menu item, the script dialog will be already pre-populated with the ID of the selected Dataset.
 - Click Run.

How to manage a server-side script

Please refer to <https://docs.openmicroscopy.org/latest/omero/developers/scripts/user-guide.html> for how to write other simple scripts, execute, edit and delete them.

Annotation scripts

This section describes how to use the annotation scripts, which provide additional support for Key-Value pairs (and Tags) to import, export, delete or convert between Namespaces by batch.

Setup

Setting up server-side scripts for OMERO has to be done by an administrator of your server. Please contact them if the scripts are not present or in a different version as the one presented in this guide, and redirect them to this page.

In the OMERO.web client, the scripts menu is located above the central pane to the left of the search bar. Please refer to the [script documentation](#) for more information.

Installing the scripts

The Key-Value pairs scripts have been updated in 2023 to enable them to annotate all the common object types in OMERO:

- Image
- Dataset
- Project
- Well

- Run/Acquisition
- Plate
- Screen

Check that the version of the currently installed scripts are of 2.0.0 and higher.

If not, you can find the scripts [here](#) and follow [these](#) instructions to install them.

Configuring the Export script

Note that the Export script can be fine-tuned to your specific OMERO.web installation. For that, open the script and find the global parameter named “WEBCLIENT_URL”, and set it to the URL of your webclient, without forgetting to add the http/https protocol in front:

```
https://your-omeroweb-adress.org/webclient
```

This will allow the script to return a link for the created CSV for a direct downloading. If not configured, the CSV can be found as an attachment of the first parent object passed in parameter of the export script (the file is attached regardless of whether the URL is set or not).

To set this configuration, you can use [OMERO CLI](#):

```
> omero script list
(out) ...
      1234 | /omero/annotation_scripts/Export_to_csv.py
      ...
> omero script edit 1234
```

Walkthrough

The aim of the four Annotation scripts is to provide a way to edit Key-Value pairs (MapAnnotations) and Tags (TagAnnotations) by batch:

- Import Key-Value pairs and Tags as Annotations from a CSV file
- Export Key-Value pairs and Tags to a CSV file
- Delete Key-Value pairs
- Convert the Namespaces of Key-Value pairs

This walkthrough should give you an understanding of the scripts and help you to manage your annotations.

Object selection with the scripts

The object selection logic is the same for all four scripts. In OMERO there are two distinct hierarchies:

- **Projects** → **Datasets** → **Images**
- **Screens** → **Plates** → (**Wells** or **Acquisition/Run**) → **Images**

More details about selecting High-Content-Screening are given elsewhere (see [Selecting High-Content-Screening](#)).

The scripts offer 4 ways of selecting the objects to work with.

Direct selection

Simply select the all the desired objects. Opening the script after selecting the object will prefill the script parameters with the right object type and IDs:

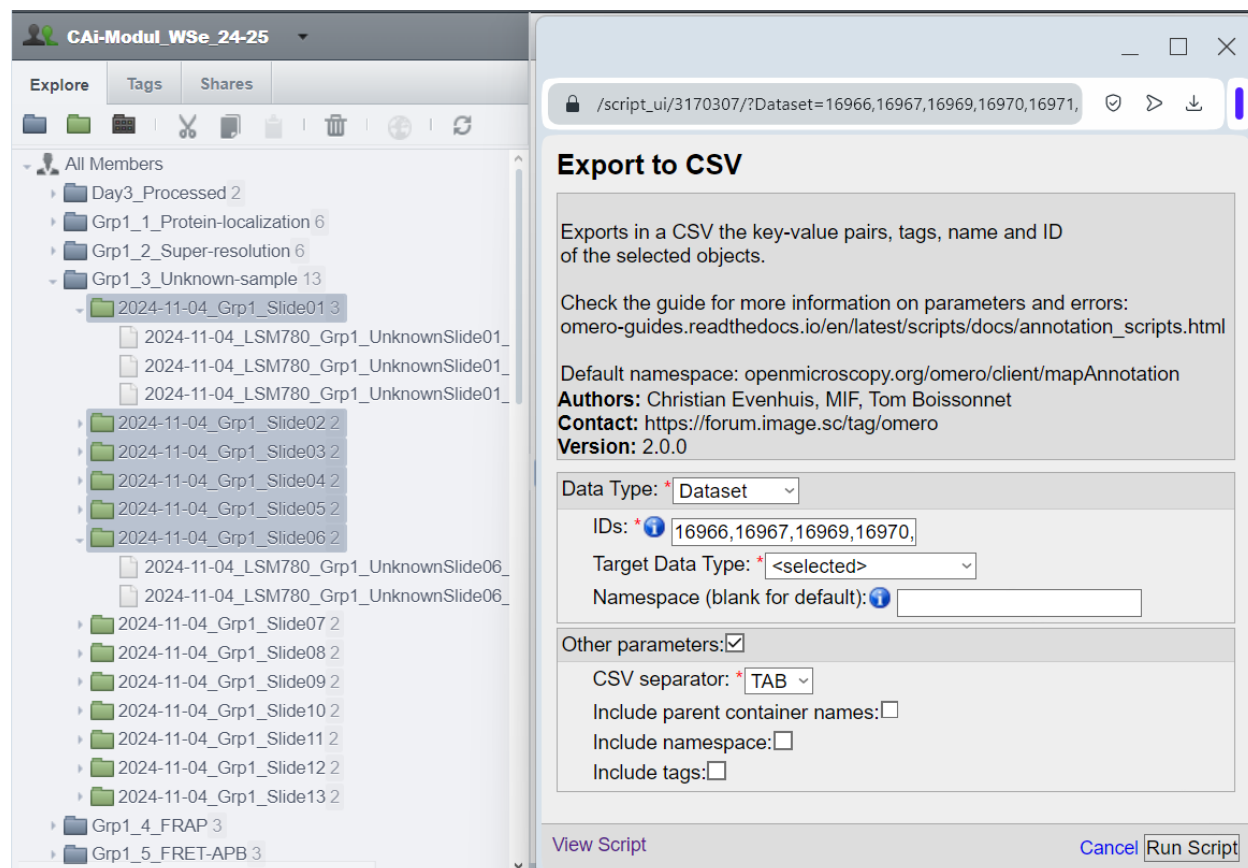


Fig. 1: Selection of multiple datasets and the auto-filled Export to CSV script.

For the direct selection, leave the Target Data Type as “<selected>” (in this example, choosing **Dataset** would also work).

Children selection

Instead of selecting the objects one by one, we can select the parent object and indicate in the “Target Data Type” the children objects we want to select.

In this example, all the **Images** found under the **Project:701** are selected.

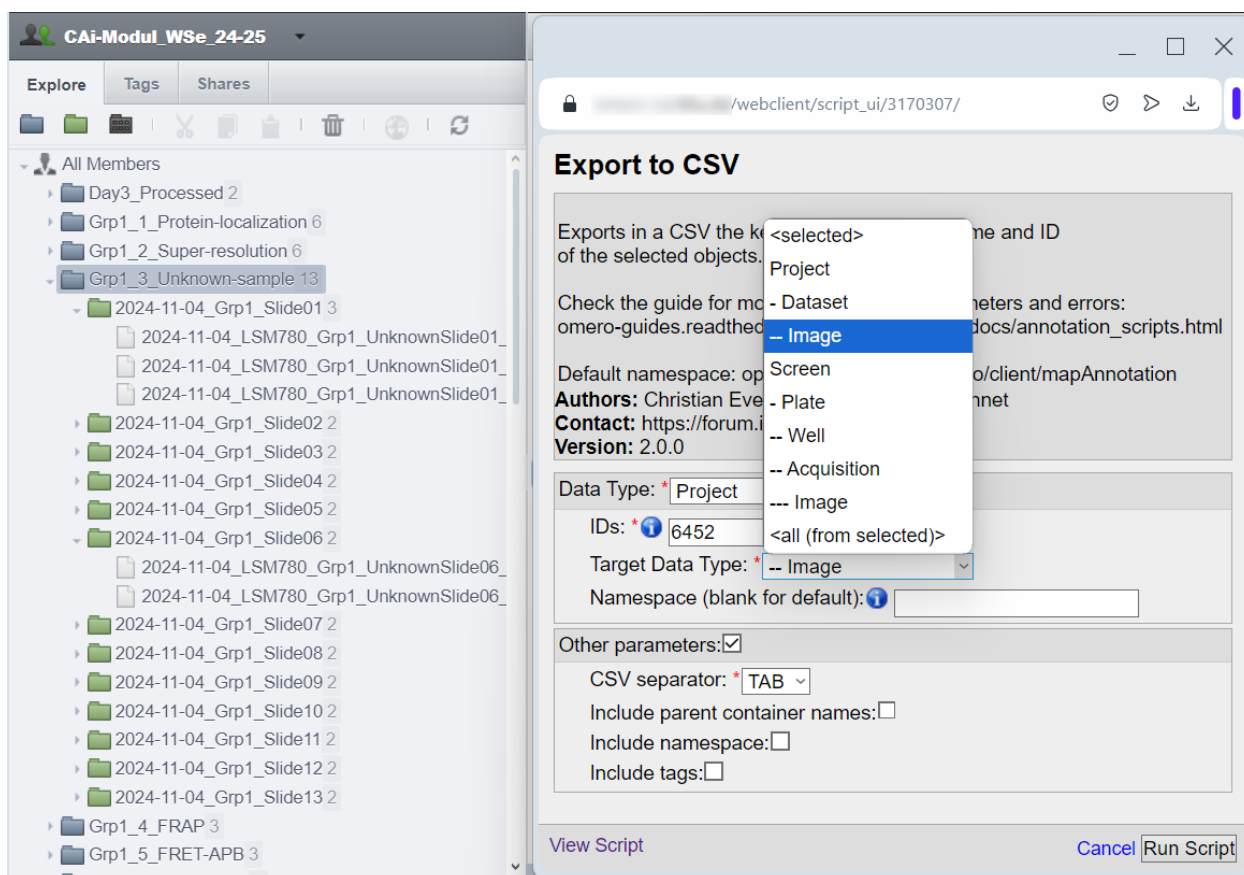


Fig. 2: Selection of all the images of a project and the auto-filled Export to CSV script.

Tag selection

You can use the Tags to select the images or other objects you want to work with. Do not confuse this option with the exporting of Tags also provided by the scripts. The usage of Tags for object selection grants additional flexibility. You can for example select objects attached to different projects or belonging to different owners.

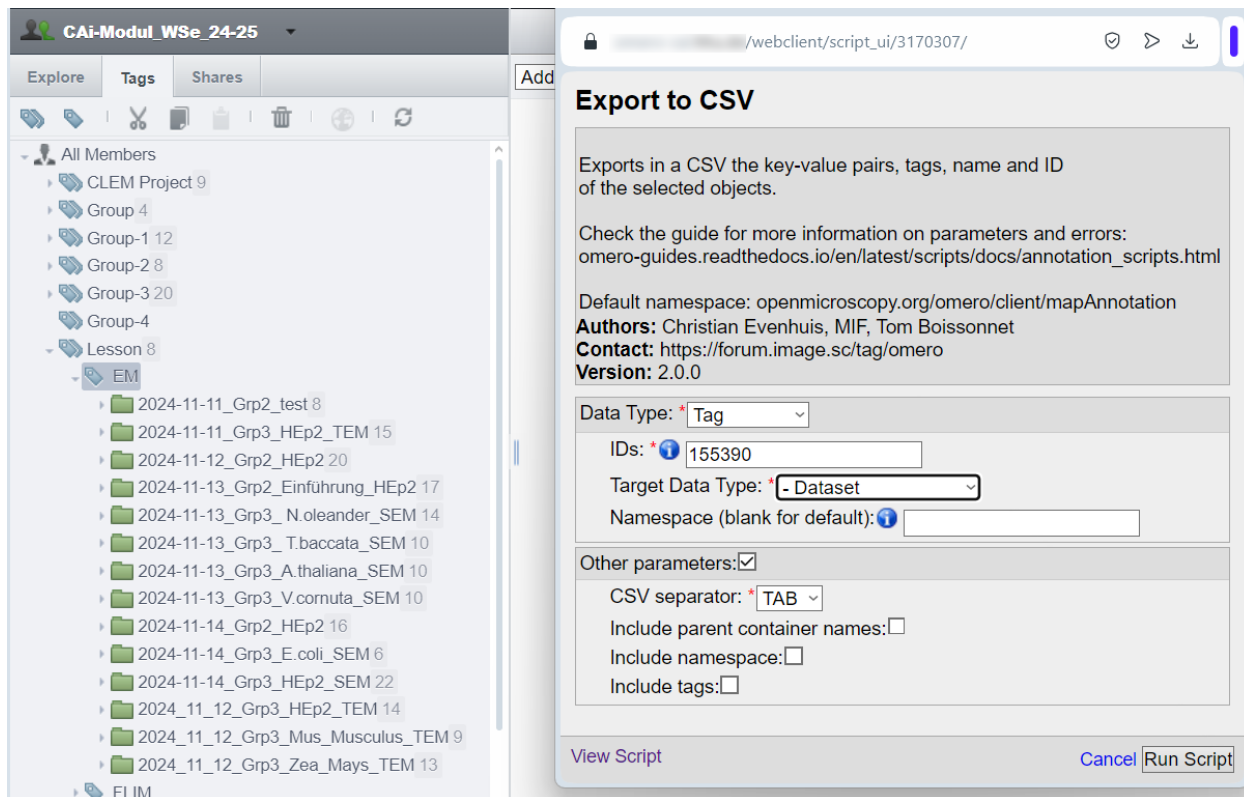


Fig. 3: Selection of a subset of datasets from a Tag and the auto-filled Export to CSV script.

In this example, all the **Datasets** found under the **Tag:155390** are selected. Note that choosing “<selected>” for “Target Data Type” will result in an error in this case.

All from selected

The fourth option is to apply the scripts to all of the selected objects and the children objects. Selecting a Project and using the option **<all (from selected)>** will result in applying the scripts to the selected project, to the linked datasets and the linked images below. Note that this option does not exist for the import of annotation from CSV.

Exporting annotations

Exporting annotations creates a CSV file with headers compatible with the import script. It can be used for:

- Export Key-Value pairs (along with data like Namespaces and Tags) for modification or transferring annotations between groups.
- Export a list of objects with their names and IDs (and excluding Key-Value pairs by specifying an unused Namespace).

OBJECT_ID	OBJECT_NAME
16966	2024-11-04_Grp1_Slide01
16967	2024-11-04_Grp1_Slide02
16969	2024-11-04_Grp1_Slide03
16970	2024-11-04_Grp1_Slide04
16971	2024-11-04_Grp1_Slide05

Fig. 4: *Exported CSV with only the object ID and name*

Importing annotations

Starting from the file we exported as explained in the previous section, we proceed to edit it within a spreadsheet editor, adding more columns to the CSV. You can also insert a row on the top of CSV to specify the Namespace for each Key:

[illegible]

Fig. 5: Result of populating the CSV shown in a text editor

We added several columns to annotate our dataset with Key-Value pairs following the REMBI guidelines (after saving the document, our TAB separator became ,).

We proceed and start the script “*Import from CSV*”.

There are many parameters here that could help you fine tune the way to import annotations. More on that in the parameter description section of this guide.

The Key-Value pairs were added to 5 datasets out of the 11 present in this project, as expected. We also obtain three sets of Key-Value pairs, one for each Namespace used in the CSV.

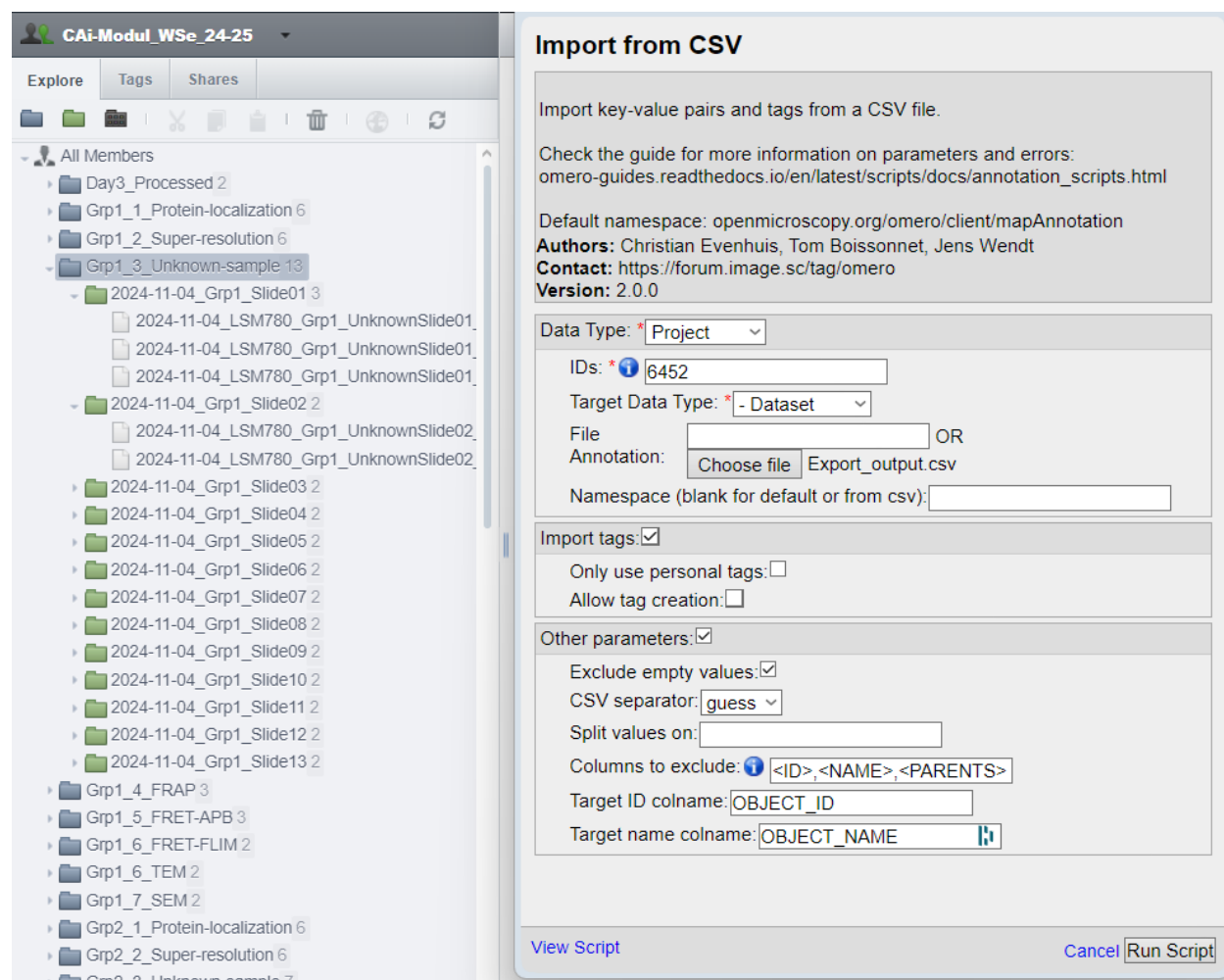
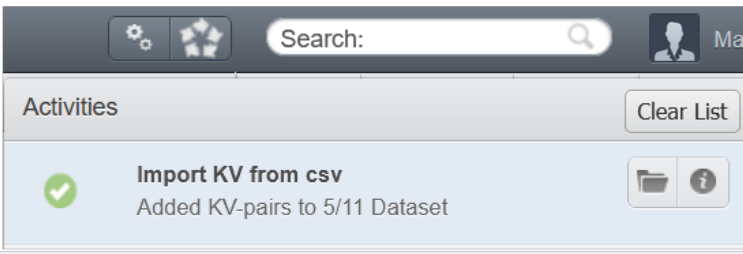


Fig. 6: Selection of all the datasets of a project and the Import from CSV script.



REMBI/Biosample
Added by: Mary Mayperson

biological entity	HEp2 cells
biological entity term accession number	http://purl.obolibrary.org/obo/BTO_0000976
biological entity term accession number source REF	BRENDA tissue / enzyme source
species	human
species term accession number	http://purl.obolibrary.org/obo/NCBITaxon_9606
species term accession number source REF	NCBI organismal classification

REMBI/Specimen
Added by: Mary Mayperson

preparation method	#EXP00114_20230615_CAI_Test_Staining_for_Practicals
staining	#EXP00114_20230615_CAI_Test_Staining_for_Practicals
channel1 - content	DAPI
channel1 - content - ontology	http://purl.obolibrary.org/obo/NCIT_C122989
channel1 - biological entity	DNA, mainly nucleus
channel2 - content	phalloidin - 488
channel2 - biological entity	filamentous Actin
channel3 - content	anti-Tom20(mouse), antiMouse-AlexaFluorophore568
channel3 - biological entity	mitochondrial import receptor
channel4 - content	antiVimentin(rabbit), antiRabbit-StarRed
channel4 - biological entity	intermediate filaments

REMBI/Study component
Added by: Mary Mayperson

imaging method	confocal laser scanning microscopy
imaging method term accession number	http://purl.obolibrary.org/obo/CHMO_0000089
imaging method term accession number source REF	chemical methods ontology

Fig. 7: The script output (5 entries in the CSV matched to 5 dataset out of 11) and the resulting Key-Value pairs annotation.

Converting the Key-Value pairs Namespace

Key-Value pairs are grouped under categories called Namespaces. This allows flexibility to distinguish or isolate annotations, such as exporting or deleting only those associated with a specific Namespace.

Note: When you create Key-Value pairs in OMERO.web, you use Namespaces without noticing it: OMERO assigns by default the “Client namespace” (`openmicroscopy.org/omero/client/mapAnnotation` in full), a special Namespace recognized by OMERO.web.

The “Client namespace” (see note above) is the only Namespace that allows editing of Key-Value pairs in the OMERO.web interface. If a Key-Value pair uses a different Namespace, its Namespace must first be converted to the “Client namespace” before editing. These annotation scripts will always default to the “Client namespace” if the Namespace fields are left empty.

Convert Key-Value pairs namespace

Converts the Namespace of Key-Value pairs.

Check the [guide](https://omero-guides.readthedocs.io/en/latest/scripts/docs/annotation_scripts.html) for more information on parameters and errors:
omero-guides.readthedocs.io/en/latest/scripts/docs/annotation_scripts.html

Default Namespace: `openmicroscopy.org/omero/client/mapAnnotation`
Authors: Tom Boissonnet
Contact: <https://forum.image.sc/tag/omero>
Version: 2.0.0

Data Type: *

IDs: *

Target Data Type: *

Input Namespace(s) (blank for default):

New Namespace (blank for default):

Create new and merge: ☐

[View Script](#) [Cancel](#) [Run Script](#)

Fig. 8: The script to convert the Namespace of Key-Value pairs annotations.

And here is our five Key-Value pairs annotations with converted Namespace:

Once the Key-Value pair has been converted to the “Client Namespace,” it can be edited in the OMERO.web interface.

Added by: Mary Mayperson		+ [icon] [icon] [icon]	
Key		Value	
imaging method		confocal laser scanning microscopy	
imaging method term accession number		http://purl.obolibrary.org/obo/CHMO_0000089	
imaging method term accession number source REF		chemical methods ontology	
REMBI/Biosample			
Added by: Mary Mayperson			
biological entity		HEp2 cells	
biological entity term accession number		http://purl.obolibrary.org/obo/BTO_0000976	
biological entity term accession number source REF		BRENDA tissue / enzyme source	
species		human	
species term accession number		http://purl.obolibrary.org/obo/NCBITaxon_9606	
species term accession number source REF		NCBI organismal classification	
REMBI/Specimen			
Added by: Mary Mayperson			
preparation method		#EXP00114_20230615_CAI_Test_Staining_for_Practicals	
staining		#EXP00114_20230615_CAI_Test_Staining_for_Practicals	
channel1 - content		DAPI	
channel1 - content - ontology		http://purl.obolibrary.org/obo/NCIT_C122989	
channel1 - biological entity		DNA, mainly nucleus	
channel2 - content		phalloidin - 488	
channel2 - biological entity		filamentous Actin	
channel3 - content		anti-Tom20(mouse), antiMouse-AlexaFluorophore568	
channel3 - biological entity		mitochondrial import receptor	
channel4 - content		antiVimentin(rabbit), antiRabbit-StarRed	
channel4 - biological entity		intermediate filaments	

Fig. 9: The script output (5 dataset had annotations with the default Namespace) and the resulting Key-Value pairs annotation.

Deleting Key-Value pairs

It turns out that there are more than a few edits to do on our Key-Value pairs, and converting back and forth the Namespace seems too tedious this time. Let's instead export what we have so far (we have done edits that we don't want to lose), delete "old" the Key-Value pairs, and reimport when we are done fixing them.

Using the same selection inputs for both scripts, we make sure that we have the Key-Value pairs on our computer before deleting them.

Warning: To export or delete multiple Namespaces simultaneously, you can use the * wildcard to include all Key-Value pairs, regardless of their Namespaces. However, use this feature with caution, as deleted annotations cannot be recovered without a backup.

Because there is no undo with the deletion of data from OMERO, think twice while checking the confirmation box and clicking the "Run script" button.

Export to CSV

Exports in a CSV the key-value pairs, tags, name and ID of the selected objects.

Check the guide for more information on parameters and errors:
omero-guides.readthedocs.io/en/latest/scripts/docs/annotation_scripts.html

Default namespace: openmicroscopy.org/omero/client/mapAnnotation
Authors: Christian Evenhuis, MIF, Tom Boissonnet
Contact: <https://forum.image.sc/tag/omero>
Version: 2.0.0

Data Type: * Project

IDs: * 6452

Target Data Type: * - Dataset

Namespace (blank for default): *

Other parameters: ☒

CSV separator: * TAB

Include parent container names: ☐

Include namespace: ☒

Include tags: ☐

[View Script](#) [Cancel](#) [Run Script](#)

Remove Key-Value pairs

Deletes key-value pairs of the selected objects.

Check the guide for more information on parameters and errors:
omero-guides.readthedocs.io/en/latest/scripts/docs/annotation_scripts.html

Default namespace: openmicroscopy.org/omero/client/mapAnnotation
Authors: Christian Evenhuis, MIF, Tom Boissonnet
Contact: <https://forum.image.sc/tag/omero>
Version: 2.0.0

Data Type: * Project

IDs: * 6452

Target Data Type: * - Dataset

Namespace (blank for default): *

I understand what I am doing and that this will result in a batch deletion of key-value pairs from the server: ☒

[View Script](#) [Cancel](#) [Run Script](#)

Fig. 10: The two scripts used one after another. Export the annotations for backup before removing them from the server with the Remove KV script.

We can now make the necessary edits in the CSV file and reupload the Key-Value pairs!.

Make sure to check the extended guide to learn about what else you can do with those scripts.

Authors

Tom Boissonnet

Version

1.0 of 2024/03/01

Parameter description

Data Type

In combinaison with the parameter “*IDs*”, this selects the object from which targets are selected (targets of the given type).

For example, if “*Data Type*” is set to **Project**, and “*ID*” to 789, it will select the **Datasets** inside if “*Target Data Type*” is set to **Dataset**.

The “*Data Type*” can also be set to **Tag**, in which case the tagged objects of the “*Target Data Type*” are selected. This can be used for a more flexible selection of target object to process.

IDs

IDs of the object for the “*Data Type*” selection.

Multiple ID separated with a comma can be provided at the same time. In this case, the objects are processed sequentially (if the script fails at the 3rd object, the 1st and 2nd will already be processed).

Target Data Type

The data type to be selected for processing by the script. Default to “<selected>” means that the objects specified by “*Data Type*” and “*IDs*” are the ones to be processed. Otherwise, “*Target Data Type*” must correspond to a child object of the selected type (Image is child of Dataset and Project, Well is child of Screen and Plate, ...). The option “<all (from selected)>” is used to apply the script on the selected object and its children (Datasets and Images in the case of a Project selected).

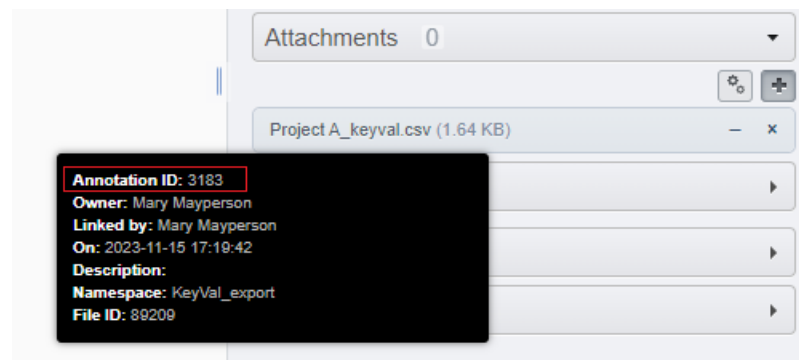
File Annotation

Parameter for the import from a CSV file. Provided from the “*Choose File*” selection menu, the file will be temporarily uploaded to OMERO for the duration of the script.

The items listed in the CSV are matched to the found target objects. If an item of the CSV correspond to no target object, the item is skipped (as such, target objects coming from different parent objects can be annotated from a single CSV).

The matching of items in the CSV to target objects is done either via IDs or names (IDs have priority over names, see “*Target ID colname*” and “*Target name colname*”).

IDs of File Annotation already attached to the object can be used instead of “*Choose File*”:



Namespace

In the case of Key-Value pairs import, the Namespace assigned to the created Key-Value pairs.

For the three other scripts, used to restrict the selection of Key-Value pairs to that Namespace.

When no Namespace is provided, matches the Namespace `openmicroscopy.org/omero/client/mapAnnotation`, which makes the Key-Value pairs editable in OMERO.web (the only Namespace that allows that).

Multiple Namespaces can be provided as a comma separated list.

All Namespaces can be selected using the `*` character.

Old Namespace

For the Namespace conversion script. List of old Namespaces (comma separated) to convert to a different Namespace (unique). This can be used to group Key-Value pairs into a single annotation (all found Key-Value pairs will be grouped to a single `MapAnnotation` (= one block of Key-Value pairs)).

New Namespace

For the Namespace conversion script. The new Namespace to assign to the Key-Value pairs found on the target object having the “*Old Namespace*”.

Create new and merge

For the Namespace conversion script. When checked, the script will create a single new annotation regrouping all the old annotation it found, instead of only updating the Namespace of each annotations. The same parameter can be given to “*Old Namespace*” and “*New Namespace*” just to regroup the annotations.

CSV separator

For the import script, the separator used in the CSV to separate elements. By default set to ‘guess’ and will attempt to detect automatically the separator among `,` `;` `TAB` `|`.

For the export script, the separator used in the generated CSV to separate elements. We recommend the use of `TAB`.

Include parent container names

For the export script. Check the box if you want to include in the CSV output for each objects the name of their parent containers.

The possible parents are `PROJECT`, `DATASET`, `SCREEN`, `PLATE`, `PLATEACQUISITION`, `RUN`, `WELL`, depending on which object type is exported. These columns are ignored by default by the import script (see “*Columns to exclude*”)

Include Namespace

For the export script. Check the box if you want to include in the CSV the Namespace associated to each Key as the first row of the document. (this output is understood by the import script).

Include Tags

For the export script. Check the box if you want to include in the CSV a column for the Tags attached to the target objects. The Tags are in the TAG[TAGSET] format if the Tags are part of a Tagset.

Split Values on

For the import script. The characters used to split the cells in the CSV into multiple entries. The character used here must be different from the chosen CSV separator character.

Columns to exclude

For the import script. If the columns specified here are found in the CSV, they will be ignored by the script.

Three default special Values are given. <ID> corresponds to the name of the column for the object IDs specified by the parameter *Target ID colname*. <NAME> corresponds to the name of the column for the object IDs specified by the parameter *Target name colname*. <PARENT> corresponds to all possible parent containers type exported when using the parameter *Include parent container names*.

Target ID colname

For the import script. The name of the column in the CSV containing the objects IDs. If not present, the script will attempt to match the target objects by name. Defaults to OBJECT_ID as used by the export script.

Target name colname

For the import script. The name of the column in the CSV containing the objects names. The names are used only to identify target objects if the ID column is not present. Defaults to OBJECT_NAME as used by the export script.

Exclude empty Values

For the import script. Check this box if you wish to avoid creating an entry in the Key-Value pairs when a cell in the CSV is empty.

Attach CSV file

For the import script. Check this box if you wish to attach the chosen CSV file to the object used for the selection of targets.

Import Tags

For the import script. Disable if you wish to ignore the Tag columns during the import of annotation.

Only use personal Tags

For the Key-Value pair import script. When Tags are specified in the CSV (under a column named TAG), check this box if you want to restrict the use of Tags to Tags owned by you.

Allow Tag creation

For the Key-Value pair import script. When Tags are specified in the CSV (under a column named TAG), check this box if you want to allow the creation of Tags when they don't exist. This also applies to Tagsets.

Authors

Tom Boissonnet

Version

1.1 of 2025/11/20

Extended description

Selecting High-Content-Screening

Selecting HCS data follows the same selection parent/child logic as for Projects/Datasets/Images, but with the HCS containers: Screen/Plate/Well/Run/Image

One specificity though is about the **Run** (Acquisition). **Runs** and **Wells** are both children objects of a **Plate**, and **Images** are attached to both a **Run** and a **Well**.

- Selecting **Images** from a **Well** will select all the **Images** inside this **Well**.
- Selecting **Images** from a **Run** will select all the **Images** coming from that **Run** (corresponds to a subset of **Images** from all combined **Well**).
- Selecting **Images** from a **Screen** or a **Plate** will follow the **Well** route selecting all **Images** inside all **Wells**.

Features of KeyVal_from_csv

- Generation of Key-Value pairs
- Automatic Generation of multiple Key-Value pairs with the same Key if multiple Values are separated by a defined character
- Assignment of Namespaces from the CSV file (or one for all Key-Value pairs from the script parameter).
- Automatic exclusion of Key-Value pairs with empty Values
- **Generation of Tags**
 - Supports Tag Sets
 - Supports selection of Tags by Tag IDs
 - Creation of new Tags and Tag Sets
 - Allows restriction to personal Tags only

Exclude empty Values from import

If you utilise a CSV with empty Values like

name	id	manual analysis	observation
A.tif	120	Hit	
B.jpg	121		
C.jpg	122		missing

by default, the empty cells are skipped during Key-Value pairs creation. If you wish to create a Key for empty cells, you can uncheck the box “*Exclude empty values*”

This might be useful if you plan to fill it in manually later on. If you used an empty Key-Value pair as a title of a “Key-Value pair section”, consider instead to assign a namespace to your Key-Value pair subsets.

Multiple Values per Key

If you utilise the Advanced parameter “*Split value on*” you can specify one character for which multiple Values get split. E.g. setting this to “!” in the following example, it would lead to the creation of two Key-Value pairs `key_1 : value_1` and `key_1 : value_2` for the Image *A.jpg*. Despite looking like a list of multiple Values for Image *B.tif* it would generate one Key-Value pair `key_1 : value_1, value_2` as the comma is not used in our example as a separator for multiple Key-Value pairs.

OBJECT_NAME	OBJECT_ID	key_1
A.tif	120	value_1!value_2
B.jpg	121	value_1, value_2
C.jpg	122	value_1!

Be aware, that for the Image *C.jpg* two Key-Value pairs `key_1 : value_1` and `key_1 :` would be created (if the “*Exclude empty values*” parameter is unchecked).

Namespaces

The default Namespace for OMERO Key-Value pairs generated in the web-interface and by our script is `openmicroscopy.org/omero/client/mapAnnotation`. Only Key-Value pairs with this Namespace can be edited in the web-interface later!

Leaving the “*Namespace*” parameter blank will assign the default Namespace to the new Key-Value pairs created, or Additionally, you can choose to use custom Namespaces, e.g. to utilize the [OMERO-Mapr](#) functionalities, for each different Key. To do so just add a first row in the CSV with the first column containing “*namespace*”. The corresponding Namespace for each Key is then put above the respective Key. If no Namespace is given, the script will fall back to the default Namespace.

namespace OBJECT_NAME	OBJECT_ID	Custom_NS key_1	REMBI_Biosample Organism
A.tif	120	value_1	Drosophila melanogaster
B.jpg	121	value_2	Mus musculus
C.jpg	122	value_1	Saccharomyces cerevisiae

Tags

Specified in dedicated columns in the CSV, you can also annotate your objects with Tags by batch. To do this simply put “*Tag*” where you would normally put the Key name. Multiple Tags can be specified as a comma separated list (or by the input parameter “*Split value on*” if given).

If you want to select a Tag in a specific Tag-Set you just add the Tag-Set in square brackets directly following the Tag. You can also specify the Tag by its Tag-Id in square brackets directly following the Tag. By default the script will not create new Tags to avoid “Tag bloat”, if you want new Tags created according to the CSV Values you have to check the “*Create new Tags*” checkbox under “*Advanced parameters*”.

Furthermore, by default the script will search through all available Tags of the group, if you want to use only your own Tags you have to check the “*Use only personal Tags*” checkbox under “*Advanced parameters*”.

OBJECT_NAME	OBJECT_ID	tag
A.tif	120	Tag_1, Tag_2, Tag_3
B.jpg	121	Tag_1[Tag-Set]
C.jpg	122	Tag_1[Tag-Id]

Choosing the CSV separator

When importing annotations from a CSV file, the script tries by default to detect the CSV separator automatically (one of `,` `;` `TAB`).

It is possible to specify directly which one is used (in the case the automatic detection fails for example). As the text in the annotations may contain commas or semi-column, it is recommended to use `TAB` as separators.

Columns of parent names

To clarify the provenance of objects listed in the CSV, you can choose to export additional columns for the name of the parent objects.

Those columns are by default excluded from the Key-Value pairs using the import script (<PARENT> Value of the “Columns to exclude” parameter matches all parent containers: PROJECT, DATASET, SCREEN, PLATE, WELL and RUN)

Default Namespace

Leaving the Namespace parameter to blank is the same as referring to the “Client namespace” (openmicroscopy.org/omero/client/mapAnnotation). The “Client namespace” is used when a new Key-Value pair is created in OMERO.web user interface. Only Key-Value pairs with this “Client namespace” are editable in OMERO.web user interface.

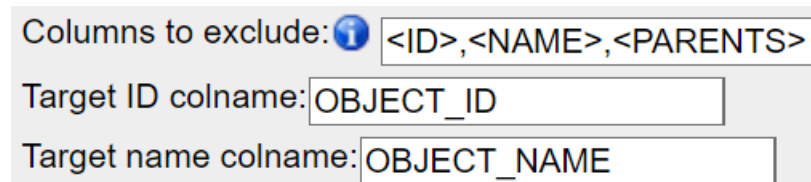
Target ID, name and excluding column from Key-Value pairs

The name of the columns for the exported objects IDs and names are always the same: OBJECT_ID and OBJECT_NAME.

This will be used by default by our import script, so we recommend to use it. If however you wished to keep your own column naming for the IDs and name of objects, you may specify those in the “*Target ID colname*” and “*Target name colname*” parameters of the import script.

While OBJECT_NAME are only used to identify the objects when OBJECT_ID is not there, it remains important to have it inside the CSV to recognize the objects more easily inside a spreadsheet editors. A good use case for the OBJECT_NAME column is to transfer annotations across groups (the IDs of the objects will differ between groups).

Note also that those two columns are excluded by defaults from the Key-Value pairs, by the use of the following three parameters:



Columns to exclude: ⓘ <ID>,<NAME>,<PARENTS>

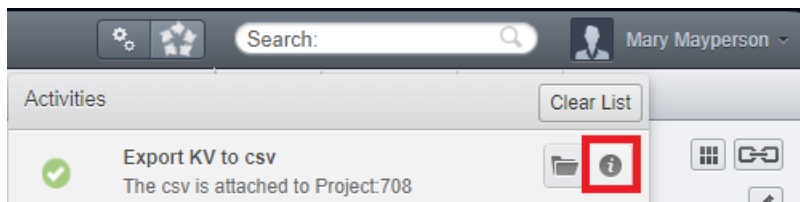
Target ID colname: OBJECT_ID

Target name colname: OBJECT_NAME

- Target ID colname: the name of the column in the CSV that contains the objects IDs
- Target name colname: the name of the column in the CSV that contains the objects names
- Columns to exclude: <ID> will exclude the column containing the objects IDs, <NAME> will do the same for the objects names, and additional columns can be excluded by indicating their name (e.g. to exclude parent objects column name when used with the export script).

Looking at the output log

When the execution of the script is over (also when it fails), you will be able to look at the output of the script by clicking that button highlighted in red in the picture below.



This output will help you understand what has been done/changed, and may help you understand things when they don't work out the way you expected them.

Authors

Tom Boissonnet Jens Wendt

Version

1.0 of 2024/03/01

Errors explained

<Something> is not a valid target for <SomethingElse>

Objects are selected from a parent object TO a child object (so for example, you cannot select a Project from a Dataset object). You could also get this error if you tried to select an HCS object (Screen, Plate, Well, Run) from a Project or Dataset (or the other way around).

Annotation <XYZ> not found

Try reloading the page. This error can happen when you try to use the script on an object of a group after opening a second tab for a different group.

Cannot split cells with a character used as CSV separator

Multiple Values per cell can be used, but the character to split them must be different from the separator used by the CSV.

CSV rows lenght mismatch: Header has <X> items, while line <line number> has <Y>

This error shows either that one of the CSV row has a different number of item from the header, or that something wrong happened with the parsing of the CSV. (make sure that your CSV separator is not a comma and that you use also a comma to have multiple Value per cells, like in the case of Tags).

Error while reading the CSV, convert your file to utf-8 encoding'utf-8'

The CSV must be in a UTF-8 encoding. You can convert it with a text editor (e.g. Notepad++ for Windows, TextEdit for Mac). We recommend to start from a CSV generated by the “Export to csv” script.

Failed to sniff CSV delimiter

It happens that the sniffer to detect fails to understand what separator was used inside you CSV file. Try instead providing the separator used inside your CSV as parameter of the script (instead of guessing).

File annotation ID must be given when using Tag as source

Because files cannot be attached to Tags, and thus cannot be searched automatically when providing Tags as source, you must provide a file as input to the script.

Neither the column for the objects name or the objects index were found

Make sure that the column in your CSV match the columns given as parameter of the import script. Both are optional, but at least one is required.

No CSV FileAnnotation was found on <OBJ_TYPE>:<OBJ_ID>

When providing no file from the select File button, nor giving a file annotation to read Key-Value pairs from, the script attempt to search specifically for a CSV file attached to the current parent object. If none is found, you obtain this error. Try to provide a CSV file instead of letting the script guessing.

Number of Source IDs and FileAnnotation IDs must match

When providing more than one parent object, the Key-Value pairs can be read from either a single CSV file, or by multiple files. In the later scenario, there must be an identical number of FileAnnotation and parent objects, in order for the script to understand which corresponds to which.

Please confirm that you understood the risks of a batch deletion

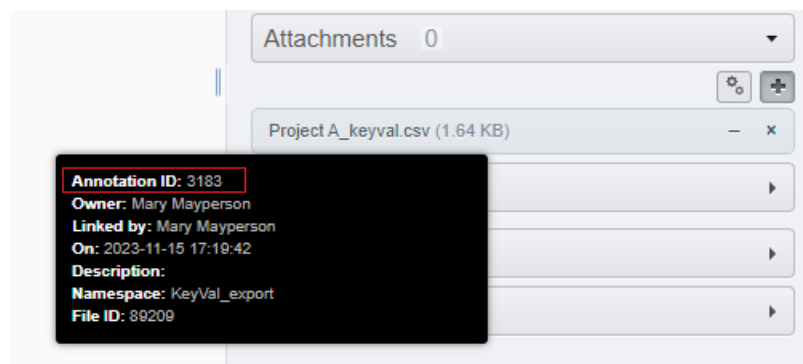
It seems that you went a bit too fast and forgot to confirm that you understood the risk associated with a batch deletion of annotations.

Target objects identified by name have duplicate

Because no column specifying the ID of the object to annotate was found, the script attempted to uniquely identify the objects by their name. It seemed however that in the list of all objects founds from the parent you gave, multiple objects have the same name. Please try again by adding a column with the IDs of the objects to annoate. (OBJECT_ID is used by default).

The provided annotation ID must reference a FileAnnotation

The ID of the file annotation seems incorrect. Make sure that you gave the FileAnnotation ID and not the LinkAnnotation ID, as indicated in the image bellow.



The Tag ID:{TAG_ID} is not in the permitted selection of tags

This error either means that the given ID for the Tag is wrong, or that the ID corresponds to a Tag in a different group or belonging to a different user (and option 'Use only personal tags' is unchecked). Another possibility is that you have a Tagset with a name made of numbers only, and thus is interpreted as a Tag ID.

The Tag {TAG_NAME} doesn't correspond to the Tag on the server with ID:{TAG_ID}

This error tells you that the Tag found from the given ID has a name different from the one you used in the CSV. This is to prevent you from using the wrong Tag.

The Tag '{TAG_NAME}' does not exist while creation of new Tags is not permitted

If your Tags do not exist yet (or if you only use your own Tag), remember to check the box 'Create new tags' if you wish to generate the Tags automatically.

The Tag '{TAG_NAME}' in Tagset '{TAG_SET}' does not exist while creation of new Tags is not permitted

Same as the error above, but in the case a Tagset is specified.

The CSV contains duplicates {DUPL} which makes it impossible to correctly allocate the annotations

When no ID is used to identify the objects to annotate, names are used instead. In that case, there can not be two objects with the same name in the CSV (and in the selected object on OMERO), in which case it is impossible to identify which object should be annotated.

Authors

Tom Boissonnet

Version

1.0 of 2024/03/01

Contribute

Changes to the documentation or the materials should be made directly in the [omero-guide-scripts](#) repository.

3.2 External Software and OMERO

Here we describe how to use third party tools to analyze data stored in OMERO. We provide installation instructions, step-by-step workflow either to use the User Interface or the API of those tools.

*CellProfiler*

CellProfiler is a free open-source software for quantitative analysis of biological images. We demonstrate how to integrate CellProfiler and OMERO using the CellProfiler Python API and the OMERO Python API.

For more details, visit the [CellProfiler website](#).

Follow the instructions in the `README.md` file to run the notebooks either locally or on [mybinder.org](#).

*Fiji*

Fiji is a popular free open-source image processing package based on ImageJ, <https://imagej.net/Fiji>. We demonstrate how to use Fiji User Interface and OMERO and we also demonstrate how to analyze data in OMERO using Fiji scripting facility.

*ilastik*

ilastik is a free open-source interactive learning and segmentation toolkit, with which can be used to leverage machine learning algorithms to easily segment, classify, track and count cells or other experimental data. For more details go to see <https://www.ilastik.org/>. We will show how to analyze data stored in OMERO, using the ilastik user interface, Fiji and the OMERO ImageJ plugin. We will then also show how to analyze images using the ilastik API and OMERO.py API.

*Orbit*

Orbit is a free open-source software with the focus on quantification of big images like whole slide scans. It offers sophisticated image analysis algorithms. Of those, tissue quantification using machine learning techniques, object/cell segmentation, and object classification are the basic ones. We demonstrate how to integrate Orbit and OMERO using both the User Interface and the API. For more details, go to <http://www.orbit.bio/>



3.2.1 CellProfiler

CellProfiler is a free open-source software for quantitative analysis of biological images. We demonstrate how to integrate CellProfiler and OMERO using the CellProfiler Python API and the OMERO Python API.

For more details, visit the [CellProfiler website](#).

Follow the instructions in the `README.md` file to run the notebooks either locally or on [mybinder.org](#).

Contents

Install CellProfiler and OMERO Python bindings

In this section, we show how to install CellProfiler in a [Conda](#) environment. We will use the CellProfiler API to analyze data stored in an OMERO server.

CellProfiler currently runs on Python 2.7. It does not yet support Python 3.

Setup

We recommend to install the dependencies using Conda. Conda manages programming environments in a manner similar to [virtualenv](#). You can install the various dependencies following the steps below (Option 1) or build locally a Docker Image using [repo2docker](#) (Option 2). When the installation is done, you should be ready to use the CellProfiler API and OMERO, see [Getting started with CellProfiler and OMERO](#).

The installation below is needed to run the scripts and/or notebooks. If you wish to start your own environment without the scripts/notebooks, copy locally into an `environment.yml` file the content of [binder/environment.yml](#), remove or add the dependencies you need and run the commands below to create a conda environment.

Option 1

- Install [Miniconda](#) if necessary.
- If you do not have a local copy of the [omero-guide-cellprofiler repository](#), first clone the repository:

```
$ git clone https://github.com/ome/omero-guide-cellprofiler.git
```

- Go into the directory:

```
$ cd omero-guide-cellprofiler
```

- Create a programming environment using Conda:

```
$ conda create -n cellprofiler python=2.7
```

- Install CellProfiler, its dependencies and `omero-py` in order to connect to an OMERO server using an installation file:

```
$ conda env update -n cellprofiler --file binder/environment.yml
```

- Activate the environment:

```
$ conda activate cellprofiler
```

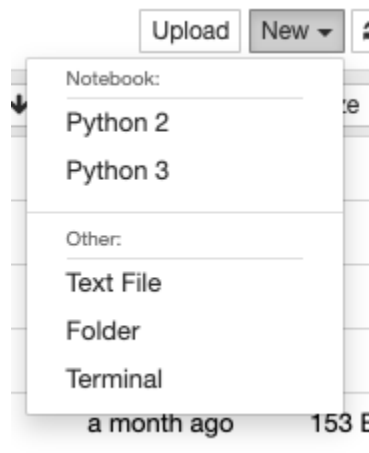
Option 2

Alternatively you can create a local Docker Image using `repo2docker`, see `README.md`:

```
$ repo2docker .
```

When the Image is ready:

- Copy the URL displayed in the terminal in your favorite browser
- Click the New button on the right-hand side of the window
- Select Terminal



- A Terminal will open in a new Tab
- A Conda environment has already been created when the Docker Image was built
- To list all the Conda environment, run:

```
$ conda env list
```

- The environment with CellProfiler and the OMERO Python bindings is named `kernel`, activate it:

```
$ conda activate kernel
```

Getting started with CellProfiler and OMERO

Description

We will use a Python script showing how to analyze data stored in an OMERO server using the CellProfiler API.

We will show:

- How to connect to server.
- How load images from a Plate using the OMERO API.
- How to run CellProfiler using its Python API.

- How to save the generated results and link them to the Plate.

Resources

We will use a CellProfiler example pipeline to analyse RNAi screening data from the Image Data Resource (IDR).

- Example pipeline from the CellProfiler website: [Cell/particle counting and scoring the percentage of stained objects](#).
- Images from IDR [idr0002](#).

For convenience, the IDR data have been imported into the training OMERO.server. This is only because we cannot save results back to IDR which is a read-only OMERO.server.

Setup

We recommend to use a Conda environment to install CellProfiler and the OMERO Python bindings. Please read first *Install CellProfiler and OMERO Python bindings*.

Step-by-Step

In this section, we go over the various steps required to analyse the data. The script used in this document is `idr0002_save.py`.

When running CellProfiler **headless**, it is important to set the following:

```
import cellprofiler_core.preferences as cpprefs
# Important to set when running headless
cpprefs.set_headless() # noqa
```

Connect to the server:

```
def connect(hostname, username, password):
    conn = BlitzGateway(username, password,
                        host=hostname, secure=True)
    conn.connect()
    return conn
```

Load the plate:

```
def load_plate(conn, plate_id):
    return conn.getObject("Plate", plate_id)
```

A CellProfiler pipeline usually expects the files to be analyzed to be available locally. This is not the case here. So we first need to remove some modules from the pipeline so we can then inject data retrieved from the OMERO server:

```
def load_pipeline(pipeline_path):
    pipeline = cpp.Pipeline()
    pipeline.load(pipeline_path)
```

(continues on next page)

(continued from previous page)

```

# Remove first 4 modules: Images, Metadata, NamesAndTypes, Groups...
# (replaced by InjectImage module below)
for i in range(4):
    print('Remove module: ', pipeline.modules()[0].module_name)
    pipeline.remove_module(1)
print('Pipeline modules:')
for module in pipeline.modules():
    print(module.module_num, module.module_name)
return pipeline

```

We are now ready to analyze the plate:

```

def analyze(plate, pipeline):
    warnings.filterwarnings('ignore')
    print("analyzing...")
    # Set Cell Output Directory
    new_output_directory = os.path.normcase(tempfile.mkdtemp())
    cpprefs.set_default_output_directory(new_output_directory)

    files = list()
    wells = list(plate.listChildren())
    wells = wells[0:5] # use the first 5 wells
    for count, well in enumerate(wells):
        # Load a single Image per Well
        image = well.getImage(0)
        print(image.getName())
        pixels = image.getPrimaryPixels()
        size_c = image.getSizeC()
        # For each Image in OMERO, we copy pipeline and inject image modules
        pipeline_copy = pipeline.copy()
        # Inject image for each Channel (pipeline only handles 2 channels)
        for c in range(0, size_c):
            plane = pixels.getPlane(0, c, 0)
            image_name = image.getName()
            # Name of the channel expected in the pipeline
            if c == 0:
                image_name = 'OrigBlue'
            if c == 1:
                image_name = 'OrigGreen'
            inject_image_module = InjectImage(image_name, plane)
            inject_image_module.set_module_num(1)
            pipeline_copy.add_module(inject_image_module)
        pipeline_copy.run()

        # Results obtained as CSV from Cell Profiler
        path = new_output_directory + '/Nuclei.csv'
        files.append(path)
    print("analysis done")
    return files

```

(continues on next page)

(continued from previous page)

Let's now save the generated CSV files and link them to the plate:

```
def save_results(conn, files, plate):
    # Upload the CSV files
    print("saving results...")
    namespace = "cellprofiler.demo.namespace"
    for f in files:
        ann = conn.createFileAnnfromLocalFile(f, mimetype="text/csv",
                                              ns=namespace, desc=None)
        plate.linkAnnotation(ann)
```

When done, close the session:

```
def disconnect(conn):
    conn.close()
```

In order to use the methods implemented above in a proper standalone script: **Wrap it all up** in an `analyze` method and call it from `main`:

```
def main():
    # Collect user credentials
    host = input("Host [wss://workshop.openmicroscopy.org/omero-ws]: ") or 'wss://
↳workshop.openmicroscopy.org/omero-ws'
    username = input("Username [trainer-1]: ") or 'trainer-1'
    password = getpass("Password: ")
    plate_id = input("Plate ID [102]: ") or '102'
    # Connect to the server
    conn = connect(host, username, password)

    # Read the pipeline
    pipeline_path = "../notebooks/pipelines/ExamplePercentPositive.cppipe"
    pipeline = load_pipeline(pipeline_path)

    # Load the plate
    plate = load_plate(conn, plate_id)

    files = analyze(plate, pipeline)

    save_results(conn, files, plate)
    disconnect(conn)
    print("done")

if __name__ == "__main__":
    main()
```

Exercises

1. Modify the script above to analyze images in a dataset (Solution).
2. Modify the script to link the generated results to the corresponding image (Solution).
3. Modify the script to aggregate the result in an OMERO.table and link the output to the plate (Solution).

Analyze OMERO data using a Jupyter Notebook

Description

We will demonstrate how to integrate CellProfiler and OMERO using the CellProfiler Python API and the OMERO Python API. We will use a Jupyter notebook to demonstrate the integration.

We will show:

- How to adjust an existing CellProfiler pipeline so that it can be used with OMERO.
- How load images from a Plate using the OMERO API.
- How to run CellProfiler using its Python API.
- How to plot the results.
- How to save the generated results back to OMERO as OMERO.table so they can be used later on by OMERO.parade.

Resources

We will use a CellProfiler example pipeline to analyse RNAi screening data from the Image Data Resource (IDR).

- Example pipeline from the CellProfiler website: [Cell/particle counting and scoring the percentage of stained objects](#).
- Images from IDR [idr0002](#).
- Notebook `idr0002_save.ipynb`.

For convenience, the IDR data have been imported into the training OMERO.server. This is only because we cannot save results back to IDR which is a read-only OMERO.server.

Step-by-Step

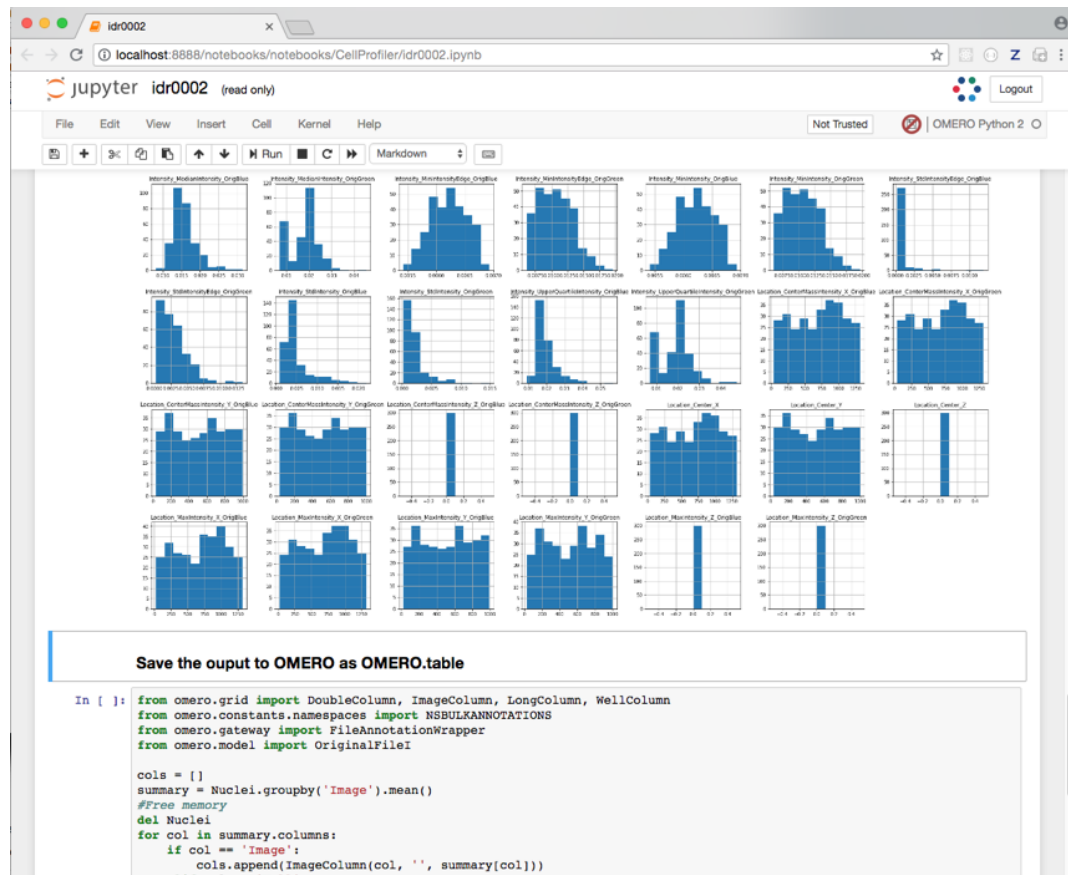
The pipeline is run on all 2-Channel images from each Well in the 96-well plate (each Well contains one image), generating a CSV file containing rows for different objects identified in the image and columns for various parameters measured.

1. First, open the webclient and find the Plate belonging to trainer-1 named plate1_1_013.
2. Launch the `idr0002_save.ipynb` notebook in [mybinder.org](#).
3. Select the first Step and click on the Run button to execute each step in turn.
4. For the connection to OMERO, you will be asked to enter your login details when running the OMERO credentials cell.
5. Select the plate in the webclient, find the Plate ID in the right-hand panel and copy this into the `plate_id` variable in the next step of the notebook.

6. The following cell loads the example pipeline and modifies it to remove the modules that are normally used for loading images from disk.
7. These modules are replaced by the InjectImage module, using numpy planes loaded from OMERO Images. This allows to pass data from OMERO to CellProfiler.
8. Note that to save time, we run on a subset of all Wells in the plate. We run it on the first 5 wells

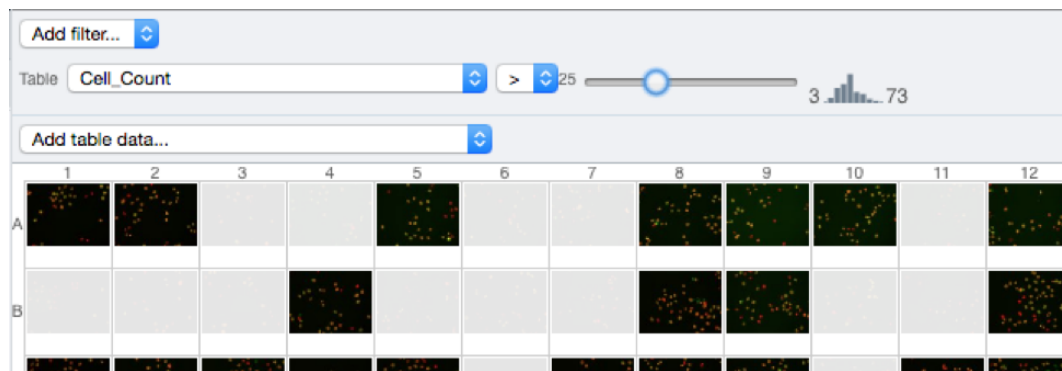
```
# Create list from generator
wells = list(plate.listChildren())
wells = wells[0:5]
well_count = len(wells)
```

9. The generated CSV file is read into a Dataframe for each image. We add the Image ID and Well ID, as well as the total number of Objects, Cell_Count, to each Dataframe.
10. All the Dataframes are then concatenated into a single Dataframe.
11. We visualize the data as histograms for each column with `df.hist()`.



12. Finally, the Dataframe rows are grouped by Image to give an average value per Image of each parameter (column) in the table.
13. This data is saved back to OMERO as an HDF5-based table attached to the Plate, which can be read by other clients.
14. Return to the webclient and select the Plate named plate1_1_013_previously_analysed.

15. Select a Well in the central pane and open the Tables harmonica in the General tab in the right-hand pane. This will show all the CellProfiler values for this Well.
16. In the Thumbnails dropdown menu at the top-right of the centre panel, select the Parade plugin.
17. At the top-left of the centre panel choose *Add filter...* > *Table* to filter Wells by the data from CellProfiler.
18. Change the filter from ImageNumber to Cell_Count (at the bottom of the list).
19. Now you can use the slider to filter Wells by Cell Count.



Contribute

Changes to the documentation or the materials should be made directly in the [omero-guide-cellprofiler repository](#).

3.2.2 Fiji

Fiji is a popular free open-source image processing package based on ImageJ, <https://imagej.net/Fiji>. We demonstrate how to use Fiji User Interface and OMERO and we also demonstrate how to analyze data in OMERO using Fiji scripting facility.

Install the plugins

How to install OMERO plugins for Fiji/ImageJ

Description

[Fiji](#) is a free open-source image processing package based on ImageJ. The following workflow shows how to install the OMERO plugin for Fiji and ImageJ.

The OMERO plugin does not have an update site yet.

Setup step-by-step

We assume that you have already Fiji/ImageJ installed locally. If not, download [Fiji](#).

In this section, we will cover the steps required to install the OMERO plugin for Fiji. If you wish to install it for ImageJ, an additional step is needed.

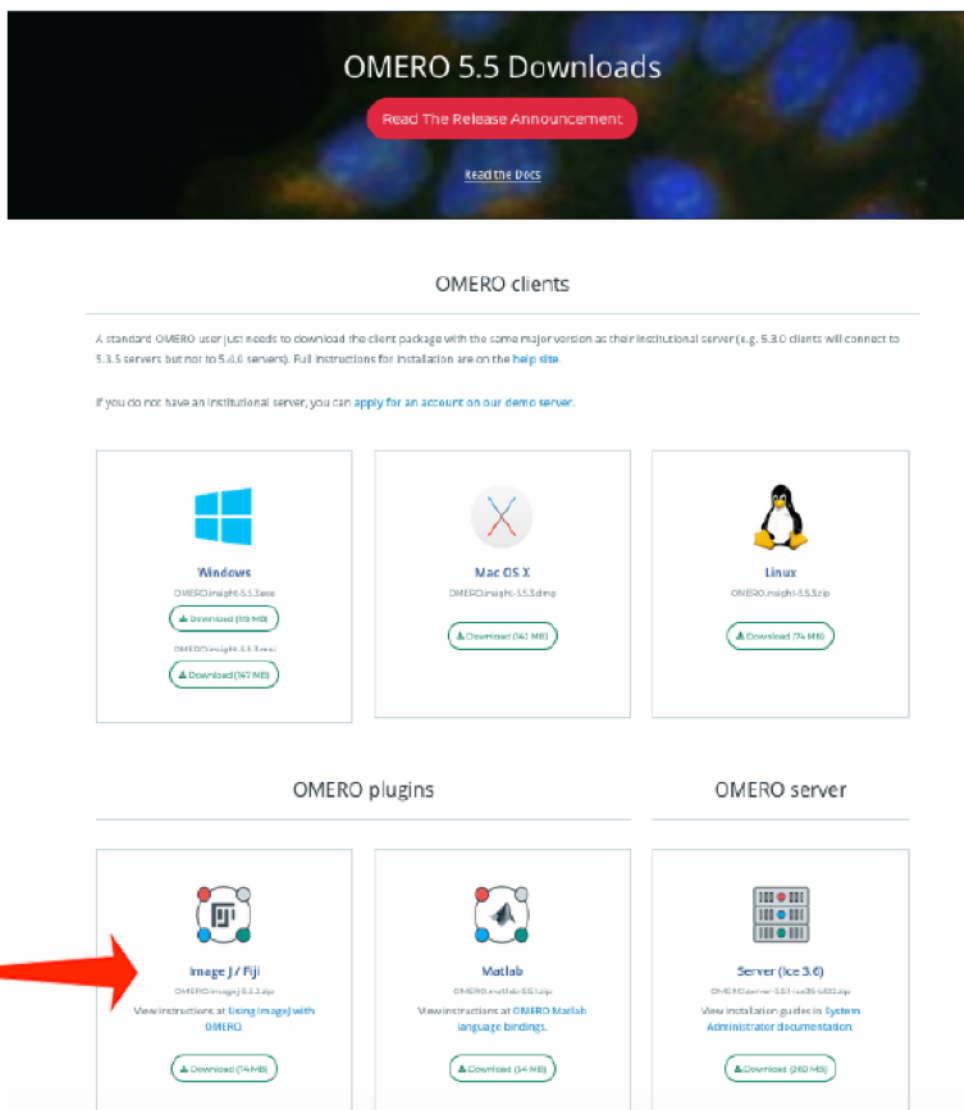
We first describe the common installation steps for ImageJ and Fiji. We then describe how to install the *Bio-Formats Package* for ImageJ.

Installing the OMERO plugin in Fiji also adds the dependencies required to connect to OMERO using the Script Editor of Fiji.

Installation of the OMERO plugin for Fiji and ImageJ

Below are the common steps that need to be followed:

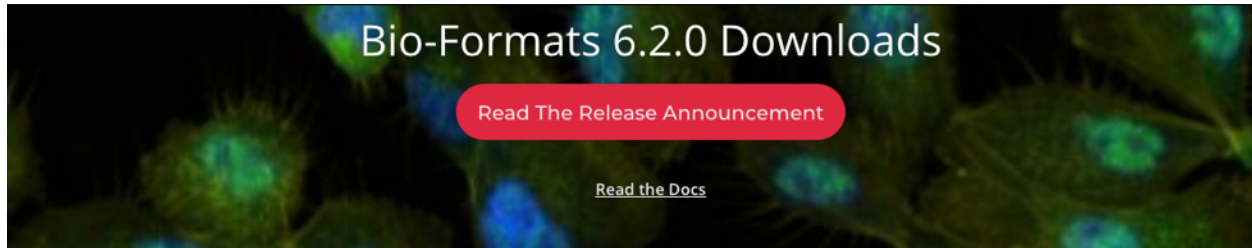
- Find the Plugins folder of your Fiji application and check if it contains any old omero_ij-5.x.x-all.jar file(s) or OMERO.imagej-5.x.x folder(s). Remove any such jar files or folders from the Plugins folder.
- Download from <https://www.openmicroscopy.org/omero/downloads> the latest 5.x.x version of ImageJ/Fiji plugin for OMERO



- For recent plugin versions (5.5.7 and higher): Note where you downloaded the `omero_ij-5.x.x-all.jar` file. Copy that file into the *plugins* folder of Fiji.
- For plugin versions lower than 5.5.7: Extract the downloaded .zip archive. Remember where you extracted it to.
- For plugin versions lower than 5.5.7: Copy the extracted folder and paste it to the *plugins* folder of Fiji.
- **Note:** For plugin versions lower than 5.5.7: Some Windows unzip apps create a double folder enclosing the plugin. If that is the case, copy the inner OMERO.imagej-5.x.x folder into *Fiji.app > plugins* folder.
- Now, restart Fiji. If you are using ImageJ, follow with the additional step below.

Installation of Bio-Formats Package, ImageJ only

- Download the latest version of the *Bio-Formats Package* from: <https://www.openmicroscopy.org/bio-formats/downloads>



Tools

 Bio-Formats Package bioformats_package.jar The complete bundle for end users containing everything needed to read images into ImageJ. Instructions for installing Bio-Formats in ImageJ are available in the user guide. Download (31.64 MB)	 Command Line Tools bfutils.zip A bundle of scripts for using Bio-Formats on the command line with bioformats_package.jar already included. For more info, see the command line tools documentation. Download (29.96 MB)	 MATLAB Toolbox bfmatlab.zip A bundle of functions for using Bio-Formats from MATLAB with bioformats_package.jar already included. For more info, see the MATLAB documentation. Download (29.97 MB)	 Octave Package bioformats-octave-6.2.0.tar.gz A bundle of functions for using Bio-Formats from Octave. You'll need to install bioformats_package.jar separately. For more info, see the Octave documentation. Download (37.44 KB)
--	---	---	---

- Move the downloaded file into the *ImageJ > plugins* folder.
- Restart ImageJ.

Installation of the plugins to interact with OMERO using the ImageJ marco language

The plugins are developed and supported by Institute of genetics, reproduction & development (iGReD) in Clermont-Ferrand (France).

- Download the latest version of *simple-omero-client* from <https://github.com/GReD-Clermont/simple-omero-client>:
 - Click on the Tags tab
 - Click on the most recent tag
 - Download the simple-omero-client-<tag>.jar file e.g. simple-omero-client-5.14.1.jar
- Download the latest version of *omero_macro-extensions* from https://github.com/GReD-Clermont/omero_macro-extensions:
 - Click on the Tags tab
 - Click on the most recent tag

- Download the `omero_macro-extensions-<tag>.jar` file e.g. `omero_macro-extensions-1.3.2.jar`
- Move the downloaded jars into the *Fiji > plugins* folder.
- Restart Fiji.

Manual Analysis

Crop and import

Description

The following workflows should work both with ImageJ and Fiji, after these have been correctly set up with the OMERO plugin for Fiji/ImageJ.

Using the User Interface of the OMERO plugin, we will show:

- How to connect to OMERO using the OMERO plugin for ImageJ/Fiji.
- How to open an image from OMERO.server into Fiji/ImageJ.
- How to import the cropped image from Fiji/ImageJ into OMERO.

Setup

- Install Fiji on the local machine with the OMERO.insight-ij plugin. The installation instructions can be found at [How to install OMERO plugins for Fiji/ImageJ](#).

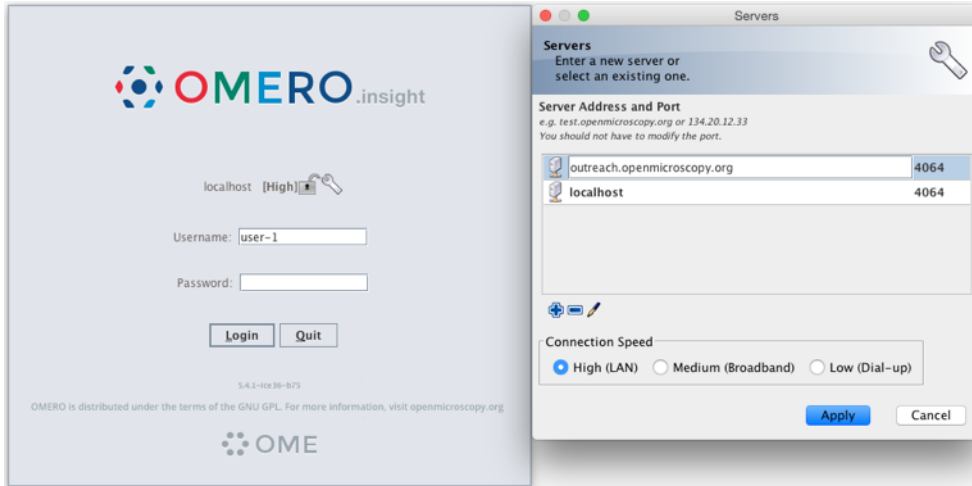
Resources

- Samples images from the Image Data Resource (IDR) [idr0021](#).

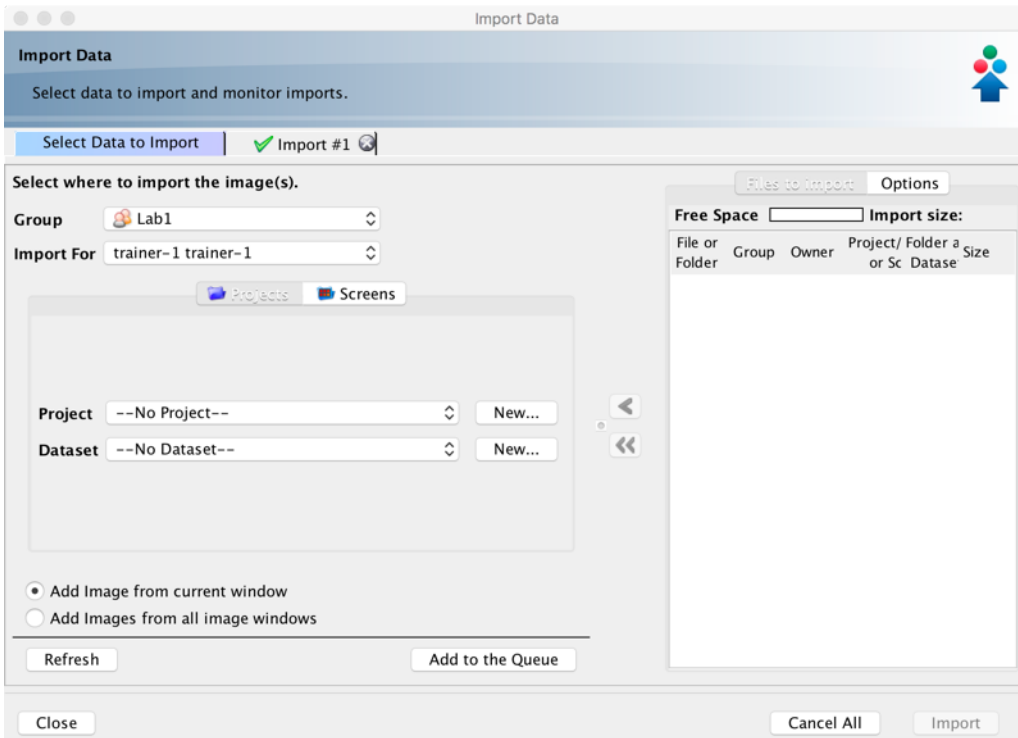
Step-by-step

In this example, we show how to open an OMERO image, crop the image and import the cropped image back to OMERO as OME-TIFF.

1. Launch Fiji/ImageJ.
2. Go to *Plugins > OMERO > Connect To OMERO*. This will show a login screen where you can enter the name of the server to connect to, the username and password. The OMERO plugin will allow you to browse your data in a similar manner to OMERO.web.
3. In the OMERO login dialog, click the wrench icon  and then add the server address in the dialog. By default, only “localhost” is listed. Click on the *plus* icon to add a new line to the list and type into the line the server address.
4. Click Apply.



5. Enter your credentials and click *Login*.
6. Select a dataset, for example the *A-Fiji-dataset* Dataset.
7. Double-click on a thumbnail or on an Image in the left-hand tree to open an Image in Fiji/ImageJ.
8. Draw a Rectangle on the Image.
9. Select the option *Image > Crop*.
10. A new Image will be displayed in a Fiji/ImageJ window.
11. Go to *Plugins > OMERO > Save Image(s) To OMERO*.
12. An Import dialog will pop up.



13. Check that the option *Add Image from current window* is selected.

14. Select where to import the cropped Image, for example an existing Dataset, e.g. *A-Fiji-dataset*. You can also select *New From Folder* option which will create a new Dataset named with the name of the image you opened from OMERO to Fiji. In case you select *No Dataset* option, the new image will be displayed in the *Orphaned Images* folder in OMERO.
15. Click *Add to the Queue* button.
16. Then click *Import*. The import will start.
17. When the import is done, go back to the Tree view in the Fiji plugin or OMERO.web. Refresh. Check the new Image.

Analyze, save ROIs and measurements

Description

The following workflows should work both with ImageJ and Fiji, after these have been correctly set up with the OMERO plugin for Fiji/ImageJ.

Using the User Interface of the OMERO plugin, we will show:

- How to connect to OMERO using the OMERO plugin for ImageJ/Fiji.
- How to open an image from OMERO.server into Fiji/ImageJ.
- How to manually save ROIs and measurements as CSV back to the original image in OMERO.server.
- How to import a newly created image from Fiji/ImageJ into OMERO.

Setup

- Install Fiji on the local machine with the OMERO.insight-ij plugin. The installation instructions can be found at [How to install OMERO plugins for Fiji/ImageJ](#).

Resources

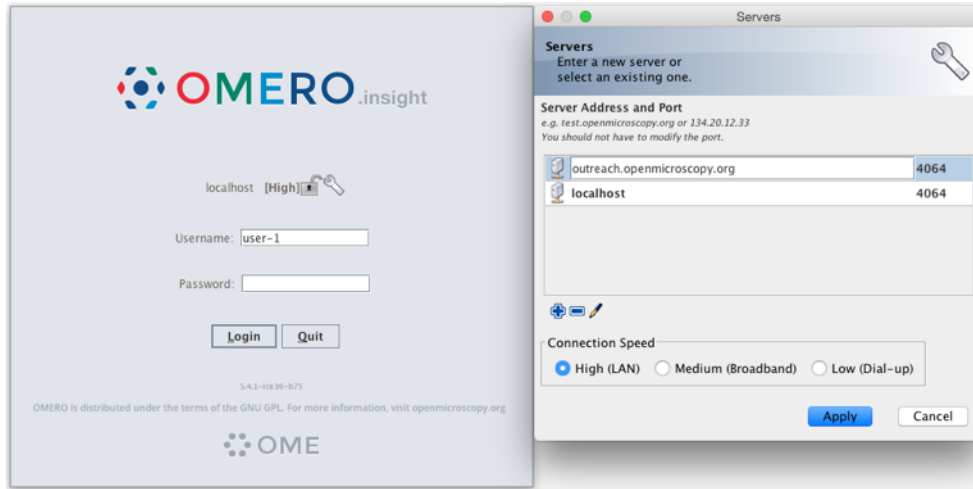
- Samples images from the Image Data Resource (IDR) [idr0021](#).

Step-by-step

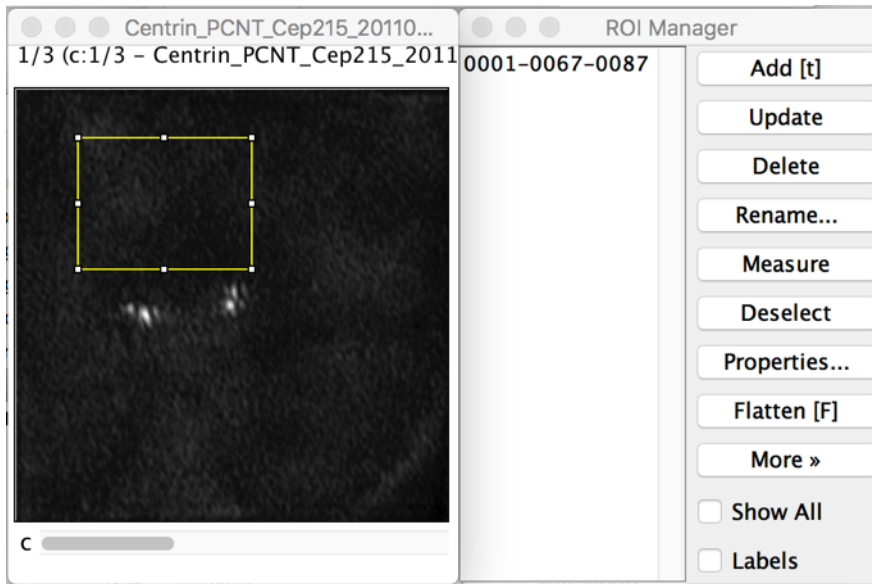
In this first example we show how to open an OMERO image in Fiji, draw ROIs, measure those ROIs and show how to save the ROIs and the measurement back to OMERO.

1. Launch Fiji/ImageJ.
2. Go to *Plugins > OMERO > Connect To OMERO*. This will show a login screen where you can enter the name of the server to connect to, the username and password. The OMERO plugin will allow you to browse your data in a similar manner to OMERO.web.
3. In the OMERO login dialog, click the wrench icon  and then add the server address in the dialog. By default, only “localhost” is listed. Click on the *plus* icon  to add a new line to the list and type into the line the server address.

4. Click Apply.

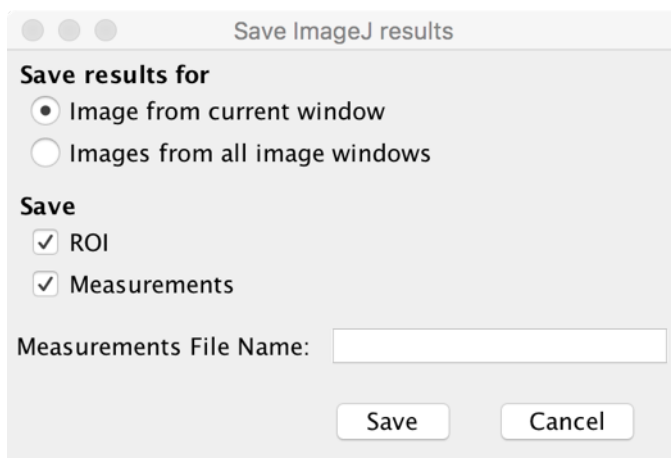


5. Enter your credentials and click *Login*.
6. Select the **A-Fiji-dataset** Dataset.
7. Double-click on a thumbnail or on an Image in the left-hand tree to open an Image in ImageJ.
8. Go to *Analyze > Tools > ROI Manager...*
9. Draw a shape using for example the Freehand selection tool.
10. In the ROI manager, click the button *Add [t]* to add the shape to the ROI Manager.



11. Move to another channel, using the *c slider*.
12. Draw other shapes if desired. Click *Add [t]* to add them to the ROI Manager.
13. When done with the drawing, click the button *Measure* in the ROI Manager.
14. A dialog with measurements for each shape pops up.
15. To save the ROI and the measurement back to OMERO, go to *Plugins > OMERO > Save ROIs To OMERO*.
16. In the dialog popping up, under the *Save* section select *ROI* and *Measurements*.

17. The measurements are saved back to OMERO as a CSV file and linked to the Image.



18. Go to OMERO.web and log in.
19. Select the image opened in Fiji/ImageJ.
20. Check that there is a new CSV file under the *Attachments* harmonica.
21. Open the image in OMERO.iviewer to see the ROIs and make sure that you can interact with them.

Manual Segmentation

Description

The following workflows should work both with ImageJ and Fiji, after these have been correctly set up with the OMERO plugin for Fiji/ImageJ.

Using the User Interface of the OMERO plugin, we will show:

- How to connect to OMERO using the OMERO plugin for Fiji/ImageJ.
- How to open an image from OMERO.server into Fiji/ImageJ.
- How to manually segment an image opened from OMERO in Fiji/ImageJ using the plugins *Auto Threshold* and *Analyze particles* in Fiji/ImageJ.
- How to record this workflow in Fiji/ImageJ, using the *Recorder* plugin, for later use.

Setup

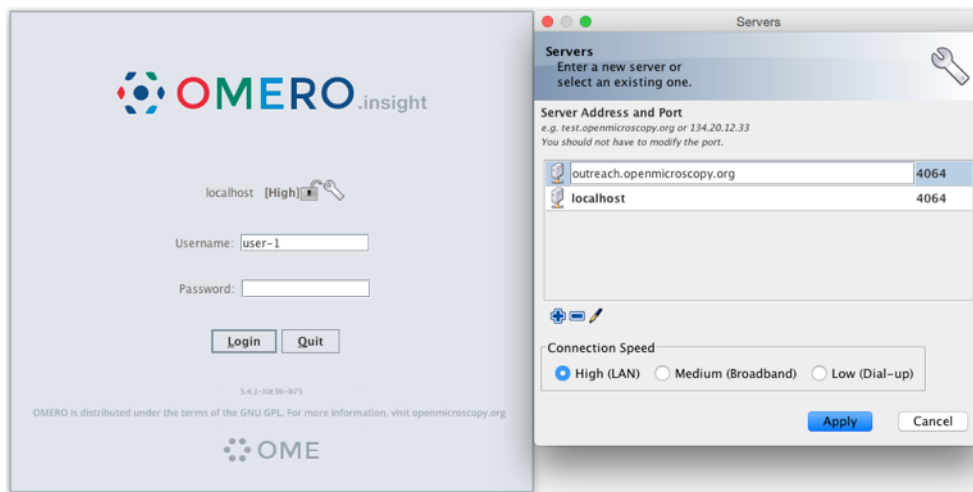
- Fiji has been installed on the local machine with the OMERO.insight-ij plugin. The installation instructions can be found at [How to install OMERO plugins for Fiji/ImageJ](#).

Resources

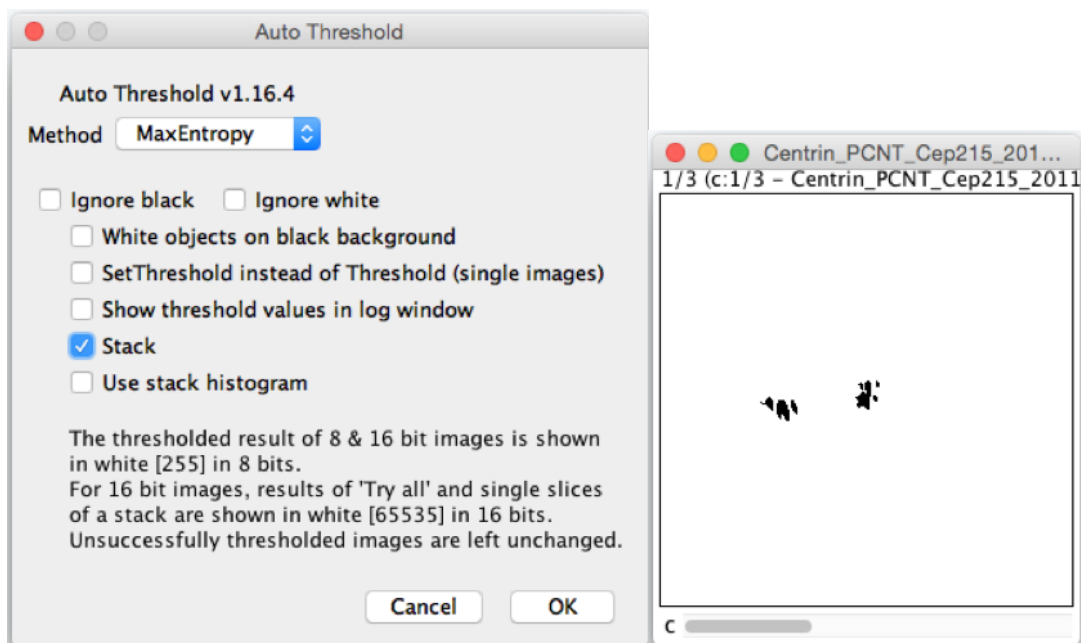
- Samples images from the Image Data Resource (IDR) [idr0021](#).

Step-by-step

1. Launch Fiji/ImageJ.
2. Go to *Plugins > OMERO > Connect To OMERO*. This will show a login screen where you can enter the name of the server to connect to, the username and password. The OMERO plugin will allow you to browse your data in a similar manner to OMERO.web.
3. In the OMERO login dialog, click the wrench icon  and then add the server address in the dialog. By default, only “localhost” is listed. Click on the *plus* icon to add a new line to the list and type into the line the server address.
4. Click Apply.



5. Enter your credentials and click *Login*.
6. Browse to the Project **idr0021**, open any Dataset and double-click once on an Image to open it in Fiji. Bio-Formats is used to view the Image.
 - Make sure to select *View stack with: Hyperstack* in the *Bio-Formats Import Options* dialog.
 - Note that each plane will be transferred from the server to the client machine so this may take a few moments.
7. To open the Recorder, go *Plugins > Macros > Record...*, select Macros to record the actions. The steps will then be used in the Scripting workflow.
8. Convert floating-point pixel-type to 8-bit using *Image > Type > 8-bit*.
9. Go to *Image > Adjust > Auto Threshold*, to open the *Auto Threshold* dialog:
 - Select *MaxEntropy* for the *Method* parameter.
 - Check the checkbox *Stack*.

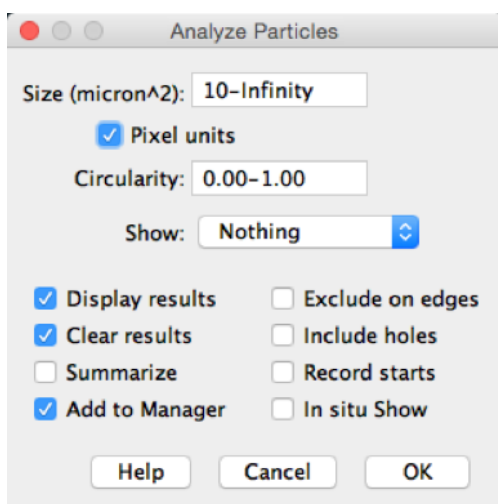


10. Click *OK*.

11. Then open *Analyze > Analyze Particles...*

12. In the dialog

- Set *Size* to *10-Infinity* and check *Pixel units*.
- Check the following checkboxes:
 - i. *Display results*
 - ii. *Clear results*
 - iii. *Add to Manager*
- Click *OK* then *Yes* in the popup dialog indicating asking to *Process all X images?*



13. To save the thresholded Image back to OMERO with the generated ROIs and the measurements:

- Select *Plugins > OMERO > Save Image(s) to OMERO*.

- Create a New Dataset for the image
 - i. Click the *New...* button next to the selection box on the Dataset row.
 - ii. In the dialog that pops up, enter a name and a description (optional).
 - iii. Click Create.
- The newly created Dataset will automatically be selected.
- Click *Add to the Queue* then *Import*.
- Go to OMERO.web and check that the measurements have been saved in a CSV file and attached to the Images. The attachment can then be downloaded at any time.

14. To save the recorded macro to OMERO.server:

- In Fiji, find the Recorder window and click “Create”. A new window will pop up with the macro you just recorded. Verify the macro on a new image. Once the verification succeeds, just close the macro window and in the dialog which pops up select “Yes” to save the macro. Save the macro locally e.g. “your-macro-name.ijm”.
- In OMERO.web, select the dataset you would like the macro to be run on and attach the “your-macro-name.ijm” macro you just recorded as file attachment to that dataset. The attachment can then be downloaded at any time. Also, the attached macro can be used by Fiji scripts - see example on [thresh-old_scripting.html](#).

Automated Analysis

Segment using ImageJ Macro language

Description

The following workflows should work in Fiji, after it has been correctly set up with the OMERO plugin for Fiji/ImageJ.

In this section we use the ImageJ macro language to access data in OMERO. To interact with OMERO using the **ImageJ macro language**, two extra plugins need to be installed.

Using the Scripting editor of Fiji, we will show:

- How to connect to OMERO..
- How to load all the images within a given Dataset.
- How to load the images in ImageJ using Bio-Formats.
- How to analyze the images using *Auto Threshold* and *Analyze particles* plugins in ImageJ within a script.
- How to save the segmented ROIs as polygons in OMERO.
- How to collect the measurement associated to the ROIs and
 - Save them as an OMERO.table and link the table to the Dataset.
 - Save them as a CSV file and link it to the Dataset.

Setup

- Install Fiji on the local machine with the OMERO.insight-ij plugin. The installation instructions can be found at [How to install OMERO plugins for Fiji/ImageJ](#).
- Install the two plugins required to use the macro language. Instructions can be found at [Installation of the plugins to interact with OMERO using the ImageJ macro language](#).

Resources

- Data: Samples images from the Image Data Resource (IDR) [idr0021](#).
- Script: ImageJ macro language script for automatic segmentation of images from OMERO using Fiji - `analyse_dataset_save_rois_and_summary_table.ijm`.
- More examples on how to use the ImageJ Macro language and OMERO can be found at [macro extension examples](#).

Step-by-Step

We will now repeat the manual analysis [Manual Segmentation](#) on a Dataset using the scripting facility available in Fiji. Let's go over the script to understand the logic and see how it matches the UI steps.

It will process all the Images in the specified Dataset, applying threshold, analyzing particles and saving ROIs back in OMERO i.e. we reproduce in a script the manual steps recorded. Further, it will create a CSV and OMERO.table to be attached to that Dataset in OMERO.

1. In your browser, go to the server address provided.
2. Log in using the credentials provided.
3. Make sure you are selecting your own data. Select the Dataset **A-Fiji-dataset**.
4. Launch Fiji.
5. Go to *File > New > Script...*
6. A dialog pops up. In the *Language* menu, select *ImageJ macro* if not already selected.
7. Copy, into the text script editor of Fiji, `analyse_dataset_save_rois_and_summary_table.ijm`.
8. You will be asked to enter your login credentials when you run the script.
9. Click *Run*.
10. Return to OMERO.web and open an Image from this Dataset in OMERO.iviewer.
11. Click the *ROIs* tab to see the added ROIs. Note that the ROIs have been assigned a Channel index to indicate which Channel they were derived from.
12. In the *Settings* tab, turning channels on/off will also show/hide ROIs assigned to those channels.

Segment using Groovy script

Description

The following workflows should work in Fiji, after these have been correctly set up with the OMERO plugin for Fiji/ImageJ.

Using the Scripting editor of Fiji, we will show:

- How to connect to OMERO using the JAVA API.
- How to load all the images within a given Dataset.
- How to load the images in ImageJ using Bio-Formats.
- How to analyze the images using *Auto Threshold* and *Analyze particles* plugins in ImageJ within a script.
- How to save the segmented ROIs as polygons in OMERO.
- How to collect the measurement associated to the ROIs and
 - Save them as an OMERO.table and link the table to the Dataset.
 - Save them as a CSV file and link it to the Dataset.

Setup

- Install Fiji on the local machine with the OMERO.insight-ij plugin. The installation instructions can be found at [How to install OMERO plugins for Fiji/ImageJ](#).

Resources

- Data: Samples images from the Image Data Resource (IDR) [idr0021](#).
- [Java API documentation](#).
- Script: Groovy script for automatic segmentation of images from OMERO using Fiji - `analyse_dataset_save_rois_and_summary_table.groovy`.

Step-by-Step

We will now repeat the manual analysis [Manual Segmentation](#) on a Dataset using the scripting facility available in Fiji.

Let's go over the script to understand the logic and see how it matches the UI steps.

This script explores the [JAVA API](#) using Groovy.

It will process all the Images in the specified Dataset, applying threshold, analyzing particles and saving ROIs back in OMERO i.e. we reproduce in a script the manual steps recorded. Further, it will create a CSV and OMERO.table to be attached to that Dataset in OMERO.

1. In your browser, go to the server address provided.
2. Log in using the credentials provided.
3. Make sure you are selecting your own data. Select the Dataset **A-Fiji-dataset**.
4. Launch Fiji.
5. Go to *File > New > Script...*

6. A dialog pops up. In the *Language* menu, select *Groovy*.
7. Copy, into the text script editor of Fiji, `analyse_dataset_save_rois_and_summary_table.groovy`.
8. You will be asked to enter your login credentials when you run the script.
9. Click *Run*.
10. Return to Omero.web and open an Image from this Dataset in Omero.iviewer.
11. Click the *ROIs* tab to see the added ROIs. Note that the ROIs have been assigned a Channel index to indicate which Channel they were derived from.
12. In the *Settings* tab, turning channels on/off will also show/hide ROIs assigned to those channels.
13. Open the image in Omero.figure for a quick publication by going to Info tab in iviewer and clicking on

OMERO.figure in the Open with line. 

Fiji and Zarr

View zarr file using MoBIE and BDV

Description

This section shows how to view ome.zarr files in Fiji using MoBIE.

We show:

- how to install the required dependencies
- how to view an ome.zarr file stored in S3 using the User Interface
- how to view an ome.zarr file stored in S3 using the scripting editor of Fiji.

Setup

- Install Fiji on a local machine.
- Go to `Help>Update...`
- In the ImageJ updater dialog, click on `Manage update sites`.
- The *MoBIE* should already be listed. Select it and click `Close`.
- Click `Apply changes` to install it.
- **If *MoBIE* is not listed:**
 - Click `Add update site`.
 - Enter for the name *MoBIE* (so you can identify it) and for the URL: `https://sites.imagej.net/MoBIE/`. See [How to install an update site](#).
 - Click `Apply changes` to install it.
- Restart Fiji.

Resources

- Samples images from the Image Data Resource (IDR) that have been converted into <https://github.com/ome/ngff>. A list of available files can be found [here](#).
- **Script: Groovy script for opening the images in BigDataViewer using MoBIE.**
 - `mobie_ome_zarr.groovy`.
- Watch an introductory [video](#).

Step-by-step

Opening an ome.zarr file from the User Interface

1. Launch Fiji.
2. Go to *Plugins > BigDataViewer > OME ZARR > Open OME ZARR from S3...*
3. A dialog pops up.
4. In the text field, enter the desired URL e.g.
`https://uk1s3.embassy.ebi.ac.uk/idr/zarr/v0.1/9836832.zarr`
1. Click the *OK* button.
2. When the image is displayed in the BigDataViewer, select the dialog and press *P* to display the rendering controls.
3. Modify the settings as you see fit.

Opening an ome.zarr file using a Groovy script

1. Launch Fiji.
2. Go to *File > New > Script...*
3. A dialog pops up. In the *Language* menu, select *Groovy*.
4. Copy the content of `mobie_ome_zarr.groovy` and paste it into the text script editor of Fiji.
5. Click *Run*.
6. When the image is displayed in the BigDataViewer, select the dialog and press *P* to display the rendering controls.
7. Modify the settings as you see fit.

Contribute

Changes to the documentation or the materials should be made directly in the [omero-guide-fiji repository](#).

Segment OMERO data using Fiji run headlessly in notebook

Description

The following examples follow up on the script shown in *Segment using Groovy script*. But, unlike in *Segment using Groovy script*, here, Fiji is included into a [Docker](#) image and run in headless mode. The scripts are embedded into a Jupyter Notebook. The Groovy language is used, which is very similar to Java and can be used in Fiji scripting. The scripts show:

- How to run a Fiji macro attached to an OMERO Dataset, save ROIs, Results and Images (as OME-TIFF) to OMERO
- How to crop an image in OMERO using Fiji cropping functionality.

Setup

Fiji has been installed in a Docker image using `repo2docker`.

Resources

- Data: Samples images from the Image Data Resource (IDR) [idr0021](#).
- Macro: `fiji-macro-segment.ijm`

Step-by-Step

1. Create a Fiji macro, for example by recording it and save it locally. You can for example use adjusted macro created in *Manual Segmentation*. See `fiji-macro-segment.ijm`.

Note: Not all commands which can be run and recorded in the Fiji user interface can be also run in the Fiji in headless mode.

2. In your browser, go to the server address provided.
3. Log in using the credentials provided.
4. Make sure you are selecting your own data. Select the Dataset **A-Fiji-dataset**.
5. Open the *Attachments* harmonica in the right-hand panel and click on the plus icon. Browse and attach the `fiji-macro-segment.ijm` to this Dataset as File Annotation.
6. Run the notebook [run_attached_macro.ipynb](#).
7. Select the first Step and click on the Run button to execute each step in turn.
8. For the connection to OMERO, you will be asked to enter your login details when running the first cell under *Description*. Note that the connection itself happens only in the next cell, highlighted as *Connect to OMERO*.
9. In the following cell, you will be asked to enter the ID of the Dataset onto which you attached the Macro file above.
10. The code in the next cell is fetching the attachments on the Dataset and offering a dropdown menu for you to select a suitable attachment i.e. your attached macro.
11. The next cell is asking for input about the types of data which should be saved back to OMERO (ROIs, Results or Images). This is of course dependent on the type of results your Macro is producing in Fiji. For this example, we select ROIs and Results.

12. The script in the next cell will process all the Images in the specified Dataset, applying threshold, analyzing particles, i.e. steps which are captured inside your macro file. Further, it will save ROIs back in OMERO and create a CSV to be attached to each image in that Dataset in OMERO.
13. Return to OMERO.web and open an Image from this Dataset in OMERO.iviewer.
14. Click the *ROIs* tab to see the added ROIs.
15. In the *Settings* tab, turning channels on/off will also show/hide ROIs assigned to those channels.
16. Open the image in OMERO.figure for a quick publication by going to the *Info* tab in OMERO.iviewer and clicking on OMERO.figure in the Open with line.

Open With: OMERO.figure |
17. Click the *ROIs* tab to see the added ROIs. Note that the ROIs have been assigned a Channel index to indicate which Channel they were derived from.
18. In the *Settings* tab, turning channels on/off will also show/hide ROIs assigned to those channels.
19. Run the notebook `crop_image.ipynb`.
20. Run through the notebook, logging in as previously and indicating the Image ID of the image you want to crop.
21. In OMERO.web, refresh and note that the cropped image has been imported into a newly created Dataset *Cropped image*.

Analyse data using ImageJ in Python

Description:

This section shows how to use ImageJ as a Python library to analyze data in OMERO.

Using the Python API allows us to easily load the 2D-plane we need to see or analyze. This is much easier than using the Java API and Bio-Formats plugin.

We will show in the examples:

- How to start ImageJ in Python.
- How to load data from OMERO.
- How to run ImageJ macro from Python.

Setup

For using the Python examples and notebooks of this guide we recommend using Conda. Conda manages programming environments in a manner similar to `virtualenv`.

Install `omero-py` and `pyimagej` via Conda:

- Install `Miniconda` if necessary.
- Create a programming environment using Conda and activate it:

```
$ conda create -n imagej_python python=3.6
$ conda activate imagej_python
```

- Install `omero-py` and `pyimagej`:

```
$ conda install -c conda-forge pyimagej
$ conda install -c ome omero-py
```

Note: we have noticed problems using the Conda approach when running the scripts on some operating systems due to issues with the ImageJ1-ImageJ2 bridge.

Step-by-Step

The script used in this document is `run_macro_python.py`. One of the advantages of this approach is that we can load only the 2D-planes we wish to analyze.

The script used in this document contains an ImageJ1 macro that needs graphical user interface (GUI) elements, and thus it requires using ImageJ in GUI mode. In this GUI mode, the resulting windows content is handled.

You will first need to update the script to point to your local installation of Fiji or use one of the options described in [ImageJ Tutorials](#). You can now run the script. To run the script, go to the folder `scripts/python` and run:

```
$ python run_macro_python.py
```

If you do not use any ImageJ1 features e.g. macro, you do **not** need the UI environment.

Below we explain the various methods in the scripts: how to start Fiji, how to load the planes to analyze and how to run an ImageJ1 macro.

In this example, Fiji has been installed locally.

Script's description

Import modules needed:

```
import imagej
from omero.gateway import BlitzGateway
```

Load Fiji. If you run the script locally, point to your local installation of Fiji or load Fiji “on the fly”. Note that the parameter *headless* has been set to *False* since we need the graphical user interface to run the ImageJ1 macro:

```
def start_fiji():
    ij = imagej.init('/srv/conda/vnc/Fiji.app', headless=False)
    ij.getVersion()
    return ij
```

Connect to the server. It is also important to close the connection again to clear up potential resources held on the server. This is done in the `disconnect` method:

```
def connect(hostname, username, password):
    """
    Connect to an Omero server
    :param hostname: Host name
```

(continues on next page)

(continued from previous page)

```

:param username: User
:param password: Password
:return: Connected BlitzGateway
"""
conn = BlitzGateway(username, password,
                    host=hostname, secure=True)
conn.connect()
conn.c.enableKeepAlive(60)
return conn

def disconnect(conn):
    """
    Disconnect from an OMERO server
    :param conn: The BlitzGateway
    """
    conn.close()

```

Load an image from IDR:

```
image = conn.getObject("Image", image_id)
```

Load the binary plane as numpy array:

```

def load_plane(image):
    """
    Load a 2D-plane as a numpy array
    :param image: The image
    """
    pixels = image.getPrimaryPixels()
    return pixels.getPlane(0, 0, 0)

```

To be used in ImageJ, the numpy array will be converted into ImageJ types using the `to_java()` method. In order the use the methods implemented above in a proper standalone script: **Wrap it all up** in an analyse method and call it from main:

```

def analyse(ij, conn, image_id):
    # Step 3 - Load image
    image = conn.getObject("Image", image_id)
    # -
    plane = load_plane(image)
    from jnius import autoclass
    autoclass('ij.WindowManager')
    ij.ui().show('Image', ij.py.to_java(plane))
    macro = """run("8-bit")"""
    ij.py.run_macro(macro)

def main():

```

(continues on next page)

(continued from previous page)

```

try:
    hostname = input("Host [wss://idr.openmicroscopy.org/omero-ws]: \
                    ") or "wss://idr.openmicroscopy.org/omero-ws"
    username = "public"
    password = "public"
    image_id = int(input("Image ID [1884807]: ") or 1884807)
    print("initializing fiji...")
    ij = start_fiji()
    print("connecting to IDR...")
    conn = connect(hostname, username, password)
    print("running the macro...")
    analyse(ij, conn, image_id)
finally:
    if conn:
        disconnect(conn)

if __name__ == "__main__":
    main()

```

3.2.3 ilastik

ilastik is a free open-source interactive learning and segmentation toolkit, with which can be used to leverage machine learning algorithms to easily segment, classify, track and count cells or other experimental data. For more details go to see <https://www.ilastik.org/>. We will show how to analyze data stored in OMERO, using the ilastik user interface, Fiji and the OMERO ImageJ plugin. We will then also show how to analyze images using the ilastik API and OMERO.py API.

Contents:

Install ilastik and OMERO Python bindings

In this section, we show how to install ilastik in a virtual environment. We will use the ilastik API to analyze data stored in an OMERO server. We will use OMERO.py to interact with the OMERO server.

Setup

We recommend to install the dependencies using [Conda](#). In case you do not have a conda installation we recommend downloading and installing the latest version of [Miniforge](#).

Create the environment running the commands below as written:

```

$ git clone https://github.com/ome/omero-guide-ilastik
$ cd omero-guide-ilastik
$ conda env create -f binder/environment.yml

```

and activate the newly created environment:

```
$ conda activate omero-guide-ilastik
```

Before creating a new environment, remember to deactivate the current one:

```
$ conda deactivate
```

See also [Conda command reference](#).

The following steps are only required if you want to run the notebooks.

- If you installed Miniforge:
 - Open jupyter notebook i.e. `jupyter notebook` and select the `omero-guide-ilastik` kernel or `[conda env:omero-guide-ilastik]` according to what is available.

To stop the notebook server, in the terminal where the server is running, press `Ctrl C`. The following question will be asked in the terminal `Shutdown this notebook server (y/[n])?`. Enter the desired choice.

- If you have an Anaconda installation:
 - Start Jupyter from the Anaconda-navigator
 - To register the environment, run `python -m ipykernel install --user --name omero-guide-ilastik`
 - Select the notebook you wish to run and select the `Kernel>Change kernel>Python [conda env:omero-guide-ilastik]` or `Kernel>Change kernel>omero-guide-ilastik`.

Getting started with ilastik API and OMERO

Description

We will use a Python script showing how to analyze data stored in an OMERO server using the ilastik API. The code snippets are extracted from the Python script `pixel_classification.py`. A notebook is also available, see [idr0062_pixel_classification.ipynb](#).

We will show:

- How to connect to server.
- How load images from a dataset using the OMERO API.
- How to run ilastik using its Python API.
- How to save the generated results as OMERO images. If you are accessing a public resource e.g. [idr.openmicroscopy.org](#), this step will not work.

Setup

We recommend to use a Conda environment to install ilastik and the OMERO Python bindings. Please read first [Install ilastik and OMERO Python bindings](#).

Resources

We will use an ilastik project created with ilastik version 1.3.3 to analyze 3D images of mouse blastocysts from the Image Data Resource (IDR).

- ilastik model.
- Images from IDR [idr0062](#).

For convenience, the IDR data have been imported into the training OMERO.server. This is **only** because we **cannot** save results back to IDR which is a read-only OMERO.server.

Step-by-Step

In this section, we go over the various steps required to analyse the data. The script used in this document is `pixel_classification.py`.

Connect to the server:

```
def connect(hostname, username, password):
    conn = BlitzGateway(username, password,
                        host=hostname, secure=True)
    conn.connect()
    return conn
```

Load the images:

```
def load_images(conn, dataset_id):
    return conn.getObjects('Image', opts={'dataset': dataset_id})
```

We are now ready to analyze the images:

```
def analyze(conn, images, model, new_dataset):
    # Prepare ilastik
    os.environ["LAZYFLOW_THREADS"] = "2"
    os.environ["LAZYFLOW_TOTAL_RAM_MB"] = "2000"
    args = app.parse_args([])
    args.headless = True
    args.project = model
    shell = app.main(args)
    for image in images:
        input_data = load_numpy_array(image)
        # run ilastik headless
        print('running ilastik using %s and %s' % (model, image.getName()))
        data = [ {"Raw Data": PreloadedArrayDatasetInfo(preloaded_array=input_data,
        ↪axistags=vigra.defaultAxiTags("tzyxc"))}] # noqa
        predictions = shell.workflow.batchProcessingApplet.run_export(data,
                                                                    export_to_
        ↪array=True) # noqa
        for d in predictions:
```

(continues on next page)

(continued from previous page)

```
save_results(conn, image, d, new_dataset)
```

The ilastik project used expects the data array to be in the order **TZYXC**. The order will need to be adjusted depending on the order expected in the ilastik project

```
def load_numpy_array(image):
    pixels = image.getPrimaryPixels()
    size_z = image.getSizeZ()
    size_c = image.getSizeC()
    size_t = image.getSizeT()
    size_y = image.getSizeY()
    size_x = image.getSizeX()
    z, t, c = 0, 0, 0 # first plane of the image
    zct_list = []
    for t in range(size_t):
        for z in range(size_z): # get the Z-stack
            for c in range(size_c): # all channels
                zct_list.append((z, c, t))
    values = []
    # Load all the planes as YX numpy array
    planes = pixels.getPlanes(zct_list)
    j = 0
    k = 0
    tmp_c = []
    tmp_z = []
    s = "z:%s t:%s c:%s y:%s x:%s" % (size_z, size_t, size_c, size_y, size_x)
    print(s)
    # axis tzyxc
    print("Downloading image %s" % image.getName())
    for i, p in enumerate(planes):
        if k < size_z:
            if j < size_c:
                tmp_c.append(p)
                j = j + 1
            if j == size_c:
                # use dstack to have c at the end
                tmp_z.append(numpy.dstack(tmp_c))
                tmp_c = []
                j = 0
                k = k + 1
        if k == size_z: # done with the stack
            values.append(numpy.stack(tmp_z))
            tmp_z = []
            k = 0
    return numpy.stack(values)
```

Let's now save the generated data and create a new OMERO image:

```
def save_results(conn, image, data, dataset):
    filename, file_extension = os.path.splitext(image.getName())
    # Save the probabilities file as an image
    print("Saving Probabilities as an Image in OMERO")
    name = filename + "_Probabilities"
    desc = "ilastik probabilities from Image:%s" % image.getId()
    # Re-organise array from tzyxc to zctyx order expected by OMERO
    data = data.swapaxes(0, 1).swapaxes(3, 4).swapaxes(2, 3).swapaxes(1, 2)

    def plane_gen():
        """
        Set up a generator of 2D numpy arrays.
        The createImage method below expects planes in the order specified here
        (for z.. for c.. for t..)
        """
        size_z = data.shape[0]-1
        for z in range(data.shape[0]): # all Z sections data.shape[0]
            print('z: %s/%s' % (z, size_z))
            for c in range(data.shape[1]): # all channels
                for t in range(data.shape[2]): # all time-points
                    yield data[z][c][t]

    conn.createImageFromNumpySeq(plane_gen(), name, data.shape[0],
                                data.shape[1], data.shape[2],
                                description=desc, dataset=dataset)
```

When done, close the session:

```
def disconnect(conn):
    conn.close()
```

Run ilastik in parallel using dask

Description

We will show how to use [dask](#) to analyze images in parallel using the ilastik API. Binary data are stored in a public S3 repository in the Zarr format.

Setup

We recommend to use a Conda environment to install ilastik and the OMERO Python bindings. Please read first [Install ilastik and OMERO Python bindings](#).

Step-by-Step

In this section, we go through the steps required to analyze the data. The script used in this document is `pixel_classification_zarr_parallel.py`.

Connect to the server:

```
def connect(hostname, username, password):
    conn = BlitzGateway(username, password,
                        host=hostname, secure=True)
    conn.connect()
    conn.c.enableKeepAlive(60)
    return conn
```

Load the images:

```
def load_images(conn, dataset_id):
    return conn.getObjects('Image', opts={'dataset': dataset_id})
```

Define the analysis function:

```
def analyze(image_id, model):
    args = app.parse_args([])
    args.headless = True
    args.project = model
    args.readonly = True
    shell = app.main(args)
    input_data = load_from_s3(image_id)
    # run ilastik headless
    data = [ {"Raw Data": PreloadedArrayDatasetInfo(preloaded_array=input_data,
    ↪axistags=vigra.defaultAxiTags("tzyxc"))}] # noqa
    return shell.workflow.batchProcessingApplet.run_export(data, export_to_array=True)
    ↪# noqa
```

Helper function load the binary as a numpy array from the Zarr storage format:

```
def load_from_s3(image_id, resolution='0'):
    endpoint_url = 'https://minio-dev.openmicroscopy.org/'
    root = 'idr/outreach/%s.zarr/' % image_id
    # data.shape is (t, c, z, y, x) by convention
    with ProgressBar():
        data = da.from_zarr(endpoint_url + root)
```

(continues on next page)

(continued from previous page)

```

values = data[:]
# re-order tczyx -> tzyxc as expected by the ilastik project
values = values.swapaxes(1, 2).swapaxes(2, 3).swapaxes(3, 4)
return numpy.asarray(values)

```

Start the Dask client and a local cluster:

```

cluster = LocalCluster()
client = Client(cluster)

```

Use the Dask Future API. The work starts immediately as we submit work to the cluster:

```

def prepare(client, images, model):
    futures = [client.submit(analyze, i.getId(), model) for i in images]
    return futures

```

We wait until this work is done and gather the results to our local process:

```

def gather_results(client, futures):
    return client.gather(futures)

```

When done, close the session:

```

def disconnect(conn):
    conn.close()

```

In order to use the methods implemented above in a proper standalone script: **Wrap it all up** in main:

```

def main():
    # Collect user credentials
    try:
        host = input("Host [wss://outreach.openmicroscopy.org/omero-ws]: ") or 'wss://
↳ outreach.openmicroscopy.org/omero-ws' # noqa
        username = input("Username [trainer-1]: ") or 'trainer-1'
        password = getpass("Password: ")
        dataset_id = input("Dataset ID [6161]: ") or '6161'
        # Connect to the server
        conn = connect(host, username, password)

        # path to the ilastik project
        ilastik_project = "../notebooks/pipelines/pixel-class-133.ilp"

        # Load the images in the dataset
        images = load_images(conn, dataset_id)

```

(continues on next page)

(continued from previous page)

```

# prepare ilastik
os.environ["LAZYFLOW_THREADS"] = "2"
os.environ["LAZYFLOW_TOTAL_RAM_MB"] = "20000"

# Create-client
cluster = LocalCluster()
client = Client(cluster)
# End-client

futures = prepare(client, images, ilastik_project)

start = time.time()
results = gather_results(client, futures)
done = time.time()
elapsed = (done - start) // 60
print("Compute time (in minutes): %s" % elapsed)
save_results(results)
finally:
    disconnect(conn)

print("done")

if __name__ == "__main__":
    main()

```

Use ilastik as a Fiji plugin and OMERO

Description

In this section, we will show how to use the ilastik user interface to perform segmentations on multi-z images stored in OMERO. The connection between OMERO and ilastik is facilitated via Fiji, for which both OMERO and ilastik have plugins. The segmentation steps in this part are recorded and saved in the form of an `ilp` file in ilastik (ilastik project). The `ilp` file is used later for the scripting workflow.

We will show:

- How to manually open images from OMERO in ilastik using the Fiji plugin for OMERO and Fiji plugin for ilastik.
- How to segment the multi-z images in ilastik and produce an ilastik Project (`ilp` file) recording the steps.
- How to save the results of the segmentation (ROIs and Probability maps) in OMERO, using the manual workflow and Fiji.
- How to run a script in Fiji, consuming the `ilp` file and running the segmentation of the images coming from an OMERO Dataset, saving the ROIs on the original images in OMERO.
- How to manually classify images.

Setup

ilastik installation

- ilastik has been installed on the local machine. See <https://www.ilastik.org/> for details.

ilastik plugin for Fiji installation instructions

- Start Fiji. Update it (Help > Update ImageJ).
- In the Manage Update Sites check the checkbox next to the “ilastik” site.
- After the update was successful, restart your Fiji.
- The new ilastik menu item should be under Plugins menu.

Note: The ilastik menu item might be the last in the Plugins dropdown, not necessarily alphabetically ordered.

OMERO plugin for Fiji installation instructions

- For installation instructions, go to [Fiji installation](#).

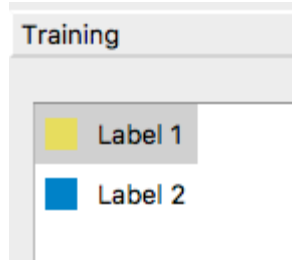
Resources

- Images from IDR [idr0062](#).

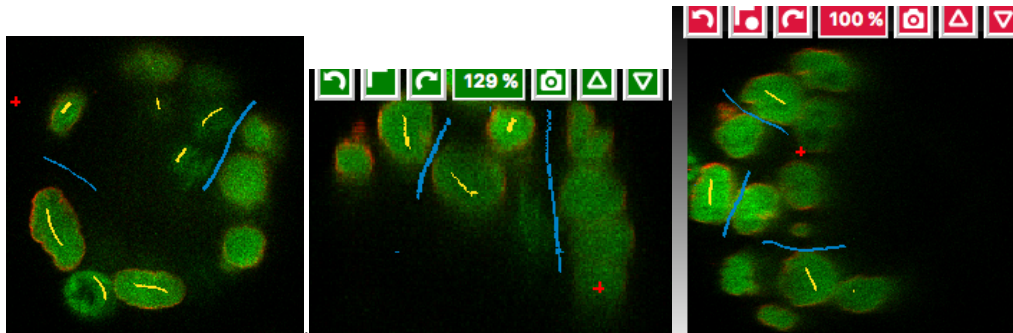
Step-by-step

Manual training of z-stack segmentation in ilastik

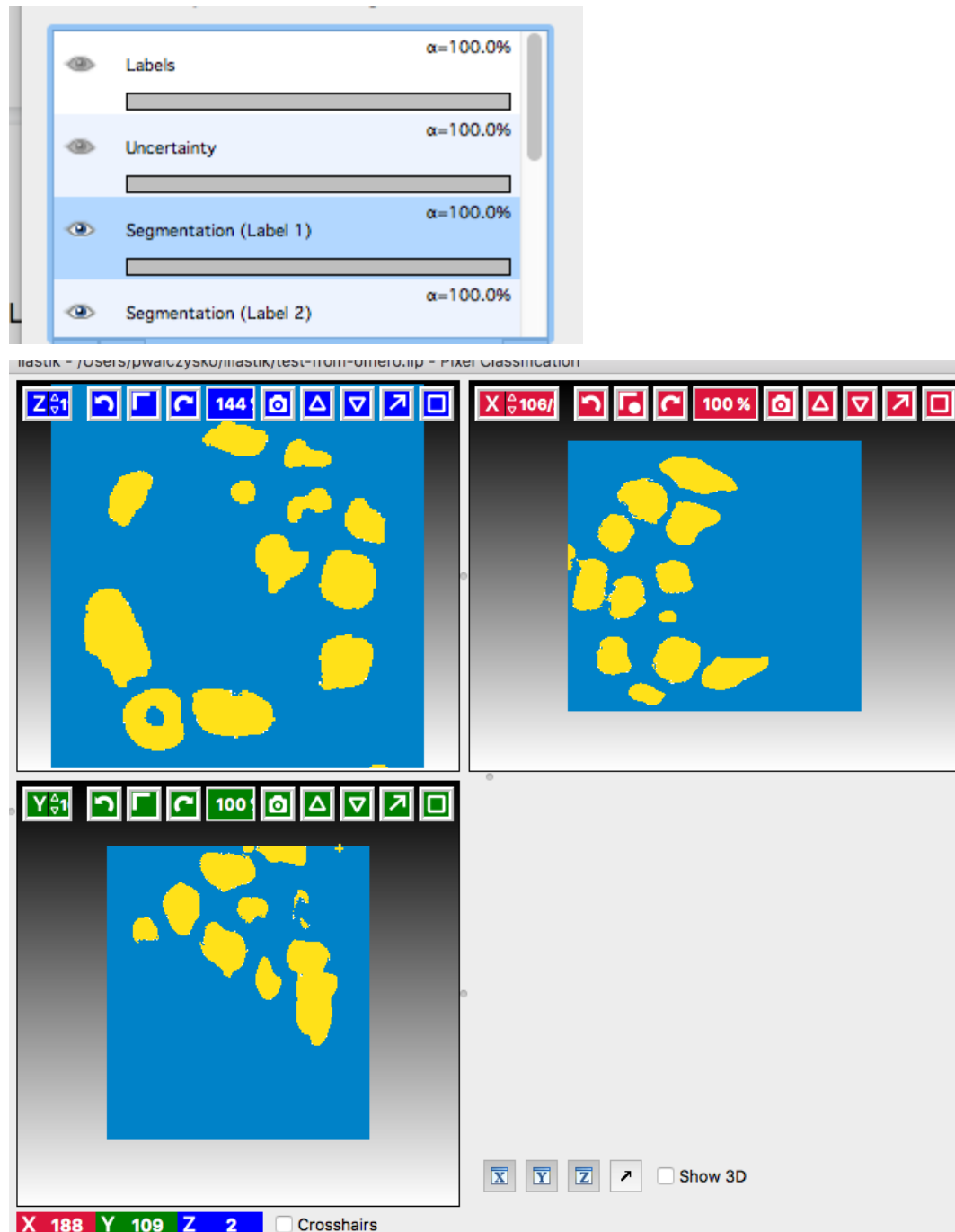
1. Open Fiji, go to Plugins > OMERO > Connect to OMERO and connect to OMERO.server provided using the credentials provided.
2. Find the idr0062 Project, the Blastocysts Dataset, open the first image in Fiji.
3. After image has opened in Fiji, go to Plugins > ilastik > Export HDF5. The ilastik menu item might be the last in the Plugins dropdown, not necessarily alphabetically ordered.
4. Select a local directory to export to and save the image locally as an .h5 file.
5. Repeat this step with several images from the [Blastocysts Dataset](#) of idr0062.
6. Start ilastik.
7. Click on Pixel Classification.
8. Save a new Project in ilastik.
9. Still in ilastik, open the image you saved as .h5 in previous steps above (central pane, Add file button).
10. Three views will open, xy, xz and yz. You can explore the orthogonal views by clicking onto the checkbox in bottom right corner.
11. In Left-hand pane, click Feature Selection. Select all available features.
12. You can explore the features at the bottom left corner, but this takes time.
13. Click on the Training harmonica in the Left-hand pane.
14. The training UI comes in left-hand pane with two labels already pre-defined by default.



15. Select the first label, and by drawing LINES into the images, select a couple of cells in all three views.



16. Select the second label, and again drawing lines, select some background (also select the narrow channels between two almost adjacent cells as bckgr (draw a line through them).
17. Click on the Live Update button - this will take time, as the image has a large number of planes.
18. Add new lines on cells which are too dim to be selected.
19. Click on Live Update.... Repeat.
20. Stop Live Update
21. Click on Suggest Features button (to the left of Live Preview button).
22. A new UI window will open.
23. Click on Run Feature Selection in the left-hand pane of this new window. This will take time.
24. Click on Select Feature Set button in the bottom middle of the window.
25. The Suggest Features window will close on this and you are back in the main ilastik window.
26. Click Live Update again.
27. Toggle the images produced visible or not using the eye icons and the rendering settings of the particular images in the list in bottom-left corner. Below is an example of viewing the Segmentation Label 1 and Segmentation Label 2 layers viewable, the other layers (e.g. Raw data) are toggled invisible.



28. Add new lines if some segmentation still does not look right.
29. Click on the Prediction Export harmonica tab. In this tab, we will prepare the parameters of the exported images only, and will do the exporting itself later using the Batch processing harmonica.
30. In the Prediction Export harmonica, select the features to be exported in the Source dropdown menu in the left-hand pane. Export sequentially Probabilities and Simple Segmentation for all three images you opened from OMERO via Fiji, using the Batch processing harmonica tab, see below.
31. First, start with selecting simple Segmentation in the Choose Export Image Settings, select the Convert to data Type parameter to be floating 32 bit

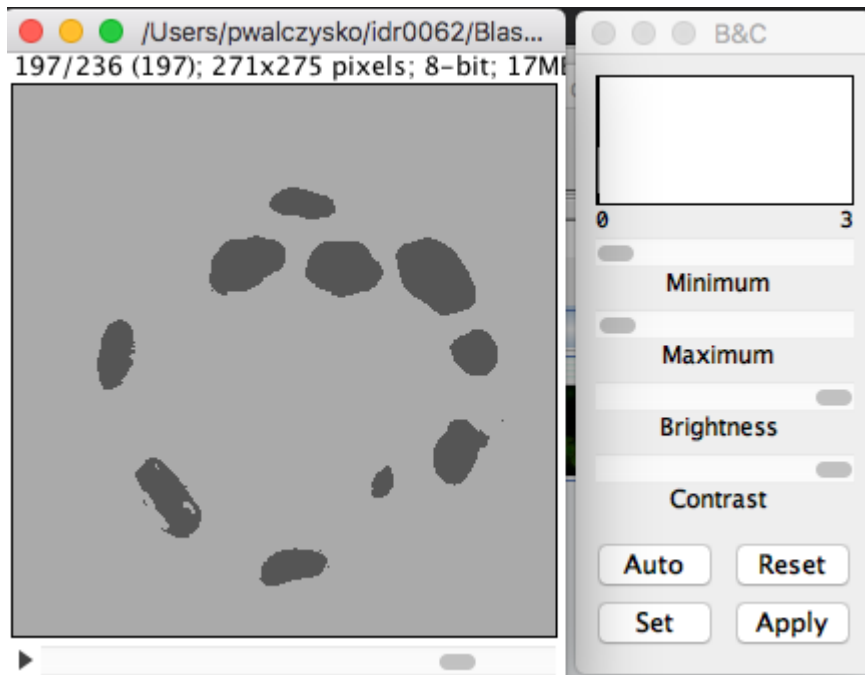
☒ Convert to Data Type: floating 32-bit

The files will be exported into the folder where the original images were, unless you choose otherwise. By default, the export format is HDF5 (file extension .h5).

32. Now, select in the left-hand pane the harmonica Batch processing. In the centre top row of the view, click on Select Raw Data Files.... Select all the three raw .h5 files on your local machine, including the one you have just trained your pixel classification on.
33. Click onto the Process all data files button in the left-hand pane.
34. This will create three .h5 files in the folder you have chosen in the Choose Export Image Settings window (by default, these files will be placed in the folder where your raw data exports from OMERO are), the files will be named ...Simple Segmentation.h5.
35. Return to Prediction Export harmonica, select the Probabilities parameter in the Source dropdown. Go to the Batch processing harmonica and click onto the Process all data files button in the left-hand pane. This will create another three .h5 files in the local folder, named ...Probabilities.h5.

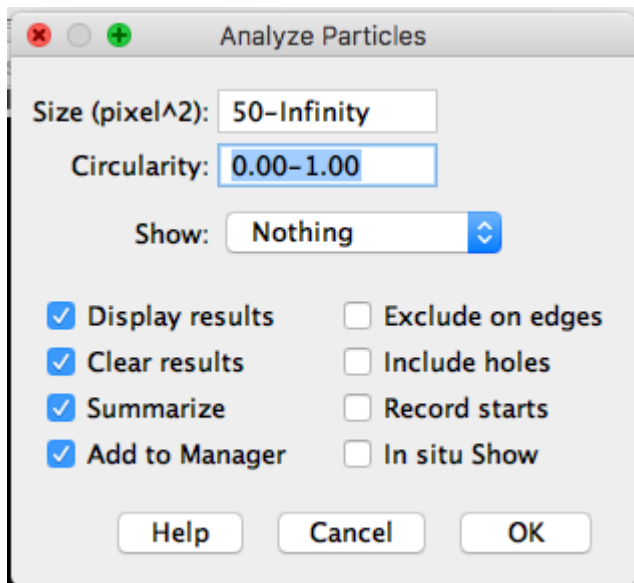
Manual creation of ROIs in Fiji based on segmentations from ilastik and saving the ROIs to OMERO

1. Go to Fiji, Plugins > Ilastik > Import...
2. Browse to one of the "..._Simple Segmentation.h5" files which was created in ilastik in previous step and set the "Axis Order" to tzyxc (this might be the default for you). Do not check the checkbox Apply LUT. Click OK.
3. The 3D image will open in Fiji. Select Image > Adjust > Brightness and Contrast. Adjust the max slider to the left, until you see the image grow grey (it is probably black just after opening).

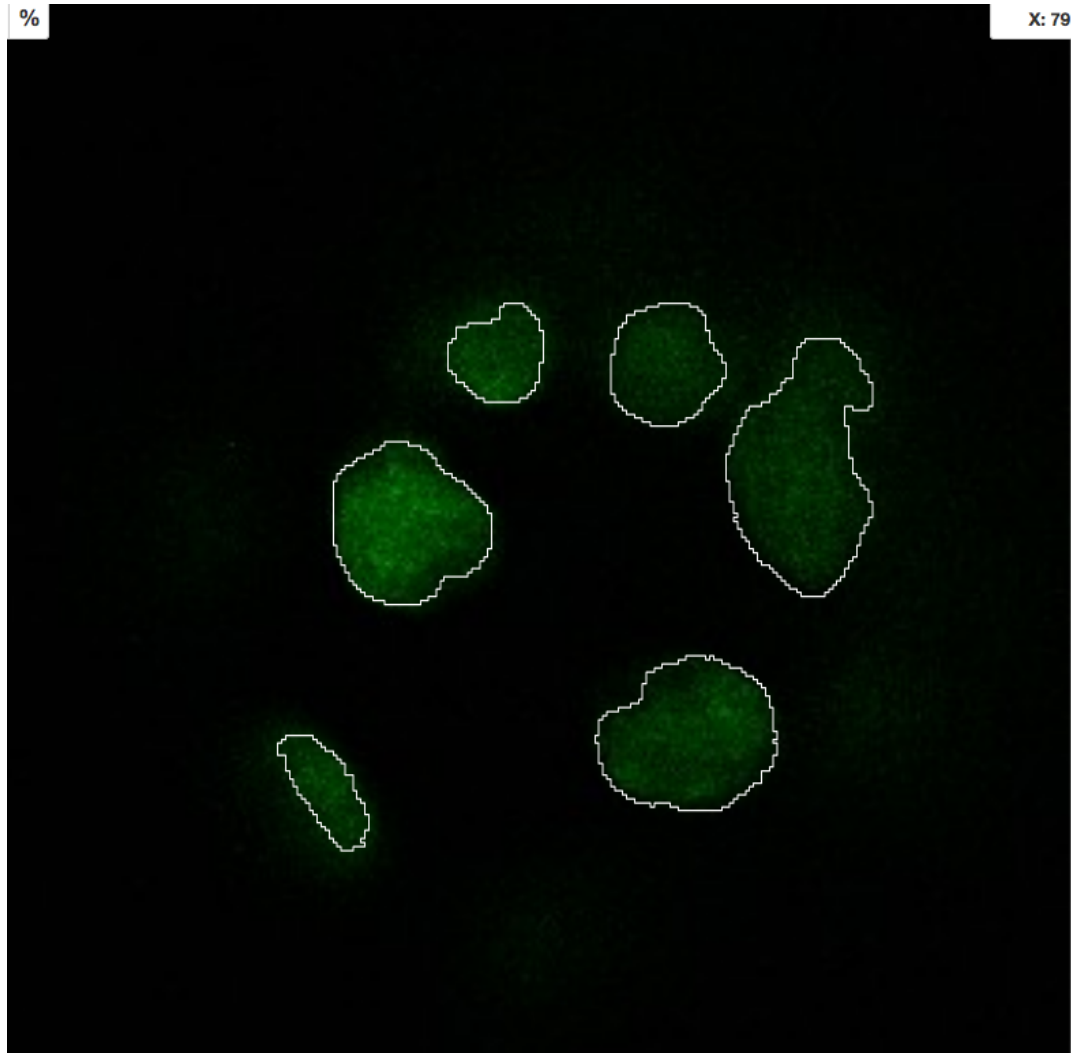


4. Note: Because in ilastik, the Simple Segmentation images have the values of 2 where there is an object and 1 for Background, we need to invert the image for object analysis in Fiji. The object analysis (done by the Analyze particles plugin) is done in order to create ROIs which can be saved to OMERO.
5. Select Image > 8 bit. This will convert the values in the image into either 0 (cells) or 255 (background).

6. Select **Edit > Invert**. This is needed for the subsequent **Analyze particles** plugin - white objects on black background.
7. Select **Analysis > Analyze Particles**.
8. Change the **Size(pixel²)** parameter to **50-infinity**



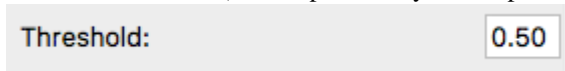
9. Click **OK** and in the next dialog answer **Yes**.
10. Select **Plugins > OMERO > Save image(s) to OMERO**. In the importer dialog, select the target **Project** and **Dataset** in OMERO or choose a new one.
11. This will import the **Simple segmentation** image into OMERO with the ROIs from Fiji on it and the contents of the **Results** table will be attached to this new image.
12. In order to have the ROIs from Fiji also on the original, raw image in OMERO.
13. Do not close the **ROI Manager** and the **Results** table.
14. Open the original raw image from OMERO into Fiji.
15. Click on the opened image.
16. Select **Plugins > OMERO > Save ROI(s) to OMERO** (alternatively, you can re-run the analysis in Fiji by clicking on **Measure** in the **ROI manager** of Fiji to produce a new **Results** table).
17. In the new dialog, select a name for your results table which will be attached now to the original image.
18. Click **OK**.
19. ROIs and results will be now added to the original, raw image in OMERO



20. Repeat this workflow with the ...Probabilities.h files. Also, attach the ilastik project itself to the Dataset containing original data in OMERO.

Manual workflow of Object classification on z-stacks in ilastik

1. Start ilastik, choose the **Object classification with Prediction maps** option and create a new Project and save it.
2. Select in the **Raw data** tab the raw image stored locally and in the **Prediction maps** tab the prediction map which you saved from the **Pixel classification** module for this image previously.
3. Click on **Threshold** and **Size filter harmonica** in the left-hand pane. This step discerns the objects from background by means of thresholding (note that the “Prediction maps” values are between 0 and 1, where 1 is 100% probability that the pixel is a cell, 0 is a 100% probability that the pixel is backgr.) The other parameter to specify the object except threshold in this tab is size of the object.
4. Threshold is 0.5 (if the probability of a pixel is higher than 0.5, then it is deemed to be a cell)

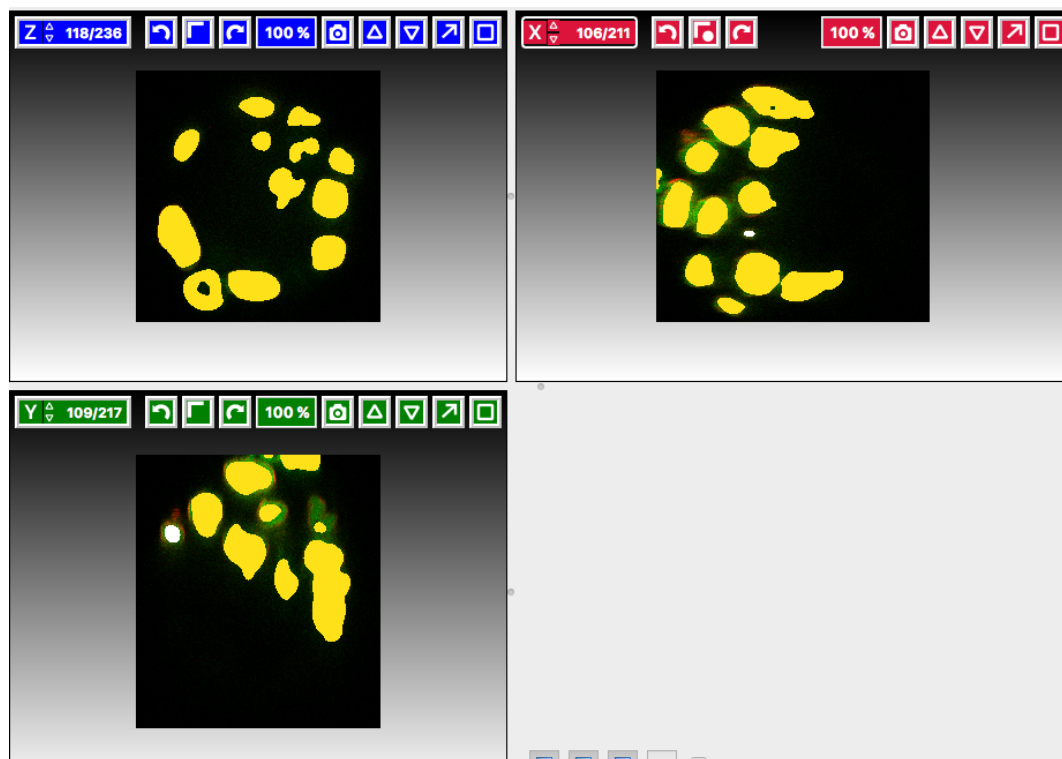


5. Change Size to minimum 50

Size FilterMin: Max: 999995

6. Leave the rest of the parameters at default and click Apply.
7. A new image will be added to the stack at bottom left called Final output. The objects are displayed on it in color coding. Again, you can toggle the images visible and change intensities in bottom left corner.
8. Click on Object Feature Selection harmonica and click on the button Select Features.
9. In the new window, click on All excl. Location button to select almost all features.

10. Click on the Label classes harmonica, click on the yellow label (Label 1) and select all the cells in all three orthogonal views images.



11. Click on Object information export harmonica.
12. Changing the Source dropdown menu, export sequentially Object Predictions and Object Probabilities.
13. Click on Configure Feature Table Export button in the left-hand pane and configure the location of the exported Also, changing the export format of the table in the Format dropdown menu, export sequentially the table as HDF as well as CSV

Format

14. In the Features harmonica, click the All button to export all features.
15. Click OK.
16. Back in the main ilastik interface, click Export All (repeat as necessary to export all formats of the images and the two formats of the export table).

17. Save the Project.
18. Import the CSV to OMERO, as well as the Probabilities.
19. Make an OMERO.table out of the CSV and attach it on the Project in OMERO. This can be done using [omero-metadata](#) plugin or from scratch using the extended groovy script from Fiji.

Use ilastik using Fiji scripting facility and OMERO

Description

In this section, the segmentation (using Pixel classification routine of ilastik) of the multi-z images is run in a batch mode. For this we provide scripts which run ilastik in a headless mode. The scripts provide for ilastik a batch of images (coming from an OMERO Dataset) and ilastik is segmenting these images according to the parameters configured and saved in the `ilp` in the manual step above. We offer two scripts covering this workflow, one running in Fiji, and the other using the python frames to export images directly from OMERO to the ilastik running headlessly. Also, we describe in this part how to use ilastik routine Object classification to classify objects on images from OMERO manually.

We will show:

- How to run a script in Fiji, consuming the `ilp` file and running the segmentation of the images coming from an OMERO Dataset,
- How to save the ROIs on the original images in OMERO.

Setup

ilastik installation

- ilastik has been installed on the local machine. See <https://www.ilastik.org/> for details.

ilastik plugin for Fiji installation instructions

- Start Fiji. Update it (Help > Update ImageJ).
- in the Manage Update Sites check the checkbox next to the “ilastik” site.
- After the update was successful, restart your Fiji.
- The new ilastik menu item should be under Plugins menu.

Note: The ilastik menu item might be the last in the Plugins dropdown, not necessarily alphabetically ordered.

OMERO plugin for Fiji installation instructions

- For installation instructions, go to [Fiji installation](#).

Resources

- Images from IDR [idr0062](#).
- Groovy script `analyse_dataset_ilastik.groovy`

Step-by-step

Scripting workflow on z-stacks using ilastik headless, Fiji and OMERO

For this example we will use the Groovy script `analyse_dataset_ilastik.groovy`. The script uses the [OMERO Java API](#).

Connect to the server:

```
def connect_to_omero() {
    "Connect to OMERO"

    credentials = new LoginCredentials()
    credentials.getServer().setHostname(HOST)
    credentials.getServer().setPort(PORT)
    credentials.getUser().setUsername(USERNAME.trim())
    credentials.getUser().setPassword(PASSWORD.trim())
    simpleLogger = new SimpleLogger()
    gateway = new Gateway(simpleLogger)
    gateway.connect(credentials)
    return gateway
}
```

Load the images contained in the specified dataset:

```
def get_images(gateway, ctx, dataset_id) {
    "List all images contained in a Dataset"

    browse = gateway.getFacility(BrowseFacility)
    ids = new ArrayList(1)
    ids.add(new Long(dataset_id))
    return browse.getImagesForDatasets(ctx, ids)
}
```

Open the images one-by-one using the Bio-Formats plugin:

```
def open_image_plus(HOST, USERNAME, PASSWORD, PORT, group_id, image_id) {
    "Open the image using the Bio-Formats Importer"

    StringBuilder options = new StringBuilder()
    options.append("location=[OMERO] open=[omero:server=")
    options.append(HOST)
    options.append("\nuser=")
    options.append(USERNAME.trim())
    options.append("\nport=")
    options.append(PORT)
    options.append("\npass=")
    options.append(PASSWORD.trim())
    options.append("\ngroupID=")
    options.append(group_id)
    options.append("\niid=")
```

(continues on next page)

(continued from previous page)

```

options.append(image_id)
options.append("] ")
options.append("windowless=true view=Hyperstack ")
IJ.runPlugIn("loci.plugins.LociImporter", options.toString())
}

```

Export each image as h5 to a local folder specified interactively by the user during the run of the script. It is assumed that the folder specified by the user contains the ilastik Project prepared beforehand. The export is facilitated by the ilastik plugin for Fiji.

```

IJ.run("Export HDF5", args);
imp = IJ.getImage()
imp.close()

```

Start ilastik headless, using the Pixel classification module The script feeds into the Pixel classification ilastik module the ilastik Project created during the manual step and also the raw the new created h5 image in the step above.

```

args = "select=" + output_file + " datasetname=" + inputDataset + " axisorder=" +
↪axisOrder
println "opening h5 file"
IJ.run("Import HDF5", args)
// run pixel classification
input_image = output_file + "/data";
args = "projectfilename=" + pixelClassificationProject + " saveonly=false inputimage=
↪" + input_image + " chosenoutputtype=" + outputType;
println "running Pixel Classification Prediction"
IJ.run("Run Pixel Classification Prediction", args);

```

The headless ilastik Pixel classification" module produces Probabilities map - this map is immediately opened into Fiji via the ilastik plugin for Fiji.

In Fiji, the Analyze Particles plugin is run on the "Probabilities" map to produce ROIs.

```

IJ.run("8-bit")
//white might be required depending on the version of Fiji
IJ.run(imp, "Auto Threshold", "method=MaxEntropy stack")
IJ.run(imp, "Analyze Particles...", "size=50-Infinity pixel display clear add stack,
↪summarize")

```

Once the ROIs are produced, they are saved to OMERO onto the original image which.

```

def save_rois_to_omero(ctx, image_id, imp) {
    " Save ROI's back to OMERO"
    reader = new ROIReader()
    roi_list = reader.readImageJROIFromSources(image_id, imp)
    roi_facility = gateway.getFacility(ROIFacility)
    result = roi_facility.saveROIs(ctx, image_id, exp_id, roi_list)
}

```

Disconnect when done

```
gateway.disconnect()
```

Track mitosis using ilastik

Description

In this section, a manual tracking workflow is shown on images of cells undergoing mitosis. The lineage of the cells is being followed. The images are timelapses from the Image Data Resource, the “mitocheck” set. As a result of this step, again, an `ilp` file is produced and saved for further use by the follow-up scripting workflow, similarly to the steps one and two described for the multi-z images above.

Setup

ilastik installation

- ilastik has been installed on the local machine. See <https://www.ilastik.org/> for details.

Step-by-step

1. Open ilastik, create a new Pixel classification project, feeding in the raw data in h5 form. The data come from <https://www.ilastik.org/download.html>, more concretely the “Mitocheck 2D+t” download <https://data.ilastik.org/mitocheck.zip>. Download, unzip and feed the h5 file which has not the “export” in its name into this step (Pixel classification).
2. Follow the steps of Pixel classification as described above in the `idr0062` workflow - you will have to
 - Adjust the parameters, saving the new project as “mitocheck-pixel-class.ilp”
 - Export “Probabilities”, which can be exported as “mitocheck_94570_2D+t_01-53_Probabilities.h5”
 - Close and reopen ilastik. Open the projec “conservationTracking.ilp” from the folder you downloaded from the ilastik site. In the “Raw data”, tab of “Input data” make sure the raw data are pointing to where you have your “mitocheck_94570_2D+t_01-53.h5” file locally. Further, in the “Prediction maps” tab of “Input data”, exchange the file there by right-clicking on it and selecting the “Replace with file” and replace this file with the “mitocheck_94570_2D+t_01-53_Probabilities.h5” which you exported from the Pixel classification workflow (see points above)
 - Run through the tabs in the LHP, making sure that when Thresholding, you swap the blue and yellow objects (my Pixel class. produced a probabilities map which is swapped in the sense objects vs bckgr coloring). Also, you have to manually select the cells which are dividing and not dividing in the corresponding tabs in LHP in quite a few timeframes, see <https://www.ilastik.org/documentation/tracking/tracking#3-division-and-object-count-classifiers> for how to do it.
 - Further, you have to discern false detections, and 1 object and 2 object blobs manually on quite a few frames, the LHP harmonice is called Object Count classification, as described in <https://www.ilastik.org/documentation/tracking/tracking#3-division-and-object-count-classifiers>, second part.
 - Once done, in the Tracking tab in left-hand paneHP, click on “Track !” button, making sure you did not change any params inadvertently. This will take a while.
 - **Select the “Tracking Results Export” tab in LHP and define your export target dir, then export in a row**
 - “mitocheck_94570_2D+t_01-53_Object-Identities.h5”,

- “mitocheck_94570_2D+t_01-53_Tracking-Result.h5”,
- “mitocheck_94570_2D+t_01-53_Merger-Result.h5” and
- “mitocheck_94570_2D+t_01-53_CSV-Table.h5.csv”

These are 3 timelapses and one CSV with the tracking results.

- Save the Project as “mitocheck-tracking-serious.ilp”. This is the main starting point for the automatic pipeline from OMERO. The pipeline is
 - “mitocheck-pixel-class.ilp” which
 - * consumes the “mitocheck_94570_2D+t_01-53.h5”
 - * produces the “mitocheck_94570_2D+t_01-53_Probabilities.h5”
 - “Mitocheck-tracking-serious.ilp” which
 - * consumes
 - “mitocheck_94570_2D+t_01-53.h5”
 - “mitocheck_94570_2D+t_01-53_Probabilities.h5”
 - * produces the outputs
 - “mitocheck_94570_2D+t_01-53_Object-Identities.h5”
 - “mitocheck_94570_2D+t_01-53_Tracking-Result.h5”
 - “mitocheck_94570_2D+t_01-53_Merger-Result.h5”
 - “mitocheck_94570_2D+t_01-53_CSV-Table.h5.csv”

Contribute

Changes to the documentation or the materials should be made directly in the [omero-guide-ilstik repository](#).

3.2.4 Orbit

Orbit is a free open-source software with the focus on quantification of big images like whole slide scans. It offers sophisticated image analysis algorithms. Of those, tissue quantification using machine learning techniques, object/cell segmentation, and object classification are the basic ones. We demonstrate how to integrate Orbit and OMERO using both the User Interface and the API. For more details, go to <http://www.orbit.bio/>

Contents:

Orbit Image Analysis

Orbit Image Analysis is a free open-source software which can connect to OMERO and work with images stored in OMERO server.

Description:

Orbit Image Analysis is a free open-source software with the focus on quantification of big images like whole slide scans. It offers sophisticated image analysis algorithms. Of those, tissue quantification using machine learning techniques, object/cell segmentation, and object classification are the basic ones. For more details, go to <http://www.orbit.bio/>.

We show:

- How to connect Orbit to OMERO
- How to open an image using Orbit user interface from OMERO.
- How to create classes in Orbit, draw ROIs, train, segment and detect objects in Orbit.
- How to save a “model” of the trained sequence from Orbit to OMERO.
- How to write a script which
 - Retrieves the saved model from OMERO to Orbit.
 - Executes the model on a part of an image.
 - Does the segmentation and draws ROIs around the segmented objects.
 - Saves the ROIs on the original image in OMERO.

Setup:

Orbit has been installed on the local machine. See <http://www.orbit.bio/> for details.

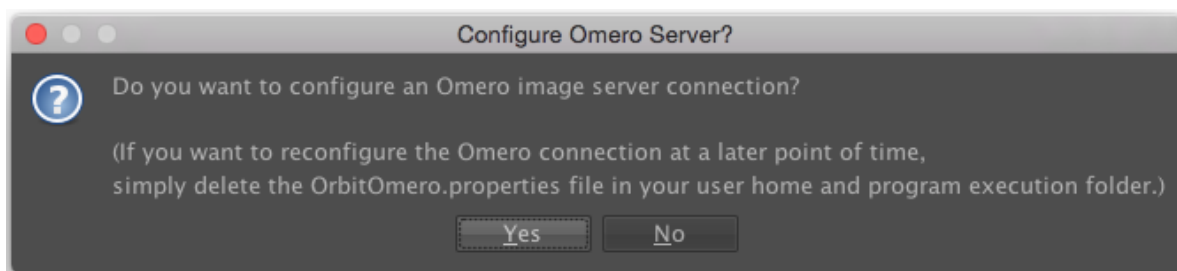
Resources:

- Data used <https://downloads.openmicroscopy.org/images/SVS/>
- Script used <https://raw.githubusercontent.com/ome/training-scripts/master/practical/orbit/segmentation.groovy>

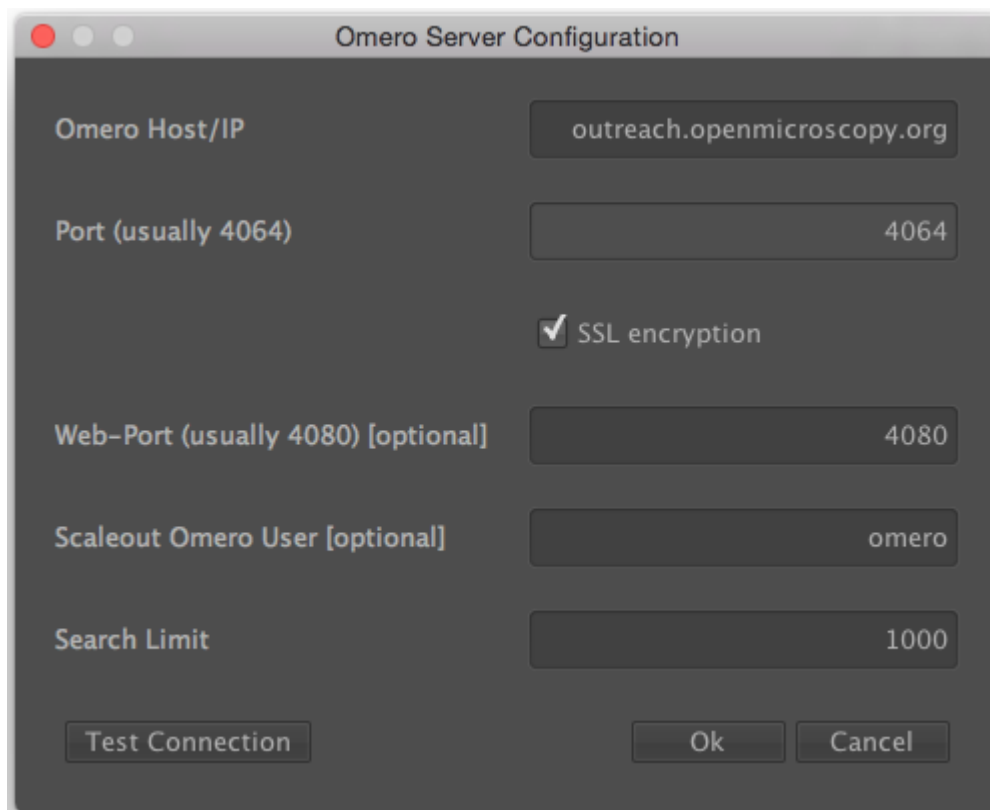
Step-by-Step:

Manual training:

1. Launch Orbit, click *Yes* in the first dialog box:



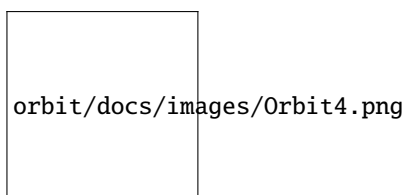
2. Then enter the server details in the next dialog. Only the Host/IP field is essential here and should be set to `<omero-server.address>`. For example, the OMERO server address could be added in the form of `demo.openmicroscopy.org` (Please replace with your own OMERO.server address).



3. Then login to OMERO with the credentials provided:



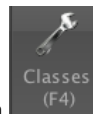
4. Orbit will show data from OMERO in the left-hand panel. Click show only my assets to filter by data your own.
5. Select the group Lab1.
6. Datasets not within a Project are listed under unassigned:



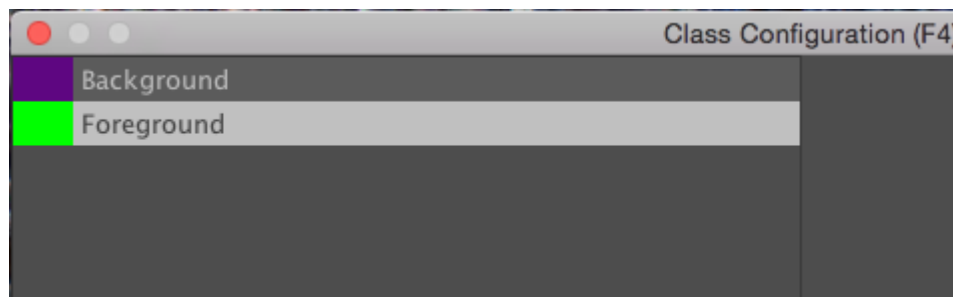
7. Select the svs Dataset. Image thumbnails will be shown in the panel below.
8. Double-click on the 77928.svs [Series 1] thumbnail or drag and drop it into the centre panel and maximise the viewer window to fill the centre panel.



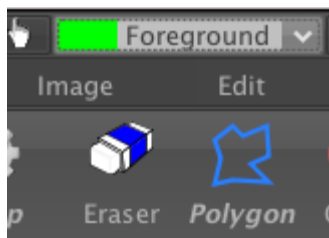
9. The aim is to train a model to recognize cell nuclei and use this for segmentation.



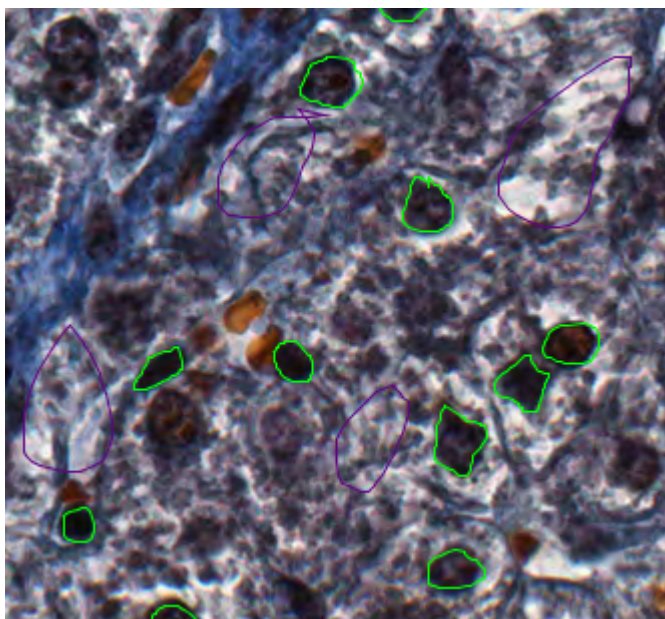
10. Click the Model tab and then the Classes button
11. In the dialog, remove the Celltype 1 class by selecting it and click the remove class button.
12. The select the Celltype 2 class, and rename it by typing Foreground in the name field and clicking rename class.
13. You should now have two classes named Background and Foreground:



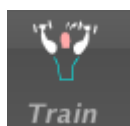
14. Click OK to close the dialog.
15. Now construct the model by defining regions of Foreground and Background on the Image.
16. Click the Object Detection tab, select the Polygon tool and choose the Foreground class from the chooser at the top-left of the screen.



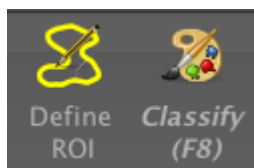
17. You can now draw around a number of cell nuclei on the Image. The more accurately you draw and the higher number of objects you define, the more you will improve the performance of the segmentation, but about a dozen should be sufficient.
18. Now switch to the Background class and draw around several background regions.



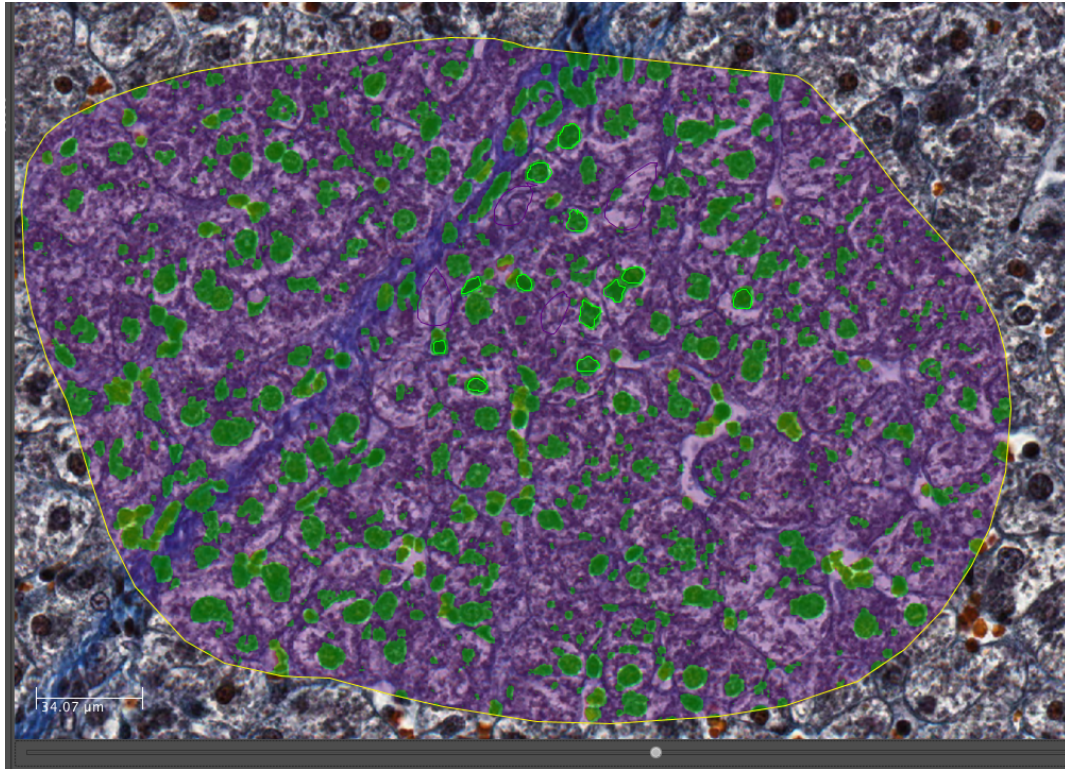
19. We can then train the model by clicking the Train button or press F7. You will see a progress bar in the right-hand panel.



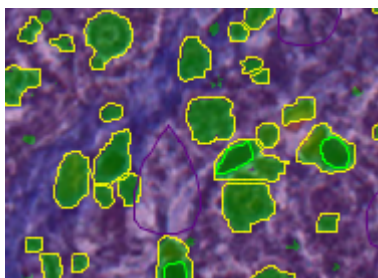
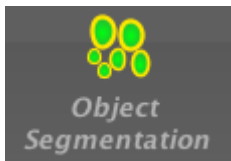
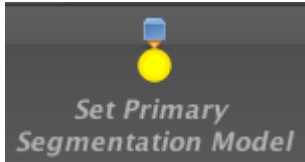
20. To see how this model classifies objects within a region, click the *Define ROI* button and draw around a region of the image. Then click *Classify*. If no ROI is drawn, Orbit will attempt to classify the whole Image which can be very time-consuming.



21. Once the classification is complete, a notification window pops up. Close it and view the results on the Image by dragging the slider below the Image to the right:



22. To segment the Image using this classification, click *Set Primary Segmentation Model* and then *Object Segmentation*.



23. Click the *Model* tab and *Save Model On Server*, enter a name to save the model to OMERO. Note that you can also use *Save Model as...* to save the model to your local drive.

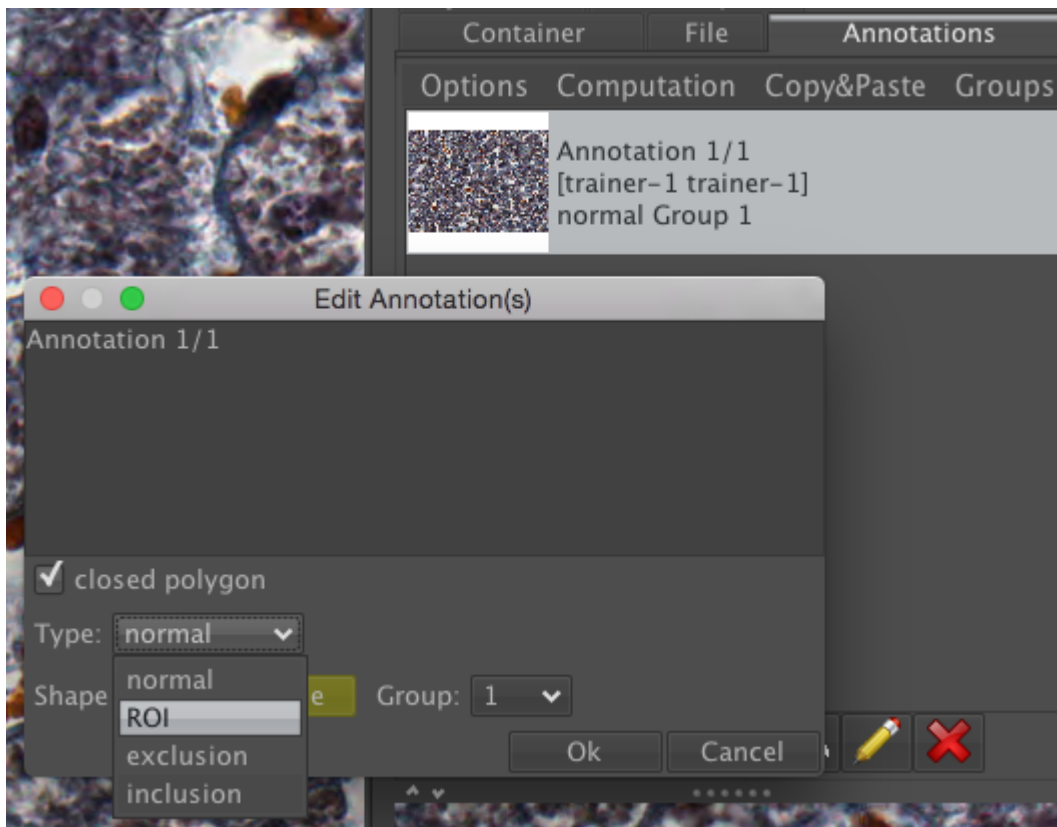
Scripted segmentation and saving to OMERO

We will use the model created in the last step above to repeat the segmentation, using a script which allows us to save the results back to OMERO. This will use a saved ROI Annotation instead of a temporary ROI as in the manual workflow.

1. Re-open the same image 77928.svs [Series 1] to clear the ROIs and in the right-hand panel select the Annotations tab.
2. Pan the Image to a region you wish to analyse, select the *Add Polygon* button and draw around a region.



3. Select this Annotation from the list in the right panel and click Edit (pencil icon).
4. In the dialog, set the *Type* to *ROI*.



5. Click Ok. This will save the ROI as an annotation on this image in OMERO.
6. Click on *Tools > Script Editor* to open a scripting window.
7. Copy the script from training-scripts: <https://raw.githubusercontent.com/ome/training-scripts/master/practical/orbit/segmentation.groovy> and replace the existing code in the script window.
8. Update the username and password
9. The script will load the Orbit model and the ROI that we saved to OMERO, segment the image within the ROI and save the segmented shapes as Polygons to OMERO.
10. Click *Run*.
11. When complete, you can use OMERO.iviewer to see the ROIs created in OMERO.

Contribute

Changes to the documentation or the materials should be made directly in the [omero-guide-orbit repository](#).

3.2.5 QuPath

[QuPath](#) is a free open-source cross-platform software application designed for bioimage analysis - and specifically to meet the needs of whole slide image analysis and digital pathology. We demonstrate how to integrate QuPath and OMERO using the User Interface.

Installation

How to install OMERO extension for QuPath

Description

In this section, we will cover the steps required to install the OMERO plugin for QuPath.

Installation

- Download QuPath [v0.5.1](#).
- Launch QuPath.
- Download the [OMERO extension for QuPath v0.4.0](#).
- Drag the `qupath-extension-omero-0.4.0.jar` file onto the main QuPath window. If you have not installed any extensions before, you will be prompted to select a QuPath user directory. The extension will then be copied to a location inside that directory.
- Restart QuPath

Note: The QuPath team is working on newer versions of the extensions under [qupath-omero-extension](#). These versions are currently under development and are meant only for testing.

Analyze

Analyze OMERO data using QuPath

Description

QuPath is a cross-platform software application designed for bioimage analysis - and specifically to meet the needs of whole slide image analysis and digital pathology. See <http://qupath.readthedocs.io/>.

We will show:

- How to connect QuPath to OMERO.server and browse the data.
- How to open an image including ROIs from OMERO.server in QuPath.
- How to draw an Annotation in QuPath and save it on an image in OMERO as OMERO ROI.

- How to perform simple Cell detection in QuPath.
- How to save the Cell detection ROIs directly to OMERO using a script in QuPath.
- How to export the Cell detection ROIs from QuPath as OME-XML.
- How to import the OME-XML to the OMERO.server and attach the QuPath ROIs to the original image in OMERO.

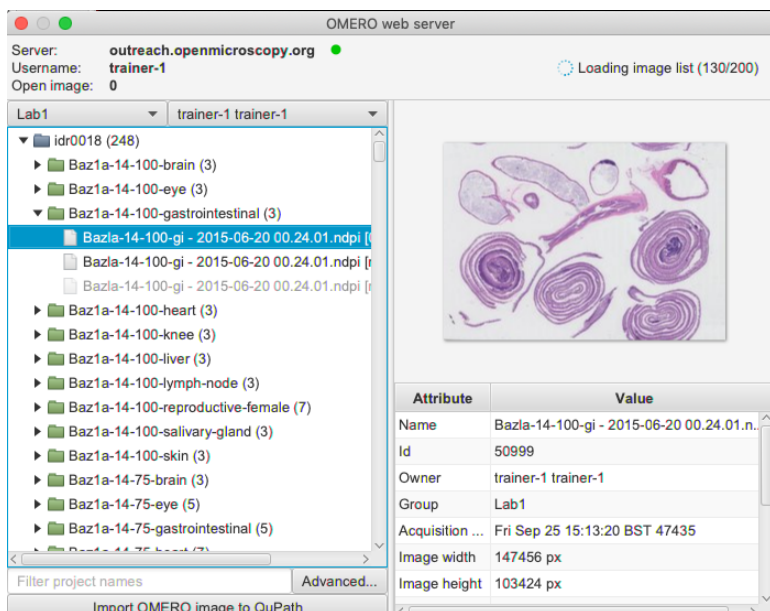
Resources

- Data: example images from
 - IDR project referenced as [idr0018](#). Note that the data also have been imported into an OMERO.server where the possibility to write ROIs/annotations exists (not the IDR server itself). See the [Step-by-step](#) section for further details.
- [Video](#) showing the usage of the QuPath OMERO extension
- QuPath documentation describing the [QuPath OMERO extension](#).
- Plugin `ome-omero-roitool v0.2.4` for import and export of ROIs to or from OMERO using OME-XML format. The `ome-omero-roitool-xxx.zip` under Releases also contains the scripts for export and import of ROIs from/to QuPath in OME-XML format. For precise installation steps, see below the [Step-by-step](#) section.
 - <https://github.com/glencoesoftware/ome-omero-roitool>

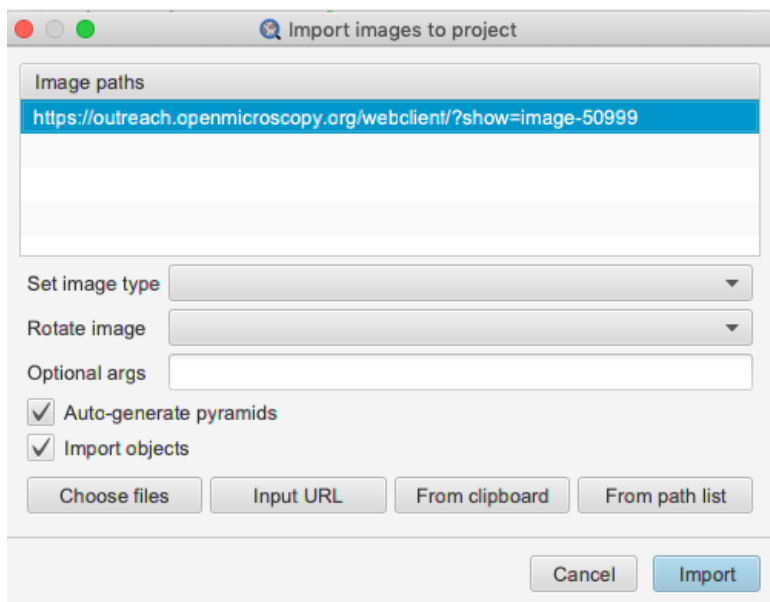
Step-by-step

Opening images with ROIs from OMERO in QuPath

1. You can go through this workflow directly using the Images from the IDR. Nevertheless, as you cannot write any data directly into IDR during your analysis, you will not be able to successfully import the resulting Annotations and ROIs back into the OMERO in IDR. Thus, you might consider using another OMERO.server which you can write data to and upload this or another RGB large image into it.
2. In OMERO.web, identify an image in the [idr0018](#) project and the dataset `Baz1a-14-100-gastrointestinal` contained in that project.
3. Select the first image and double-click on it. This will open the image in OMERO.iviewer, in a new tab of your browser.
4. If on a read-write OMERO server (i.e. not IDR), you can draw and save some ROIs on that image in OMERO.iviewer to be able to open them in QuPath later below, see [OMERO.iviewer guide](#) for how to do it.
5. Start your locally installed QuPath. Create a new QuPath Project by creating a new empty folder on your machine and dropping this new folder into the main QuPath window. Answer Yes when prompted.
6. Create a connection to your OMERO server by clicking `Extensions > OMERO > Browse server > New server` and paste into the dialog a valid server url including the `http` or `https` motives, for example `https://<server>.com`. Details are described in the [Browsing an OMERO server chapter](#) of the QuPath documentation.
7. Once connection to the server is established, QuPath will pop up a new dialog. In this dialog, select the correct group in OMERO in top left corner and the correct user. Expand Projects and Datasets as necessary, selecting the image with ROIs which you worked on in previous steps.



8. Double click on the image in the tree. In the new window, select the image again in the Image paths list, check the Import Objects checkbox and click Import in bottom-right corner.



9. Click on the imported image in your QuPath project to open it in QuPath. Inspect the ROIs imported from OMERO.
10. To draw new ROIs or annotations in QuPath, find a region with well-defined cells and nuclei in the image, zoom in.
11. Draw an Annotation which denotes the region in which the cells will be detected using the Wand tool .
12. Select the Annotations tab, select the class from the list to the right (e.g. Stroma) and click Set class. Click Extensions > OMERO > Send annotations to OMERO. A dialog will inform you how many ROIs are to be saved. Click OK.
13. Go to OMERO.iviewer, refresh the image and verify that the annotation was saved as an OMERO ROI (polygon).

14. Note that there is some loss of metadata when going through the Extensions > OMERO > Send annotations to OMERO step
 - The Class of the Annotation in QuPath will be indicated only by a fill color of the ROI in OMERO. If you reopen the image in QuPath again from OMERO, the ROI fetched by QuPath from OMERO will have the correct name of the Annotation if you gave it one in QuPath, but both the Class as well as the Annotation color will be lost by the round trip to OMERO and back.
 - All the holes in your Annotation will be ignored (filled in), as the Annotation is translated into a polygon ROI in OMERO. The ROI in OMERO will appear as a filled-in object, as shown in the cartoon in the [Send objects back to your OMERO server chapter](#) of the QuPath documentation.
 - The “derived” ROIs which were created for example by Cell detection algorithm in QuPath will be ignored when saving Annotations to OMERO. To save them either [Save detection ROIs using QuPath script](#) or [ome-omero-roitool](#) workflows can be used.

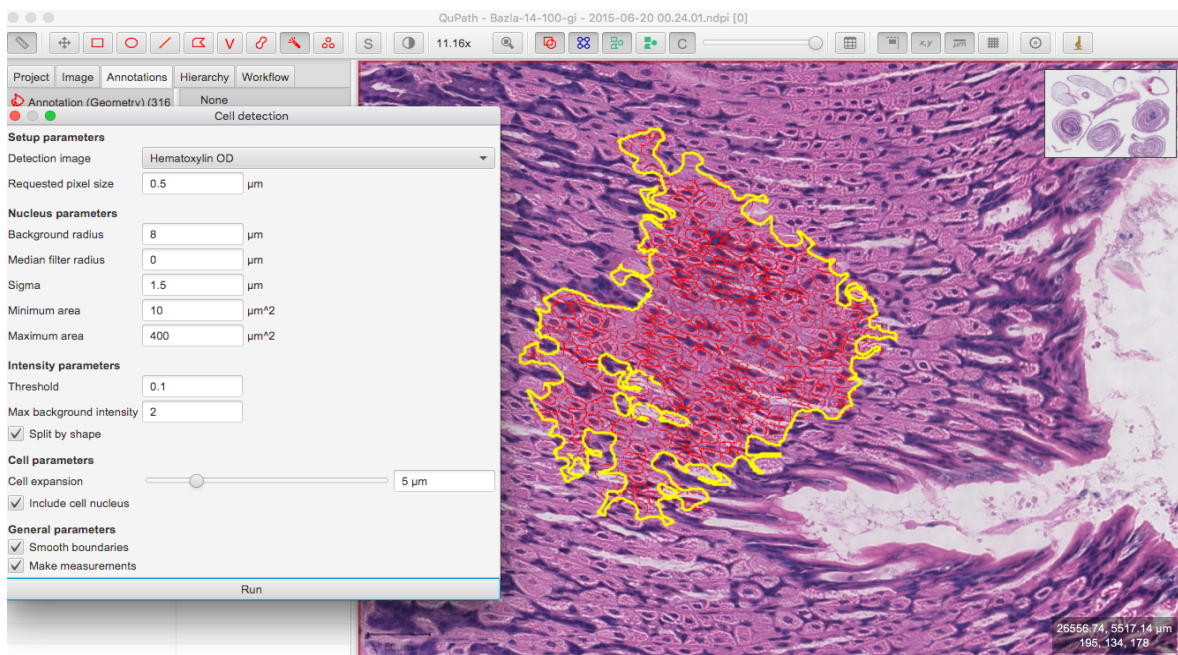
Saving of derived ROIs from QuPath to OMERO

The QuPath plugin for OMERO described above allows saving of the Annotations drawn in QuPath to OMERO, but it does not enable the saving of “derived” ROIs, such as Cell detection ROIs. To save the Cell detection ROIs either [Save detection ROIs using QuPath script](#) or [ome-omero-roitool](#) workflows can be used.

Save detection ROIs using QuPath script

Warning: The feature described in [Save detection ROIs using QuPath script](#) was not really designed for saving large amounts of ROIs (thousands) back to OMERO. An attempt to save large amounts of ROIs might result in slow performance or other problems.

1. Connect QuPath to OMERO, open an image from OMERO in QuPath and draw an Annotation on it as described in [Opening images with ROIs from OMERO in QuPath](#).
2. Adjust your Annotation using the Brush tool .
3. Select Analyze > Cell detection > Cell detection.
4. You can adjust the parameters. Click Run. This will draw red ROIs around cells and nuclei inside your Annotation.



5. Click on Hierarchy tab in the left-hand pane of QuPath. Expand the Annotation you have just run the Cell detection on.
6. Select several detection ROIs.
7. Open the scripting dialog in QuPath Automate > Script editor and paste into it the following code:

```
import qupath.lib.images.servers.omeiro.OmeroTools
OmeroTools.writePathObjects(getSelectedObjects(), getCurrentServer())
```

8. Click Run. This saves the detection ROIs you selected in the Hierarchy tab into OMERO.
9. Go to OMERO.iviewer and refresh the image. Inspect the saved detection ROIs.

Save detection ROIs using ome-omero-roitool

This workflow necessitates the usage of the Command Line Interface. The limitation here are the Annotation ROIs, which are transformed into masks in OMERO. Although this preserves the holes in the Annotations, if the Annotation ROIs are too large, it might result in performance problems or even running out of resources on the machine where the export of the mask from QuPath is attempted.

1. Connect QuPath to OMERO, open an image from OMERO in QuPath and draw an Annotation on it as described in [Opening images with ROIs from OMERO in QuPath](#).
2. Adjust your Annotation using the Brush tool .
3. Select Analyze > Cell detection > Cell detection.
4. You can adjust the parameters. Click Run. This will draw red ROIs around cells and nuclei inside your Annotation.
5. Use the ROI OME-XML export script to export your ROIs from QuPath into OME-XML file. Find the version of ome-omero-roitool mentioned in Resources on [ome-omero-roitool releases](#) and from there download the ome-omero-roitool-xxx.zip. The downloaded zip contains both the plugin and the QuPath scripts needed for this workflow.

6. Unzip the downloaded artifact and drag and drop the `OME_XML_export.groovy` into your QuPath.
7. To run the script, select `Run > Run`.
8. Note: If you run a `Cell detection` in QuPath, the nuclei ROIs will be drawn as well as the ROIs around the cells. The ROI OME-XML export script will export both the ROIs around the cells as well as the nuclei ROIs.
9. Import the OME-XML with the ROIs from QuPath into OMERO. These steps must be run on a command line.
10. Open your terminal window and `cd` into the directory containing the `ome-omero-roitool-xxx` folder downloaded in previous steps, then run:

```
cd ome-omero-roitool-xxx  
cd bin
```

11. On Mac or Linux, run:

```
./ome-omero-roitool import --help
```

12. On Windows, run:

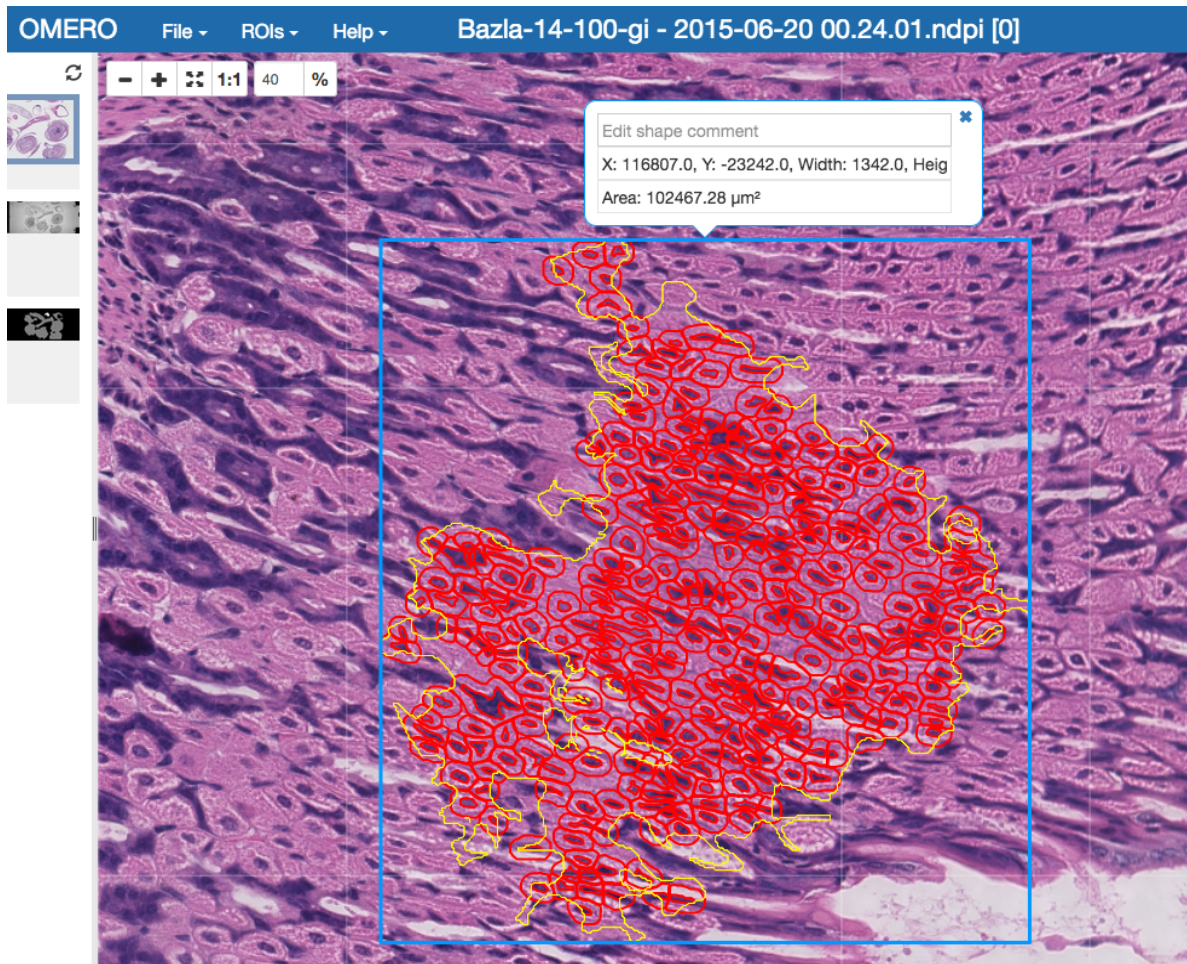
```
ome-omero-roitool.bat import --help
```

13. The `--help` option will give you a helpful output about how to construct the import command.
14. In the command below, replace the `$IMAGE_ID` parameter with the ID of the image in OMERO. You can obtain this ID for example from OMERO.iviewer (see beginning of this workflow).
15. To achieve the import of the ROIs to OMERO, you can run:

```
./ome-omero-roitool import --password $PASSWORD --port 4064 --server $SERVER --  
↪username $USERNAME $IMAGE_ID $PATH/TO/OME-XML/FILE
```

Note: if you are using websockets, set the port to 443 and the server with the protocol e.g. `wss://outreach.openmicroscopy.org/omero-ws`.

16. After you executed the `import` command above, go to OMERO.iviewer in your browser and view the ROIs on the image. The Annotation from QuPath is displayed as a mask ROI in OMERO.iviewer (the yellow ROI in the screenshot below). Masks cannot be edited in OMERO.iviewer at the moment, but they can be viewed. The mask, when selected displays a blue bounding box around the Annotation on the image.



Contribute

Changes to the documentation or the materials should be made directly in the [omero-guide-qupath repository](#).

3.2.6 TrackMate

[TrackMate](#) is Fiji plugin for single-particle tracking. We demonstrate how to use TrackMate and OMERO using both the User Interface and the API.

Contents

Analyze OMERO timelapse images using the TrackMate User Interface

In this example we open in Fiji an image stored in an OMERO server and use [TrackMate](#) to analyze it.

Description

In this section, we show how to use TrackMate via its User Interface. The manual steps are essential to determine the suitable parameters to analyze the images. Note that when using the TrackMate User Interface, the generated tracks **cannot** be saved as ROIs into OMERO.server.

Setup

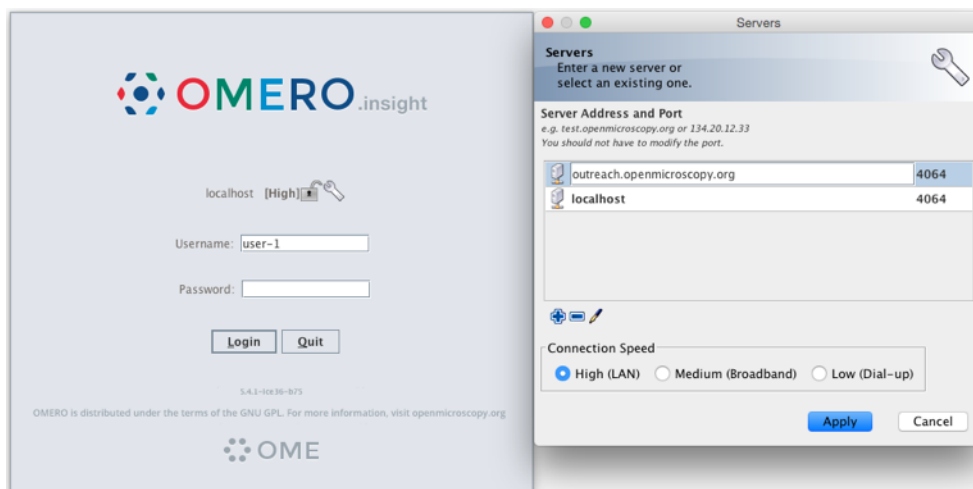
- Install Fiji on the local machine with the OMERO.insight-ij plugin. The installation instructions can be found in the [Fiji guide](#).

Resources

- We use an artificial track image <https://samples.fiji.sc/FakeTracks.tif>.

Step-by-Step

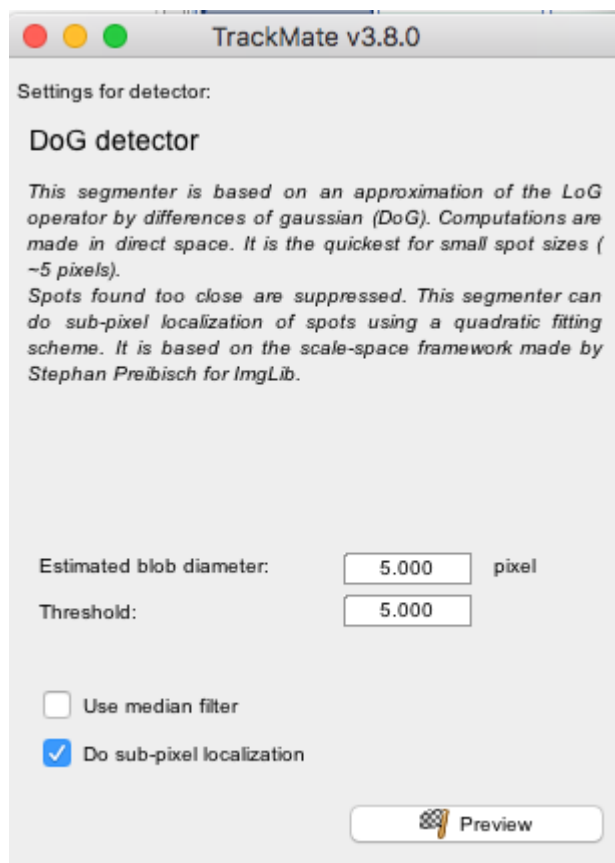
1. Launch Fiji.
2. Go to *Plugins > OMERO > Connect To OMERO*. This will show a login screen where you can enter the name of the server to connect to, the username and password. The OMERO plugin will allow you to browse your data in a similar manner to OMERO.web.
3. In the OMERO login dialog, click the wrench icon  and then add the server address in the dialog. By default, only “localhost” is listed. Click on the *plus* icon to add a new line to the list and type into the line the server address.
4. Click *Apply*.



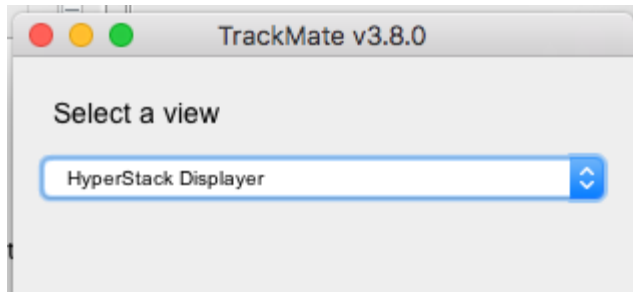
5. Enter your credentials and click *Login*.
6. Browse the Dataset where your image to be tracked is located, e.g. *artificial-trackmate*.
7. Double-click on the *FakeTracks.tif* Image to open it in Fiji.
 1. Make sure it is opened using the Hyperstack viewer. This option depends on what you did last time when using your Fiji.
 2. In case you are not sure, Open Plugins > Bio-Formats > Bio-Formats importer and then select any image on your local computer to open it in Fiji. In the popup dialog, in the top left dropdown menu.



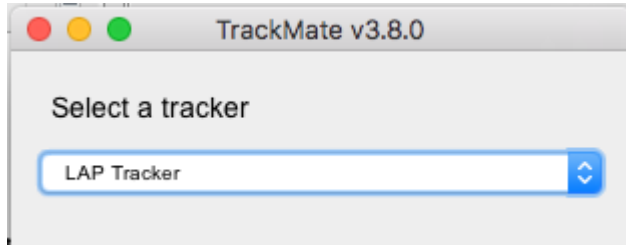
3. Select Hyperstack and click OK. Then, close and re-open the *FakeTracks.tif* Image from OMERO.
8. Go to Plugins > Tracking > TrackMate.
9. Click Next in the first dialog that pops up.
10. A new dialog pops up indicating to select a detector. Select the DoG detector. Click Next.
 1. Set the Estimated blob diameter to 5.0
 2. Set the Threshold to 5.0



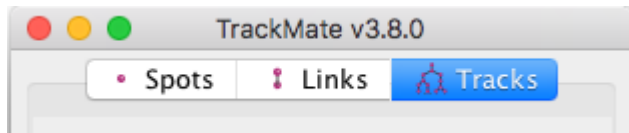
11. Click the Preview button to find spots. 4 spots should be found.
12. Click the Next button several times until you get to the Select a view dialog.
13. Select the HyperStack Displayer view



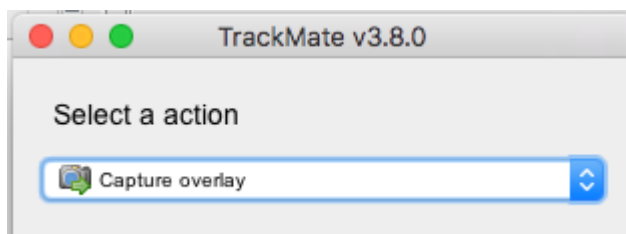
14. Click the Next button twice until you get to the Select a tracker window. Select the LAP Tracker.



15. Click the Next button several times until a dialog with three tabs pops up
16. Select the Tracks tab to display the Tracks.



17. Click Next until you get to Select an action dialog.
 1. Select the option Capture overlay
 2. Click Execute.



18. Click OK in the following dialog, leaving the defaults (the whole stack will be captured).
19. New image appears. Select it. This new Image can then be imported as an OME-TIFF using Plugins > OMERO > Save Image(s) to OMERO. The tracks will be part of the generated OME-TIFF and the timepoints will be captured as z-sections.

Analyze OMERO timelapse images using the TrackMate API

In this example we open an image stored in an OMERO server and use the [TrackMate](#) API to analyze it. One of the advantages of this approach over the User interface workflow is that the generated track **can** be saved as OMERO ROIs.

Description

First, we will show how to use the TrackMate API and the Scripting editor of Fiji.

We will show:

- How to connect to OMERO using the JAVA API.
- How to retrieve an image.
- How to open the image using Bio-Formats.
- How to create a TrackMate model using its API.
- How to save the tracks as polygon ROIs in OMERO.

Setup

- Install Fiji on the local machine with the OMERO.insight-ij plugin, version 5.5.10 or higher. The installation instructions can be found at [here](#).

Resources

- We will use a timelapse image available at [DV/iain/438CTR](#)
- Script: Groovy script for tracking timelapse images - `tracking.groovy`.

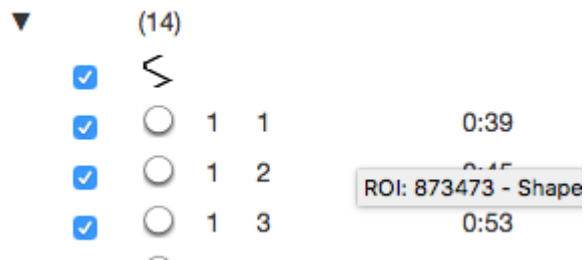
Step-by-Step

In this section, we go through the steps required to analyze the data. The script used in this document is `tracking.groovy`. The script follows similar steps than the ones used via the User Interface, see *[Analyze OMERO timelapse images using the TrackMate User Interface](#)*.

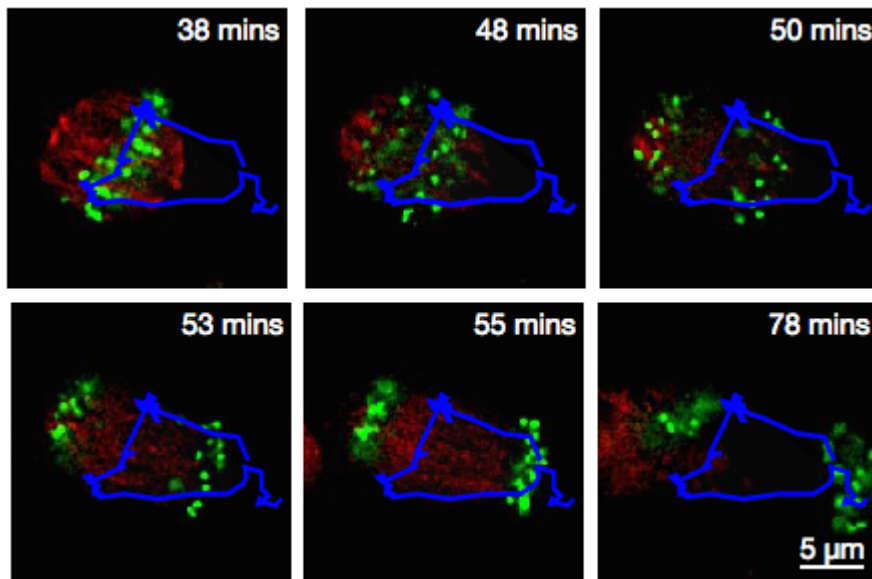
One advantage of the scripting approach is that we can save the generated tracks as ROIs in an OMERO server.

1. Launch Fiji.
2. Open File > New > Script....
3. Select Groovy as a language.
4. Copy the content of `tracking.groovy` in the text area.
5. A dialog will pop up. Enter the credentials to connect to the server and select an Image. If you are not using websockets i.e. no wss in front of the host name, the port value needs to be changed to 4064.
6. Click Run.
7. Go back to OMERO.web to visualize the tracks. Double-click on the image in OMERO.web to open it in OMERO.iviewer.

8. Click on the ROI tab and observe that you now have ROIs under which there are Shapes. Each ROI is a collection of shapes. The ROI corresponds to a track in Trackmate. There is always one polyline shape in each ROI which represents the track. The other, elliptical shapes in the same ROI represent the tracked spots.



9. Play the timelapse video in OMERO.iviewer.
10. Go to the Info tab, and in the Open with: line click on OMERO.figure. In OMERO.figure, add the Tracks and ellipses to the panel by selecting the appropriate ROIs in the Labels tab of OMERO.figure.



Script's description

Import packages needed:

```
import java.awt.Color
import java.util.ArrayList

import omero.gateway.Gateway
import omero.gateway.LoginCredentials
import omero.gateway.SecurityContext
import omero.gateway.facility.BrowseFacility
import omero.gateway.facility.ROIFacility
import omero.gateway.model.EllipseData
import omero.gateway.model.PolylineData
import omero.gateway.model.ROIData
import omero.log.SimpleLogger
```

(continues on next page)

(continued from previous page)

```

import omero.model.PolylineI
import static omero.rtypes.rstring

import ij.IJ

import fiji.plugin.trackmate.Spot
import fiji.plugin.trackmate.Settings
import fiji.plugin.trackmate.Model
import fiji.plugin.trackmate.SelectionModel
import fiji.plugin.trackmate.TrackMate
import fiji.plugin.trackmate.detection.DetectorKeys
import fiji.plugin.trackmate.detection.DogDetectorFactory
import fiji.plugin.trackmate.tracking.sparselap.SparseLAPTrackerFactory
import fiji.plugin.trackmate.tracking.LAPUtils
import fiji.plugin.trackmate.visualization.hyperstack.HyperStackDisplayer
import fiji.plugin.trackmate.features.spot.SpotContrastAndSNRAnalyzerFactory
import fiji.plugin.trackmate.features.spot.SpotIntensityAnalyzerFactory
import fiji.plugin.trackmate.features.track.TrackSpeedStatisticsAnalyzer

```

Connect to the server. It is also important to close the connection again to clear up potential resources held on the server. This is done in the `disconnect` method:

```

def connect_to_omero() {
    "Connect to OMERO"

    credentials = new LoginCredentials()
    credentials.getServer().setHostname(HOST)
    credentials.getUser().setUsername(USERNAME.trim())
    credentials.getUser().setPassword(PASSWORD.trim())
    simpleLogger = new SimpleLogger()
    gateway = new Gateway(simpleLogger)
    gateway.connect(credentials)
    return gateway
}

def disconnect(gateway) {
    gateway.disconnect()
}

```

Load the image from the server:

```

def get_image(gateway, image_id) {
    "Retrieve the image"
    browse = gateway.getFacility(BrowseFacility)
    user = gateway.getLoggedInUser()
    ctx = new SecurityContext(user.getGroupId())
    return browse.getImage(ctx, image_id)
}

```

Read the binary data using Bio-Formats:

```
def open_image_plus(host, port, username, password, group_id, image_id) {
    "Open the image using the Bio-Formats Importer"

    StringBuilder options = new StringBuilder()
    options.append("location=[OMERO] open=[omero:server=")
    options.append(host)
    options.append("\nuser=")
    options.append(username.trim())
    options.append("\nport=")
    options.append(port)
    options.append("\npass=")
    options.append(password.trim())
    options.append("\ngroupID=")
    options.append(group_id)
    options.append("\niid=")
    options.append(image_id)
    options.append("] ")
    options.append("windowless=true view=Hyperstack ")
    IJ.runPlugIn("loci.plugins.LociImporter", options.toString())
}
```

Create a tracking model using the TrackMate API:

```
def create_tracker(imp) {
    "Create the trackmate model for the specified ImagePlus object"
    // Instantiate model object
    model = new Model()

    // Prepare settings object
    settings = new Settings()
    settings.setFrom(imp)
    // Configure detector
    settings.detectorFactory = new DogDetectorFactory()
    settings.detectorSettings.put(DetectorKeys.KEY_DO_SUBPIXEL_LOCALIZATION, true)
    settings.detectorSettings.put(DetectorKeys.KEY_RADIUS, new Double(2.5))
    settings.detectorSettings.put(DetectorKeys.KEY_TARGET_CHANNEL, 1)
    settings.detectorSettings.put(DetectorKeys.KEY_THRESHOLD, new Double(5.0))
    settings.detectorSettings.put(DetectorKeys.KEY_DO_MEDIAN_FILTERING, false)
    // Configure tracker
    settings.trackerFactory = new SparseLAPTrackerFactory()
    settings.trackerSettings = LAPUtils.getDefaultLAPSettingsMap()
    settings.trackerSettings['LINKING_MAX_DISTANCE'] = new Double(10.0)
    settings.trackerSettings['GAP_CLOSING_MAX_DISTANCE'] = new Double(10.0)
    settings.trackerSettings['MAX_FRAME_GAP'] = 3

    // Add the analyzers for some spot features
    settings.addSpotAnalyzerFactory(new SpotIntensityAnalyzerFactory())
    settings.addSpotAnalyzerFactory(new SpotContrastAndSNRAnalyzerFactory())

    // Add an analyzer for some track features, such as the track mean speed.
    settings.addTrackAnalyzer(new TrackSpeedStatisticsAnalyzer())
    settings.initialSpotFilterValue = 1
}
```

(continues on next page)

(continued from previous page)

```

// Instantiate trackmate
trackmate = new TrackMate(model, settings)
ok = trackmate.checkInput()
if (!ok) {
    print(str(trackmate.getErrorMessage()))
    return null
}

ok = trackmate.process()
if (!ok) {
    print(str(trackmate.getErrorMessage()))
    return null
}
// Display the results on top of the image
selectionModel = new SelectionModel(model)
displayer = new HyperStackDisplayer(model, selectionModel, imp)
displayer.render()
displayer.refresh()
// The feature model, that stores edge and track features.
fm = model.getFeatureModel()
space_units = model.getSpaceUnits()
time_units = model.getTimeUnits()
return model
}

```

Convert the tracks into OMERO ROIs:

```

def convert_tracks(model, dx, dy) {
    "Convert the tracks into OMERO objects"
    rois = new ArrayList()
    tracks = model.getTrackModel().trackIDs(true)
    tracks.each() { track_id ->
        track = model.getTrackModel().trackSpots(track_id)
        roi = new ROIData()
        rois.add(roi)
        points = ""
        track.each() { spot ->
            sid = spot.ID()
            // Fetch spot features directly from spot.
            x = spot.getFeature('POSITION_X')/dx
            y = spot.getFeature('POSITION_Y')/dy
            r = spot.getFeature('RADIUS')
            z = spot.getFeature('POSITION_Z')
            t = spot.getFeature('FRAME')
            // Save spot as Point in OMERO
            ellipse = new EllipseData(x, y, r, r)
            ellipse.setZ((int) z)
            ellipse.setT((int) t)
            // set trackmate track ID and spot ID for later

```

(continues on next page)

(continued from previous page)

```

        ellipse.setText(track_id+':'+sid)
        // set a default color
        settings = ellipse.getShapeSettings()
        settings.setStroke(Color.RED)
        roi.addShapeData(ellipse)
        points = points + x + ',' + y + ' '
    }
    // Save the track
    points = points.trim()
    polyline = new PolylineI()
    polyline.setPoints(rstring(points))
    pl = new PolylineData(polyline)
    // set a default color
    settings = pl.getShapeSettings()
    settings.setStroke(Color.YELLOW)
    roi.addShapeData(pl)
}
return rois
}

```

Save the converted tracks back to the OMERO server:

```

def save_rois(gateway, rois, image_id) {
    roi_facility = gateway.getFacility(ROIFacility)
    user = gateway.getLoggedInUser()
    ctx = new SecurityContext(user.getGroupId())
    results = roi_facility.saveROIs(ctx, image_id, user.getId(), rois)
}

```

In order to use the methods implemented above in a proper standalone script, **Wrap it all up:**

```

gateway = connect_to_omero()
exp = gateway.getLoggedInUser()
group_id = exp.getGroupId()

image = get_image(gateway, image_id)
open_image_plus(HOST, PORT, USERNAME, PASSWORD, group_id, image_id)
imp = IJ.getImage()
dx = imp.getCalibration().pixelWidth
dy = imp.getCalibration().pixelHeight
trackmate_model = create_tracker(imp)
if (trackmate_model == null) {
    print("unable to create the trackmate model")
} else {
    omero_rois = convert_tracks(trackmate_model, dx, dy)
    save_rois(gateway, omero_rois, image_id)
    print("done")
}
disconnect(gateway)

```

Contribute

Changes to the documentation or the materials should be made directly in the [omero-guide-trackmate repository](#).

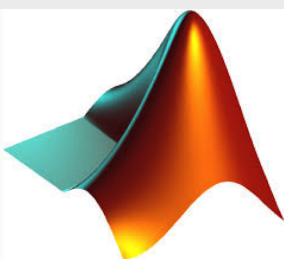
3.3 OMERO Application Programming Interfaces

The OMERO Application Programming Interface (API) allows clients to be written in Java, Python, R, C++ or MATLAB. In this section we show how to set up the various libraries and use a combination of Jupyter notebook and scripts to demonstrate how to use the APIs.

*Java Gateway*

The section demonstrates how to install and use the Java API. Jupyter notebooks are used to analyze data and save results back the server.

See docs for OMERO Java Gateway at <https://docs.openmicroscopy.org/latest/omero/developers/Java.html>.

*MATLAB*

This section shows how to install and use the OMERO MATLAB toolbox. The exercises demonstrate how to analyze data and store the results back to server.

*Python API*

The section demonstrates how to install and use the Python API. Jupyter notebooks are used to analyze data and save results back the server.

*R Gateway*

This section shows how to install and use the R Gateway to analyze data. Jupyter notebooks are available to demonstrate some of the functionalities available.

3.3.1 Java Gateway

The section demonstrates how to install and use the Java API. Jupyter notebooks are used to analyze data and save results back the server.

See docs for OMERO Java Gateway at <https://docs.openmicroscopy.org/latest/omero/developers/Java.html>.

Contents:

3.3.2 MATLAB

This section shows how to install and use the OMERO MATLAB toolbox. The exercises demonstrate how to analyze data and store the results back to server.

Contents:

Analyze OMERO data using MATLAB Online

This workflow was contributed by the community, see <https://github.com/mathworks/Open-Microscopy-Data-MATLAB>. It allows to work with OMERO data using MATLAB Online.

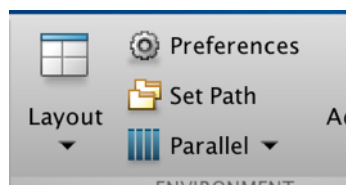
This means that users do not need to download and install MATLAB locally as MATLAB is running in your browser. You will just need a valid MathWorks account. An easy to understand environment with Jupyter Notebooks is provided. Follow the steps on the <https://uk.mathworks.com/matlabcentral/fileexchange/156034-open-microscopy-data-matlab>.

Install the OMERO.matlab toolbox

In this section, we show how to install the OMERO.matlab toolbox

Setup

- Download the [toolbox](#).
- Make sure that the OMERO.matlab toolbox is on the MATLAB path. To add it to the path, you can
 - Launch MATLAB.
 - Under the *HOME* tab, click on *Set Path* (middle of the top task bar).



- A *Set Path* dialog pops up.
- Click on the button *Add with Subfolders...*
- Select the OMERO.matlab toolbox, Click *Open*.
- Close the *Set Path* dialog, you can either save the path for future use or not.

It is also recommended to install the [Image Processing toolbox](#) is only necessary for the image analysis. This is a convenient toolbox for analysis purpose. You do not need to install that toolbox to integrate OMERO and MATLAB.

Analyze OMERO data using MATLAB

MATLAB is a powerful programming platform. We show here how you can analyze data stored in OMERO using MATLAB.

We will use <https://docs.openmicroscopy.org/latest/omero/developers/Matlab.html> as a reference.

Description

Here we demonstrate how to analyze a batch of images associated with the paper [Subdiffraction imaging of centrosomes reveals higher-order organizational features of pericentriolar material](#).

We will show:

- How to connect to OMERO using MATLAB.
- How to load data (dataset, channels information, binary data).
- How to analyze images. The channel's name will be used to determine the channel to analyze.
- How to save the generated ROIs to OMERO.
- How to save the results stored in a CSV file locally back to the OMERO.server as a FileAnnotation.
- How to convert the CSV file into an OMERO.table.

Setup

Please read *[Install the OMERO.matlab toolbox](#)* first.

Resources

We will use:

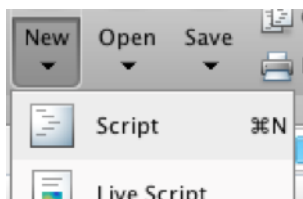
- Images from IDR [idr0021](#).

For convenience, the IDR data have been imported into the training OMERO.server. This is **only** because we **cannot** save results back to IDR which is a read-only OMERO.server.

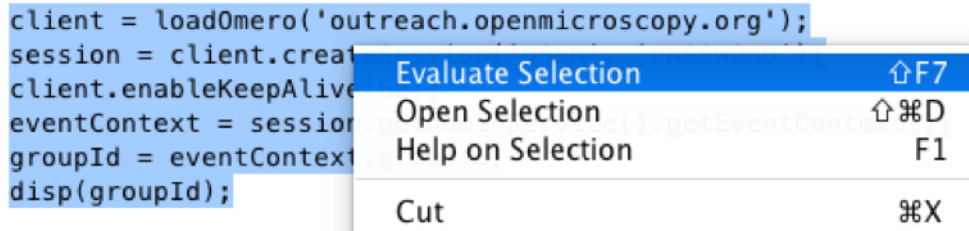
Step-by-Step

The script used in this document is `idr0021_steps.m`.

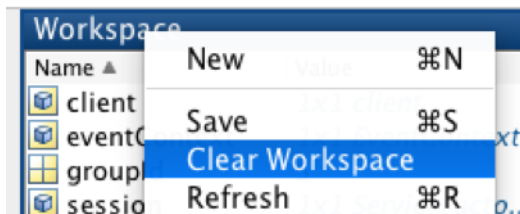
1. In the *EDITOR* tab create a new script:



2. Copy the code for the exercises from `idr0021_steps.m`
3. Paste it into the new file and save the script under whatever name you like. **DO NOT RUN** the whole script.
4. To follow along the exercises only select the code block of each exercise and run it with “Evaluate Selection”:



5. Later exercises cannot be run unless the previous exercises have been executed successfully.
6. If **you get stuck**, right-click on the *Workspace* tab, clear the workspace and start again from the beginning:



Exercise 1

Objectives: Connect to OMERO and print out your group ID.

Steps:

- Replace the USER and PASSWORD placeholders with your assigned credentials.
- Select the code block of **Exercise 1**
- Run it with “Evaluate Selection”.

Exercise 2

Objectives: Load dataset and list the images contained in the dataset.

Steps:

- In OMERO.web find the dataset ‘matlab-dataset’ (in Project ‘matlab-project’)
- Copy its ID
- In the MATLAB code replace DATASET_ID with this ID
- Run the code block.

Exercise 3

Objectives: Read metadata; in particular find out which protein is the target in the images by looking through the image's map annotations (key-value pairs). It is the same protein for all four sample images.

Steps:

- Select one image from the dataset
- Load the map annotation linked to the image
- Select the entry whose key is 'Antibody Target'

Exercise 4

Objectives: Find out in which channels the target protein is stained.

Steps:

- Iterate through the dataset
- For each Image
 - Find the channel's name using the LogicalChannel
 - Determine the index of the channel whose name matches the value found in the previous exercise

Exercise 5

Objectives: Perform a simple image segmentation on one image and display the result.

Steps:

- Iterate through the dataset
- Analyze the image whose name is *siControl_N20_Cep215_I_20110411_Mon-1509_0_SIR_PRJ.dv*
- Retrieve the plane with $z=0$, $t=0$, $c=channel-1$. Indexes start at 0 in OMERO.
- Determine the mean, the standard deviation.

Exercise 6

Objectives: Perform the image segmentation on the whole dataset and save the results as ROIs and CSV file. The CSV file is saved as a FileAnnotation

Exercise 7

Objectives: Save the results as OMERO.table. This shows how to convert the CSV file into an OMERO.table

Steps:

- Run the code
- Go back to OMERO.web
- Select an image from the evaluated dataset
- Expand the *Tables* harmonica. You should see the results there.

- Double-click on the thumbnail of the image and inspect the ROIs in OMERO.iviewer.
- Note: You can also use OMERO.parade on the OMERO.table data created in this manner. As OMERO.parade works only on Projects, in OMERO.web
 - Create a new Project
 - Put the analyzed Dataset into that Project
 - Attach the OMERO.table created in **Exercise 7** to the Project
 - Now you can use OMERO.parade on the Project

3.3.3 Python API

The section demonstrates how to install and use the Python API. Jupyter notebooks are used to analyze data and save results back the server.

Contents

Install omero-py

In this section, we show how to install omero-py together with additional packages for image analysis in a [Conda](#) environment.

We will use the Python API to access data stored in an OMERO server. The image analysis is typically done with packages like [scikit-image](#) or [dask](#).

If you want to use the Python API only for interacting with the OMERO server, like scripting management / import workflows you can just install the omero-py package by itself, see [omero-py](#).

Setup

For using the examples and notebooks of this guide we recommend using Conda (Option 1). Conda manages programming environments in a manner similar to [virtualenv](#).

Alternatively you can use [repo2docker](#) to build and run a Docker image locally (Option 2). This Docker image will provide the Conda environment and Jupyter notebooks with some image analysis workflows.

When the installation is done, you should be ready to use the OMERO Python API, see [Getting started with the OMERO Python API](#).

Option 1

Install omero-py and additional packages for image analysis (see `environment.yml`) via Conda.

- Install [Miniconda](#) if necessary.
- If you do not have a local copy of the [omero-guide-python repository](#), first clone the repository:

```
$ git clone https://github.com/ome/omero-guide-python.git
```

- Go into the directory:

```
$ cd omero-guide-python
```

- Install `omero-py` and other packages useful for demonstration purposes in order to connect to an OMERO server using an installation file:

```
$ mamba env create -f binder/environment.yml
```

- Activate the environment:

```
$ conda activate omero-guide-python
```

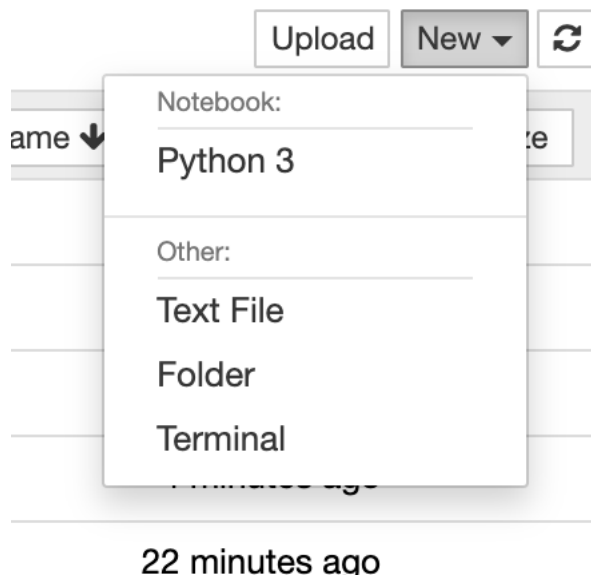
Option 2

Create a local Docker Image using `repo2docker`, see `README.md`:

```
$ pip install jupyter-repo2docker
$ git clone https://github.com/ome/omero-guide-python.git
$ cd omero-guide-python
$ repo2docker .
```

When the Image is ready:

- Copy the URL displayed in the terminal in your favorite browser
- Click the New button on the right-hand side of the window
- Select Terminal



- A Terminal will open in a new Tab
- A Conda environment has already been created when the Docker Image was built
- To list all the Conda environments, run:

```
$ conda env list
```

- The environment with the OMERO Python bindings and a few other libraries is named `notebook`, activate it:

```
$ conda activate notebook
```

Getting started with the OMERO Python API

Description

We will show:

- Connect to a server.
- Load images.
- Run a simple FRAP analysis measuring the intensity within a pre-existing ellipse ROI in a named Channel.
- Save the generated mean intensities and link them to the image(s).
- Save the generated plot on the server.

Setup

We recommend to use a Conda environment to install the OMERO Python bindings. Please read first *Install omero-py*.

For the FRAP analysis you need a fluorescence time-lapse image, available at <https://downloads.openmicroscopy.org/images/DV/will/FRAP/>.

The bleached spot has to be marked with an ellipse. Make sure that the ellipse ROI spans the whole timelapse.

Step-by-Step

In this section, we go through the steps required to analyze the data. The script used in this document is `simple_frap.py`.

It is also available as the ‘SimpleFRAP’ Jupyter notebook in the notebooks section.

Modules and methods which need to be imported:

```
from omero.gateway import BlitzGateway, MapAnnotationWrapper
from omero.model import EllipseI
from PIL import Image
import numpy as np
import matplotlib
from matplotlib import pyplot as plt
from getpass import getpass
```

Connect to the server. It is also important to close the connection again to clear up potential resources held on the server. This is done in the `disconnect` method.

```
def connect(hostname, username, password):
    """
    Connect to an OMERO server
    :param hostname: Host name
    :param username: User
```

(continues on next page)

(continued from previous page)

```

:param password: Password
:return: Connected BlitzGateway
"""
conn = BlitzGateway(username, password,
                    host=hostname, secure=True)
conn.connect()
conn.c.enableKeepAlive(60)
return conn

def disconnect(conn):
    """
    Disconnect from an OMERO server
    :param conn: The BlitzGateway
    """
    conn.close()

```

Load the image:

```
img = conn.getObject("Image", image_id)
```

Get relevant channel index:

```

def get_channel_index(image, label):
    """
    Get the channel index of a specific channel
    :param image: The image
    :param label: The channel name
    :return: The channel index (None if not found)
    """
    labels = image.getChannelLabels()
    if label in labels:
        idx = labels.index(label)
        return idx
    return None

```

Get Ellipse ROI:

```

def get_ellipse_roi(conn, image):
    """
    Get the first ellipse ROI found in the image
    :param conn: The BlitzGateway
    :param image: The Image
    :return: The shape ID of the first ellipse ROI found
    """
    roi_service = conn.getRoiService()
    result = roi_service.findByImage(image.getId(), None)
    shape_id = None
    for roi in result.rois:

```

(continues on next page)

(continued from previous page)

```

    for s in roi.copyShapes():
        if type(s) == EllipseI:
            shape_id = s.id.val
    return shape_id

```

Get intensity values:

```

def get_mean_intensities(conn, image, the_c, shape_id):
    """
    Get the mean pixel intensities of an roi in a time series image
    :param conn: The BlitzGateway
    :param image: The image
    :param the_c: The channel index
    :param shape_id: The ROI shape id
    :return: List of mean intensity values (one for each timepoint)
    """
    roi_service = conn.getRoiService()
    the_z = 0
    size_t = image.getSizeT()
    meanvalues = []
    for t in range(size_t):
        stats = roi_service.getShapeStatsRestricted([shape_id],
                                                    the_z, t, [the_c])
        meanvalues.append(stats[0].mean[0])
    return meanvalues

```

Plot the data using matplotlib:

```

def plot(values, plot_filename):
    """
    Create a simple plot of the given values
    and saves it.
    :param values: The values
    :param plot_filename: The file name
    :return: Nothing
    """
    matplotlib.use('Agg')
    fig = plt.figure()
    plt.subplot(111)
    plt.plot(values)
    fig.canvas.draw()
    fig.savefig(plot_filename)
    pil_img = Image.open(plot_filename)
    pil_img.show()

```

Save the results as Map annotation:

```
def save_values(conn, image, values):
    """
    Attach the values as map annotation to the image
    :param conn: The BlitzGateway
    :param image: The image
    :param values: The values
    :return: Nothing
    """
    namespace = "demo.simple_frap_data"
    key_value_data = [[str(t), str(value)] for t, value in enumerate(values)]
    map_ann = MapAnnotationWrapper(conn)
    map_ann.setNs(namespace)
    map_ann.setValue(key_value_data)
    map_ann.save()
    image.linkAnnotation(map_ann)
```

Save the plot:

```
def save_plot(conn, image, plot_filename):
    """
    Save the plot to OMERO
    :param conn: The BlitzGateway
    :param image: The image
    :param plot_filename: The path to the plot image
    :return: Nothing
    """
    pil_img = Image.open(plot_filename)
    np_array = np.asarray(pil_img)
    red = np_array[:, :, 0]
    green = np_array[:, :, 1]
    blue = np_array[:, :, 2]
    plane_gen = iter([red, green, blue])
    conn.createImageFromNumpySeq(plane_gen, plot_filename, sizeC=3,
                                dataset=image.getParent())
```

In order to use the methods implemented above in a proper standalone script: Wrap it all up in an `analyse` method and call it from `main`:

```
def analyse(conn, image_id, channel_name):
    # Step 2 - Load image
    img = conn.getObject("Image", image_id)
    # -
    ci = get_channel_index(img, channel_name)
    shape_id = get_ellipse_roi(conn, img)
    values = get_mean_intensities(conn, img, ci, shape_id)
    plot(values, 'plot.png')
    save_values(conn, img, values)
    save_plot(conn, img, 'plot.png')
```

(continues on next page)

(continued from previous page)

```
def main():
    try:
        hostname = input("Host [wss://workshop.openmicroscopy.org/omero-ws]: \
                        ") or "wss://workshop.openmicroscopy.org/omero-ws"
        username = input("Username [trainer-1]: ") or "trainer-1"
        password = getpass("Password: ")
        image_id = int(input("Image ID [28662]: ") or 28662)
        channel = input("Channel name [528.0]: ") or "528.0"
        conn = connect(hostname, username, password)
        analyse(conn, image_id, channel)
    finally:
        if conn:
            disconnect(conn)

if __name__ == "__main__":
    main()
```

Further Reading

How to turn a script into an [OMERO script](#) which runs on the server and can be directly launched via the web interface, see *[How to convert a client-side script into a server-side script](#)*.

This `simple_frap.py` example as server-side OMERO.script: `simple_frap_server.py`.

How to convert a client-side script into a server-side script

In this section, we show how to convert a Python script into a script that can be run server-side. We recommend that you read [OMERO.scripts guide](#).

Description

We will make a simple Hello World script as client-side Python script and show how to convert it into a server-side script. The script will:

- Connect to the server
- Load images in a dataset

Setup

We recommend to use a Conda environment to install the OMERO Python bindings. Please read first *[Install omero-py](#)*.

Step-by-step

The scripts used in this document are `hello_world.py` and `hello_world_server.py`.

It is also available as the ‘OMEROHelloWorld’ Jupyter notebook in the notebooks section.

Client-side script

Let’s first start by writing a **client-side** script named `hello_world.py`.

Connect to the server:

```
def connect(hostname, username, password):
    """
    Connect to an OMERO server
    :param hostname: Host name
    :param username: User
    :param password: Password
    :return: Connected BlitzGateway
    """
    conn = BlitzGateway(username, password,
                        host=hostname, secure=True)
    conn.connect()
    conn.c.enableKeepAlive(60)
    return conn
```

Load the images in the dataset:

```
def load_images(conn, dataset_id):
    """
    Load the images in the specified dataset
    :param conn: The BlitzGateway
    :param dataset_id: The dataset's id
    :return: The Images or None
    """
    dataset = conn.getObject("Dataset", dataset_id)
    images = []
    for image in dataset.listChildren():
        images.append(image)
    if len(images) == 0:
        return None

    for image in images:
        print("---- Processing image", image.id)
    return images
```

Collect the script parameters:

```
host = input("Host [wss://workshop.openmicroscopy.org/omero-ws]: ") or 'wss://
workshop.openmicroscopy.org/omero-ws' # noqa
```

(continues on next page)

(continued from previous page)

```
username = input("Username [trainer-1]: ") or 'trainer-1'
password = getpass("Password: ")
dataset_id = input("Dataset ID [2391]: ") or '2391'
```

In order to use the methods implemented above in a proper standalone script: Wrap it all up and call them from main:

```
if __name__ == "__main__":
    try:
        # Collect parameters
        host = input("Host [wss://workshop.openmicroscopy.org/omero-ws]: ") or 'wss://
↳workshop.openmicroscopy.org/omero-ws' # noqa
        username = input("Username [trainer-1]: ") or 'trainer-1'
        password = getpass("Password: ")
        dataset_id = input("Dataset ID [2391]: ") or '2391'

        # Connect to the server
        conn = connect(host, username, password)

        # Load the images container in the specified dataset
        load_images(conn, dataset_id)
    finally:
        conn.close()
    print("done")
```

Server-side script

Now let's see how to convert the script above into a **server-side** script.

The first step is to declare the parameters, the UI components will be built automatically from it. This script only needs to collect the dataset ID:

```
# Define the script name and description, and a single 'required' parameter
client = scripts.client(
    'Hello World.py',
    """
    This script does connect to OMERO.
    """,
    scripts.Long("datasetId", optional=False),

    authors=["OME Team", "OME Team"],
    institutions=["University of Dundee"],
    contact="ome-users@lists.openmicroscopy.org.uk",
)
```

Process the arguments:

```
script_params = {}
for key in client.getInputKeys():
    if client.getInput(key):
        script_params[key] = client.getInput(key, unwrap=True)
```

(continues on next page)

(continued from previous page)

```
dataset_id = script_params["datasetId"]
```

Access the data using the Python gateway:

```
conn = BlitzGateway(client_obj=client)
```

We can then use the same method that the one in the client-side script to load the images:

```
def load_images(conn, dataset_id):
    """
    Load the images in the specified dataset
    :param conn: The BlitzGateway
    :param dataset_id: The dataset's id
    :return: The Images or None
    """
    dataset = conn.getObject("Dataset", dataset_id)
    images = []
    if dataset is None:
        return None
    for image in dataset.listChildren():
        images.append(image)
    if len(images) == 0:
        return None

    for image in images:
        print("---- Processing image", image.id)
    return images
```

In order to use the methods implemented above in a proper standalone script and return the output to the users, wrap it all up and call them from main:

```
if __name__ == "__main__":
    # Start declaration
    # Define the script name and description, and a single 'required' parameter
    client = scripts.client(
        'Hello World.py',
        """
        This script does connect to Omero.
        """,
        scripts.Long("datasetId", optional=False),

        authors=["OME Team", "OME Team"],
        institutions=["University of Dundee"],
        contact="ome-users@lists.openmicroscopy.org.uk",
    )
    # Start script
    try:
        # process the list of arguments
        script_params = {}
```

(continues on next page)

(continued from previous page)

```
for key in client.getInputKeys():
    if client.getInput(key):
        script_params[key] = client.getInput(key, unwrap=True)

dataset_id = script_params["datasetId"]

# wrap client to use the Blitz Gateway
conn = BlitzGateway(client_obj=client)

# load the images
images = load_images(conn, dataset_id)

# return output to the user
if images is None:
    message = "No images found"
else:
    message = "Returned %s images" % len(images)
    # return first image:
    client.setOutput("Image", robject(images[0]._obj))

client.setOutput("Message", rstring(message))
# end output
finally:
    client.closeSession()
```

Analyze data stored in a public S3 repository in parallel

Description

We will show how to use `dask` to analyze an IDR image stored in a public S3 repository

We will show:

- How to connect to IDR to retrieve the image metadata.
- How to load the Zarr binary stored in a public repository.
- How to run a segmentation on each plane in parallel.

Setup

We recommend to use a Conda environment to install the OMERO Python bindings. Please read first [Install omero-py](#).

Step-by-Step

In this section, we go through the steps required to analyze the data. The script used in this document is `public_s3_segmentation_parallel.py`.

Load the image and reate a dask array from the Zarr storage format:

```
def load_binary_from_s3(id, resolution='4'):
    endpoint_url = 'https://uk1s3.embassy.ebi.ac.uk/'
    root = 'idr/zarr/v0.1/%s.zarr/%s/' % (id, resolution)
    return da.from_zarr(endpoint_url + root)
```

Define the analysis function:

```
def analyze(t, c, z):
    plane = data[t, c, z, :, :]
    smoothed_image = dask_image.ndfilters.gaussian_filter(plane, sigma=[1, 1])
    threshold_value = 0.33 * da.max(smoothed_image).compute()
    threshold_image = smoothed_image > threshold_value
    label_image, num_labels = dask_image.ndmeasure.label(threshold_image)
    name = 't:%s, c: %s, z:%s' % (t, c, z)
    print("Plane coordinates: %s" % name)
    ref = 't_%s_c_%s_z_%s' % (t, c, z)
    return label_image, ref
```

Make our function lazy using `dask.delayed`. It records what we want to compute as a task into a graph that we will run later in parallel:

```
def prepare_call(dimensions):
    middle_z = dimensions[2] // 2
    middle_t = dimensions[0] // 2
    range_t = 2
    range_z = 2
    number_c = dimensions[1]
    lazy_results = []
    for t in range(middle_t - range_t, middle_t + range_t):
        for z in range(middle_z - range_z, middle_z + range_z):
            for c in range(number_c):
                lazy_result = dask.delayed(analyze)(t, c, z)
                lazy_results.append(lazy_result)
    return lazy_results
```

We are now ready to run in parallel using the default number of workers see [Configure dask.compute](#):

```
def compute(lazy_results):
    return dask.compute(*lazy_results)

# Save the first 5 results on disk
def save_results(results):
    print("Saving locally the first 5 results as png")
    for r, name in results[:5]:
        array = numpy.asarray(r)
        value = "image_%.png" % name
        plt.imsave(value, array)
```

In order to use the methods implemented above in a proper standalone script: **Wrap it all up** in main:

```
def main():
    # Collect image ID
    image_id = "4007801"

    global data
    data = load_binary_from_s3(image_id)
    print("Dask array: %s" % data)

    lazy_results = prepare_call(data.shape)

    start = time.time()
    results = compute(lazy_results)
    elapsed = time.time() - start
    print('Compute time (in seconds): %s' % elapsed)
    save_results(results)
    print('done')

if __name__ == "__main__":
    main()
```

Contribute

Changes to the documentation or the materials should be made directly in the [omero-guide-python repository](#).

3.3.4 R Gateway

This section shows how to install and use the R Gateway to analyze data. Jupyter notebooks are available to demonstrate some of the functionalities available.

Contents

Installing rOMERO-gateway

rOMERO-gateway will use rJava to communicate with an OMERO server.

Install in a Conda environment

In this section, we show how to install rOMERO-gateway and additional packages for image analysis (see `environment.yml` and `postBuild`) in a [Conda](#) environment.

- Install [Miniconda](#) if necessary.
- If you do not have a local copy of the [omero-guide-r repository](#), first clone the repository:

```
$ git clone https://github.com/ome/omero-guide-r.git
```

- Go into the directory:

```
$ cd omero-guide-r
```

- Create a programming environment using Conda:

```
$ conda create -n rgateway
```

- Install dependencies and some packages used in the notebooks using an installation file:

```
$ conda env update -n rgateway --file binder/environment.yml
```

- Run a bash script to install rOMERO-gateway and the Java dependencies:

```
$ bash binder/postBuild
```

- Activate the environment:

```
$ conda activate rgateway
```

Install in RStudio

See *Installing and using romero.gateway package in RStudio*

Installing and using romero.gateway package in RStudio

Linux

This example uses Debian 10. On other distributions the commands and packages might be called slightly differently.

As *root* user install the system dependencies: `apt install r-base openjdk-11-jdk libssl-dev libcurl4-openssl-dev`

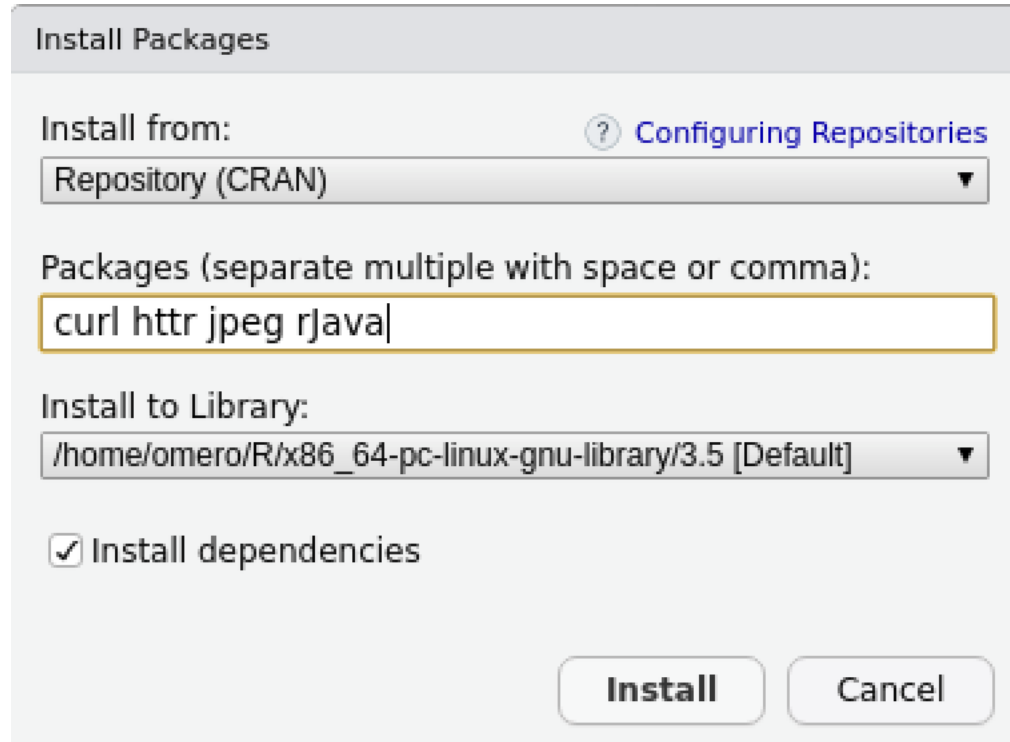
Then you need to find out where exactly the JDK was installed: `ls -l /etc/alternatives/ | grep java`

Lookout for 'openjdk' to find the location.

Configure Java for R (still using *root* user):

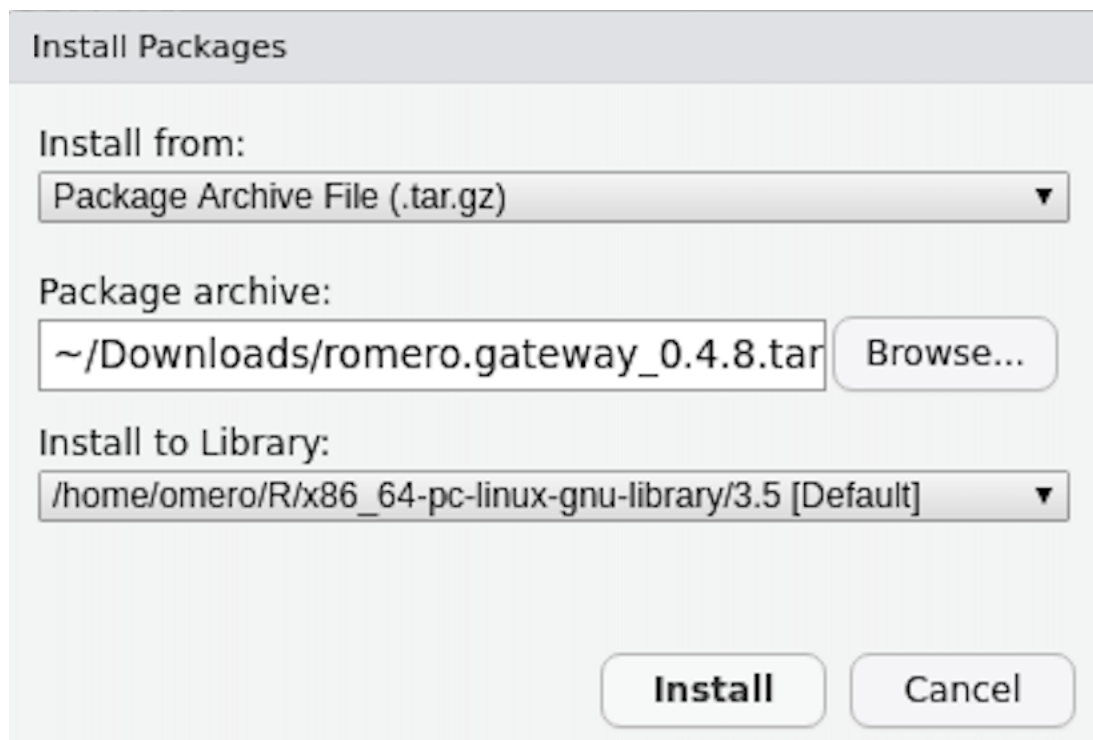
```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64 # Whatever your path to openjdk is
R CMD javareconf
```

As normal user install and start up RStudio. In RStudio install the dependencies from CRAN needed for the `romero.gateway` package:

The image shows the 'Install Packages' dialog box in RStudio. It has a title bar 'Install Packages'. Below the title bar, there is a section 'Install from:' with a dropdown menu set to 'Repository (CRAN)' and a link '? Configuring Repositories'. Below that is a text input field for 'Packages (separate multiple with space or comma):' containing the text 'curl httr jpeg rJava'. Below the input field is a section 'Install to Library:' with a dropdown menu set to '/home/omero/R/x86_64-pc-linux-gnu-library/3.5 [Default]'. At the bottom left, there is a checked checkbox labeled 'Install dependencies'. At the bottom right, there are two buttons: 'Install' and 'Cancel'.

Alternative: Run the following in the console: `install.packages(c("jpeg", "curl", "httr", "rJava"))`

Download the latest `romero.gateway` release tar.gz from [Github](#). Install it in RStudio:



Install Packages

Install from:
Package Archive File (.tar.gz) ▼

Package archive:
~/Downloads/romero.gateway_0.4.8.tar Browse...

Install to Library:
/home/omero/R/x86_64-pc-linux-gnu-library/3.5 [Default] ▼

Install Cancel

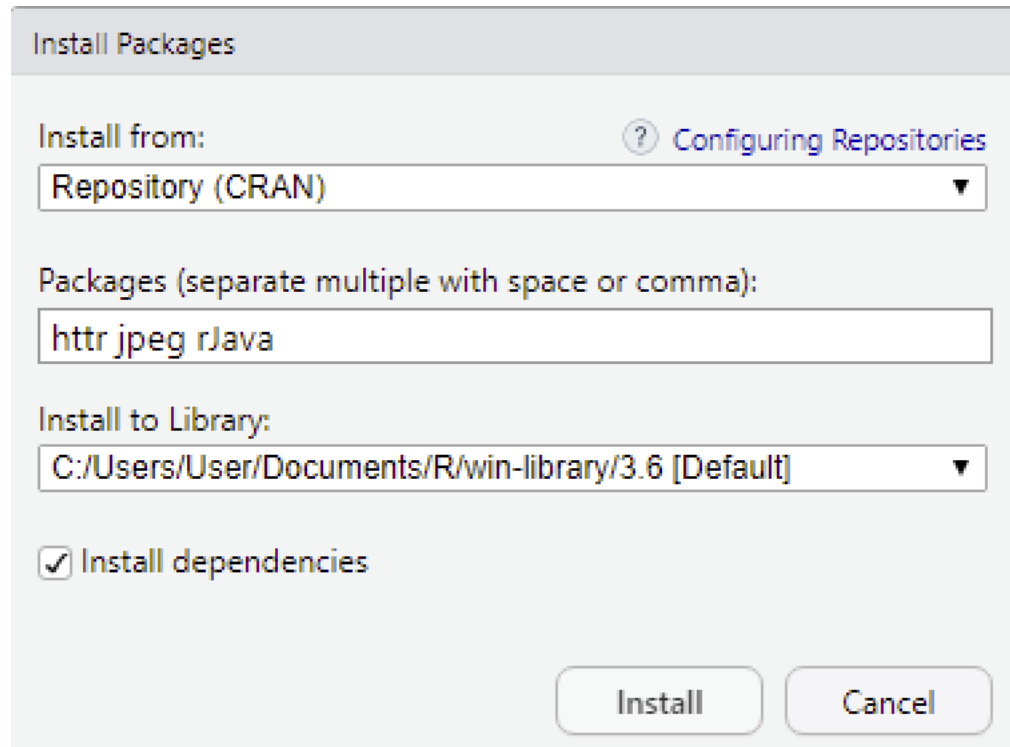
Alternative: Run the following in the console: `install.packages("~/Downloads/romero.gateway_0.4.8.tar.gz", repos = NULL, type = "source")`

Windows

This example uses Windows 10, other Windows version should be similar.

Install a JDK, for example [AdoptOpenJDK](#). Make sure to tick the checkbox to set the *JAVA_HOME* and *PATH* environment variables!

Install the dependencies `httr`, `jpeg` and `rJava` from CRAN:



Install Packages

Install from: [? Configuring Repositories](#)

Repository (CRAN) ▼

Packages (separate multiple with space or comma):

httr jpeg rJava

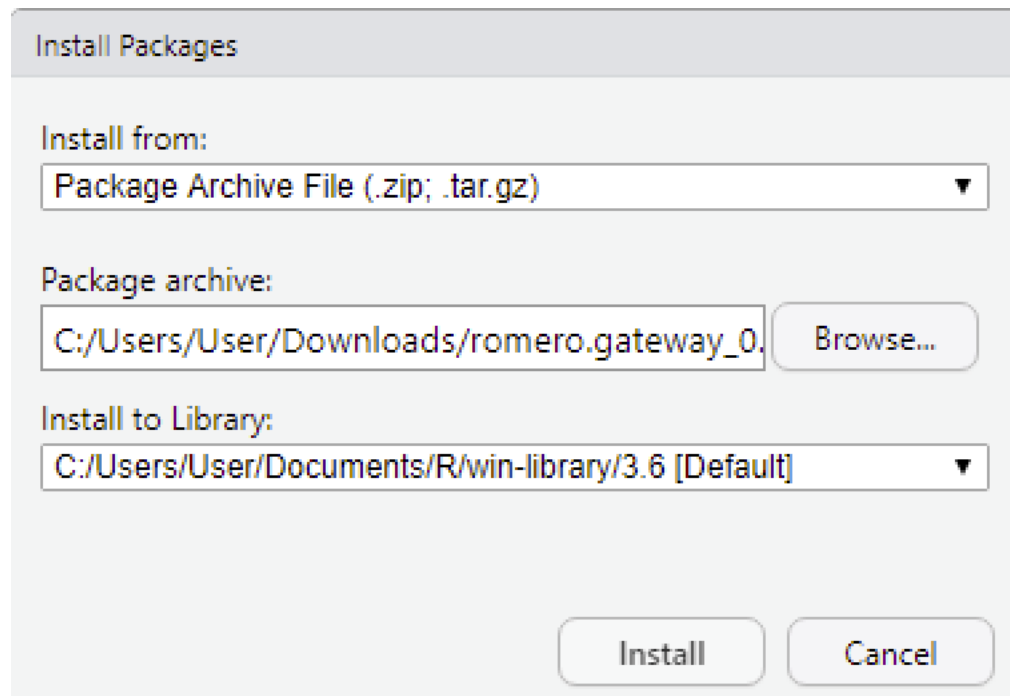
Install to Library:

C:/Users/User/Documents/R/win-library/3.6 [Default] ▼

☒ Install dependencies

Install Cancel

Download the latest `romero.gateway` release zip from [Github](#) and install it:



Install Packages

Install from:

Package Archive File (.zip; .tar.gz) ▼

Package archive:

C:/Users/User/Downloads/romero.gateway_0. Browse...

Install to Library:

C:/Users/User/Documents/R/win-library/3.6 [Default] ▼

Install Cancel

Analyse data using R (statistical package)

R is a free software environment for statistical computing and graphics. See <https://www.r-project.org/> for more information.

Description

Here we demonstrate how to connect to OMERO using R, load images and other objects from OMERO and perform image analysis followed up by statistical analysis in R. We use R as part of a Jupyter notebook. In the notebook we are going to use the `idr0021` data. These are super resolution microscopy images showing certain proteins around the centrioles. With these images the authors of the article [Subdiffraction imaging of centrosomes reveals higher-order organizational features of pericentriolar material](#) showed that the area around the centrioles has a specific structure. By measuring the shape and diameter of these proteins and comparing them to each other they could show that each protein builds a specific part of this structure, see [Figure 1](#). Instead of manually measuring each centriole ring like the authors did, we are going to try to do this in an automated way in order to show that there is a significant difference between these proteins and create a plot similar to the one seen on [Figure 1](#).

We will show:

- How to connect to OMERO using R.
- How to load images from OMERO to R.
- How to perform segmentation of objects in an image in R.
- How to calculate properties of the segmented objects (e.g. diameter) in R.
- How to save the properties of the objects as an R dataframe.
- How to create and save ellipse representations of the ROI for each segmented object onto the original image in OMERO.
- How to save the properties of the objects as attachments in OMERO.
- How to load images in a whole Dataset or Project from OMERO and perform the steps above on all images.
- How to perform statistical analysis on the generated data in R and save results back to OMERO.

Setup

In order to be able to connect to OMERO from within R we need the `rOMERO-gateway` and `rJava` packages.

Note: Everything we are going to do there can be done in Rstudio as well, if the `rOMERO-gateway` package and its dependencies are available on your system.

See [Installing rOMERO-gateway](#).

Resources

We will use data from the Image Data Resource (IDR).

- IDR data `idr0021`.
- Notebook `idr0021_Segmentation.ipynb`.

For convenience, the IDR data have been imported into the training OMERO.server. This is only because we cannot save results back to IDR which is a read-only OMERO.server.

Step-by-step

All the steps are described in detail inside the Jupyter notebook. In you are running locally or in mybinder, go to:

- In the *notebooks* folder, select the notebook *idr0021_Segmentation.ipynb*.

Note: We are going to show the image segmentation and feature calculation on a small subset of the idr0021 data. The full analysis of the idr0021 project has already been done and the result saved as ‘Summary from R’ table. We are going to load this table in order to do the statistical analysis at the end of the notebook.

Here, we give a digest of the steps inside the notebook: The notebook steps will:

1. Fetch the images from OMERO using rOMERO-gateway.
2. Perform image segmentation to identify the protein rings around the centrioles using EBImage in R.
3. Calculate the properties (features) of the identified objects (shape, diameter, etc.) using EBImage.
4. Store these features in R as an R dataframe.
5. The segmentation and feature calculation is a rather costly process. You don’t want to do this all over again. Therefore we save the identified objects (ROIs, region of interests) and the properties back to OMERO using rOMERO-gateway.
6. Perform statistical analysis on that data in R.
7. At the end of the notebook a plot is created which is similar to the one shown in [Figure 1](#) of the article. One dataset in particular has extreme outliers. OMERO.parade <https://github.com/ome/omero-parade> can be used to find the images that cause the outliers.
8. Note: OMERO.parade uses the *Summary from R* table that is attached to the project in order to provide enhanced filtering and plotting features.

Contribute

Changes to the documentation or the materials should be made directly in the [omero-guide-r repository](#).

3.4 Add-ons for OMERO

OMERO is a flexible platform offering various extension points e.g. Command Line Interface plugins or OMERO.insight plugins. Those extensions are not installed by default when deploying OMERO. They require some additional setup steps.

3.4.1 Insight extensions

OMERO.mde

OMERO.mde is an extension of OMERO.importer to get an overview of available metadata provided by the selected image container and annotate images at import step by standardized key-value input forms. Changes of metadata provided by image will be saved as key-value pairs.

OMERO.mde

OMERO.mde is an extension of OMERO.importer to get an overview of available metadata provided by the selected image container and annotate images at import step by standardized key-value input forms. Changes of metadata provided by image will be saved as key-value pairs.

Contents:

Using OMERO.mde to edit ome metadata

Description

By using OMERO.mde you can annotate your images with standardized key-values at the import step.

We will show:

- How to enable OMERO.mde feature for the OMERO.importer
- How to edit ome metadata
- How to modify the object tree

Setup

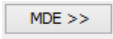
Install the OMERO.insight client as described in [import-desktop-client.html#setup](#)

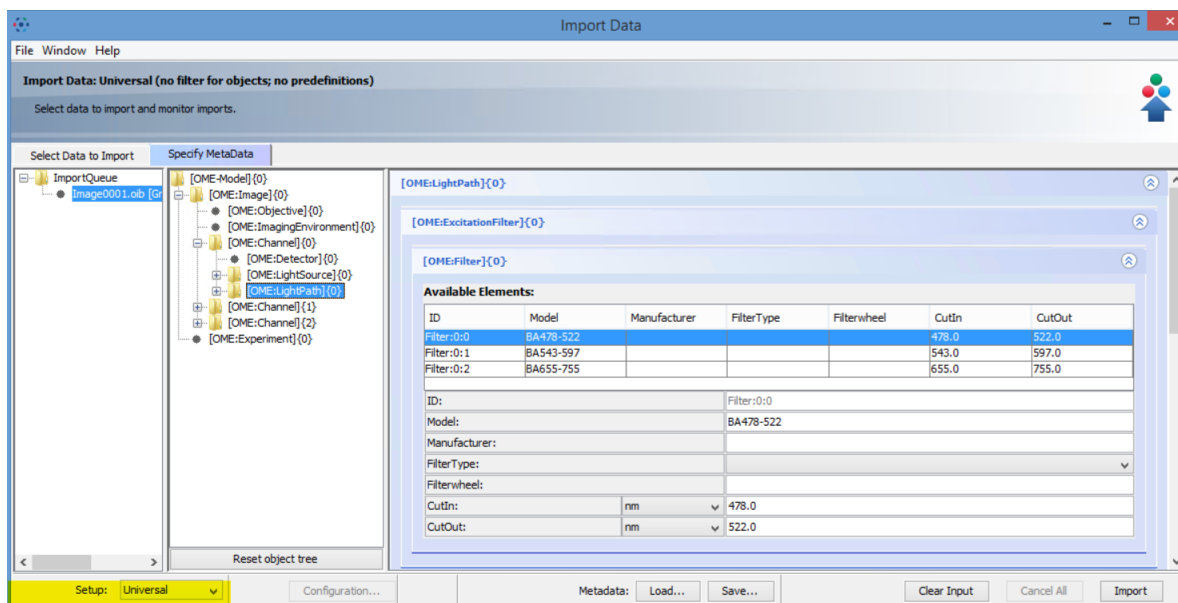
Go to `OMERO.insight/config/` and edit `container.xml` and `containerImporter.xml`. To enable the OMERO.mde set boolean to true in:

```
<!-- mde on/off -->  
  
<entry name="omero.client.import.mde.enabled" type="boolean">false</entry>
```

and save the files.

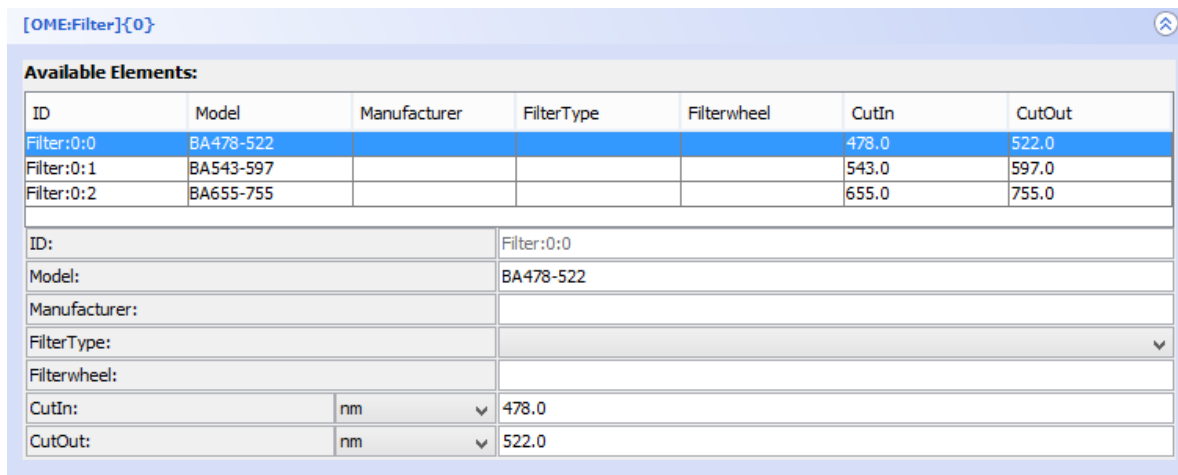
Step-by-Step

1. Start OMERO.importer and login.
2. Select your files for import and add it to the queue.
3. Click on the MDE Button  at the right bottom pane.



On the left pane you find the import queue, on the middle pane the object tree and on the right pane the input forms.

4. In the Specify MetaData window, select a file to show available metadata stored in the file container. After reading the metadata of the file by Bio-Formats you can find in the middle pane the object tree with standard OME objects like Image, Experiment, Detector etc. The properties of the selected object in the object tree and its subtree are displayed on the right pane as key-value pairs. For instrument objects there is a table with all objects of the same type as available Elements as defined in the file container.



Now you can editing the properties of the object.

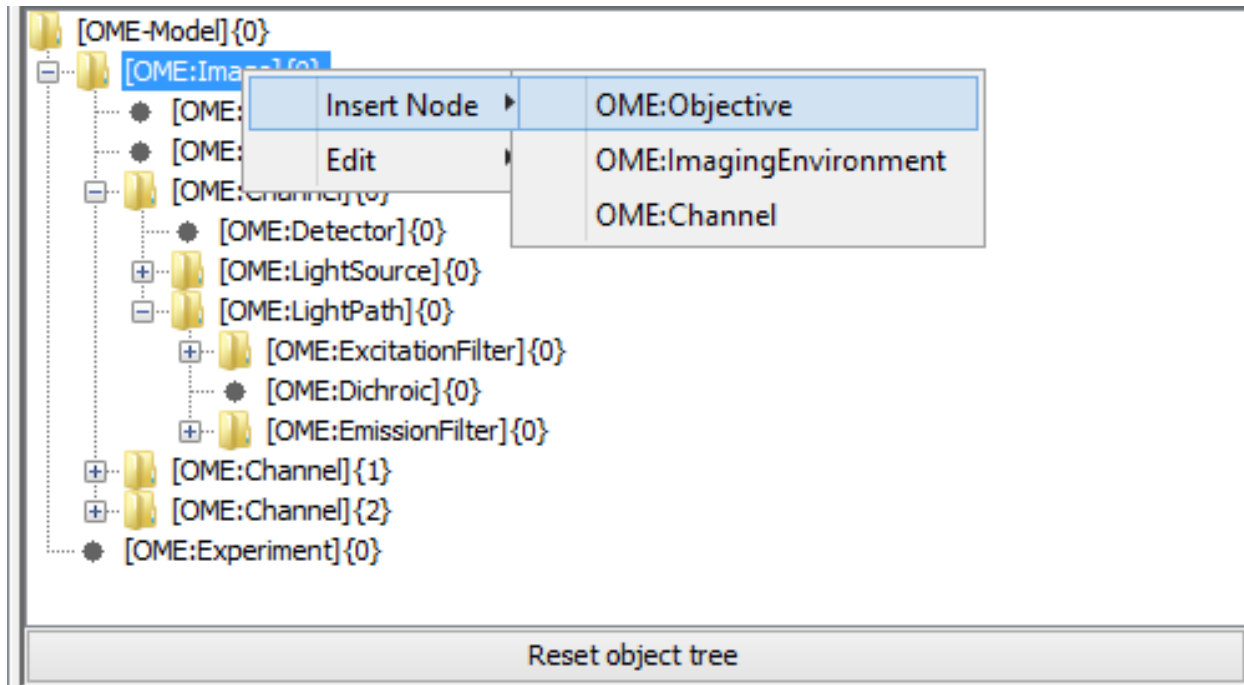
You can:

- change the unit
- add or change the value or select another value from the list of possible values
- replace all properties of an instrument object by selecting an element from the available Element list

Modifying the object tree

Add an additional object in the object tree:

1. Click with the right mouse button on the object for which a child object should be created.
2. Select an object from list under Insert Node to insert an object.



Copy and paste of a subtree:

1. Click with the right mouse button on the root of the subtree you want to copy.
2. Select **Edit > Copy** to copy this subtree.
3. Select the object where you want to insert the subtree (you can only insert the subtree where this type of rootobject is allowed as a child) and select by right mouse click **Edit > Paste**. Only the structure of the subtree will be inserted, not the values of the properties.

To reset to original object tree based on file information press the **Reset object tree** button. To reset all object and properties changes press the **Clear Input** button.

If you edit the properties and object for a folder, all files in the folder inherit the changes.

All changes will be add as key-value annotation to the file after import.

Key-Value Pairs 1	
Add Key Add Value	
Added by: [REDACTED]	
[OME-Model]{0}#[OME:Image]{0}#[OME:Channel]{0}#[OME:LightPath]{0}#[OME:ExcitationFilter]{0}#[OME:Filter]{0} ID	Filter:0:1
[OME-Model]{0}#[OME:Image]{0}#[OME:Channel]{0}#[OME:LightPath]{0}#[OME:ExcitationFilter]{0}#[OME:Filter]{0} Model	BA543-597
[OME-Model]{0}#[OME:Image]{0}#[OME:Channel]{0}#[OME:LightPath]{0}#[OME:ExcitationFilter]{0}#[OME:Filter]{0} CutIn	543.0 nm
[OME-Model]{0}#[OME:Image]{0}#[OME:Channel]{0}#[OME:LightPath]{0}#[OME:ExcitationFilter]{0}#[OME:Filter]{0} CutOut	597.0 nm

Customize OMERO.mde

Description

You can configure and customize the OMERO.mde by defining and configuring elements in a `mdeConfiguration.xml` file. A template of this file can be found in the `config/` directory of the installed OMERO.insight installation or at [mdeConfiguration.xml](#).

Structure of `mdeConfiguration.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
  <MDEConfiguration>
    <MDEPredefinitions> <!-- Predefinitions for properties of objects group by
↳ setups--!>
      <SetupPre Name="SetupName">
        <ObjectPre>
          <TagData.../>
          ...
        </ObjectPre>
        ...
      </SetupPre>
      ...
    </MDEPredefinitions>
    <MDEObjects>
      <Definitions> <!-- Definitions of objects, properties and hierarchy--!>
        <ObjectDef Type="ObjectName">
          <TagData.../>
          ...
          <Parents.../>
        </ObjectDef>
        ...
      </Definitions>
      <Configurations> <!--Definition of setups and configuration of objects and
↳ properties belonging to the setup --!>
        <SetupConf Name="SetupName">
          <ObjectConf Type="ObjectName">
            <TagDataProp.../>
            ...
          </ObjectConf>
          ...
        </SetupConf>
        ...
      </Configurations>
    </MDEObjects>
  </MDEConfiguration>
```

The location of the configuration file should be in the `config/` folder or in `<userHome>/omero/`. Please specify the location you will use by editing `container.xml` and `containerImporter.xml`. To use `config/` as location:

```
<!-- mde config file location (. for in config dir; omero for local user omero dir) -->
  <entry name="omero.client.import.mde.path">.</entry>
```

or for `<userHome>/omero/` as location:

```
<!-- mde config file location (. for in config dir; omero for local user omero dir) -->
<entry name="omero.client.import.mde.path">omero</entry>
```

and save the files. Here we will use `config/` as location for `mdeConfiguration.xml`. Save the configuration file under `config/` and restart the OMERO.mde to load the custom configurations.

You can customize the OMERO.mde in following point:

Customize user input form of OMERO.mde

Description

You can generate a filter view of the sets of objects. Also you can hide properties for objects and change the default unit. You can mark fields as required to restrict your input forms to objects with required metadata (OMERO.insight version $\geq 5.5.15$). We will show in this section how to create a new setup and how to generate customized input forms of available objects in this setup.

Step-by-Step

1. Open the `mdeConfiguration.xml` file under `config/`.
2. Insert a new `<SetupConf>` element under `<Configuration>` and specify the name for the setup (here `MyCustomSetup`):

```
<MDEObjects>
  <Configurations>
    <SetupConf Name="MyCustomSetup"/>
  </Configurations>
</MDEObjects>
```

3. Define now the object elements `<ObjectConf>` that should be enabled (OME standard objects) or visible/choosable (other). Please pay attention of the hierarchy (the parent of and object has also to be defined here). If an object is not mentioned, it is disabled and also all child objects (OME standard objects) or not visible (defined custom objects):

```
<MDEObjects>
  <Configurations>
    <SetupConf Name="MyCustomSetup">
      <ObjectConf Type="OME:Image"/>
      <ObjectConf Type="OME:Objective"/>
    </SetupConf>
  </Configurations>
</MDEObjects>
```

For this objects also add `<TagDataProp>` elements for the properties where you want to change the unit or you want to hide this property:

```
<MDEObjects>
  <Configurations>
    <SetupConf Name="MyCustomSetup">
      <ObjectConf Type="OME:Image">
        <TagDataProp Name="Description" Unit="" Visible="false"/>
      </ObjectConf>
    </SetupConf>
  </Configurations>
</MDEObjects>
```

(continues on next page)

(continued from previous page)

```

    <ObjectConf Type="OME:Objective"/>
  </SetupConf>
</Configurations>
</MDEObjects>

```

4. Save the file and restart OMERO.importer to load the new configuration for OMERO.mde. Select your images or dataset for import. Switch to the OMERO.mde pane and select your setup to load your customize forms.

Setup: Universal

Import Data

File Window Help

Import Data: Universal (no filter for objects; no predefinitions)

Select data to import and monitor imports.

Select Data to Import Specify MetaData

ImportQueue

- 0016_Amp_med_GB

[OME-Model]{0}

- [OME:Image]{0}
 - [OME:Objective]{0}
 - [OME:ImagingEnvironment]{0}
 - [OME:Channel]{0}
 - [OME:Experiment]{0}

Reset object tree

[OME:Image]{0}

Name: 0051_Amphiglena_med_GB6JP_L_Q10_S1_3000

Description:

Acquisition Time: 2019-08-19 00:00:00

Dim X x Y: 2048 2048

Pixel Depth: uint8

Pixel Size (XY): μm 86084980668966 86084980668966

Dim Z x T x C: 1 1 3

Time Increment: ms

Stage Label (XY): reference fra...

Setup: Universal Configuration... Metadata: Load... Save... Clear Input Cancel All Import

Setup: MyCustomSetup

Import Data

File Window Help

Import Data: MyCustomSetup

Select data to import and monitor imports.

Select Data to Import Specify MetaData

ImportQueue

- 0016_Amp_med_GB

[OME-Model]{0}

- [OME:Image]{0}
 - [OME:Objective]{0}
 - [OME:ImagingEnvironment]{0}
 - [OME:Channel]{0}
 - [OME:Experiment]{0}

Reset object tree

[OME:Image]{0}

Name: 0051_Amphiglena_med_GB6JP_L_Q10_S1_3000

Acquisition Time: 2019-08-19 00:00:00

Dim X x Y: 2048 2048

Pixel Depth: uint8

Pixel Size (XY): μm 86084980668966 86084980668966

Dim Z x T x C: 1 1 3

Time Increment: ms

Stage Label (XY): reference fra...

[OME:Objective]{0}

Setup: MyCustomSetup Configuration... Metadata: Load... Save... Clear Input Cancel All Import

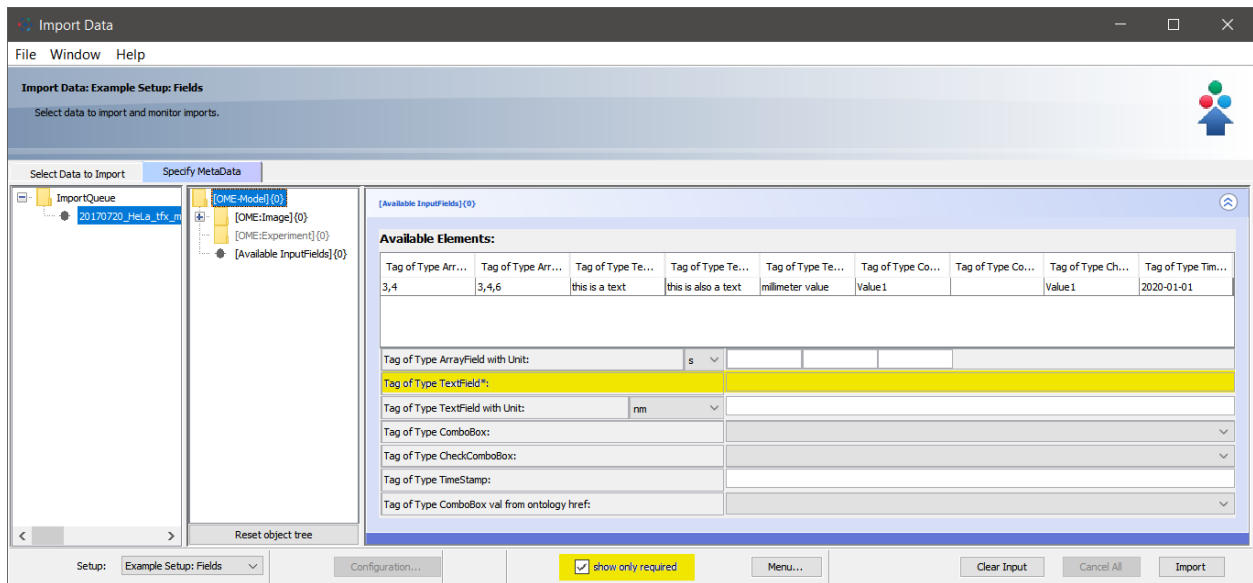
You can mark any field as required by adding `Required = "true"` as an attribute of `TagDataProp` (OMERO.insight version $\geq 5.5.15$). By selecting ☒ **show only required**, you can restrict the displayed objects of the selected setup to those that contain at least one required metadata field:

```

<MDEObjects>
  <Configurations>
    <SetupConf Name="Example Setup: Fields">
      <ObjectConf Type="OME:Image"/>
      <TagDataProp Name="Name" Unit="" Visible="true" Required="true"/>
      <ObjectConf Type="OME:Objective"/>
    </SetupConf>
  </Configurations>
</MDEObjects>

```

Setup: Example Setup: Fields



Add custom objects to OMERO.mde

Description

By using OMERO.mde you can create standardized key-value input forms with certain values and units. You can group key-values by creating an object module with the key-values as properties. We will show in this section how to create new objects for OMERO.mde and how you could create new key-values as properties of objects.

Step-by-Step

1. Open the `mdeConfiguration.xml` file under `config/`.
2. Insert a new `<ObjectDef>` element under `<Definitions>` and specify the name for the object (here `MyCustomObject`):

```

<MDEObjects>
  <Definitions>
    <ObjectDef Type="MyCustomObject"/>
  </Definitions>

```

(continues on next page)

(continued from previous page)

```
<Configurations.../>
</MDEObjects>
```

3. Define now the properties (key-values) for your new object. For every property you have to add an `<TagData>` element. See below section Type of input fields.
4. Define now in which hierarchy the object should be insert. For that define the parent element:

```
<MDEObjects>
  <Definitions>
    <ObjectDef Type="MyCustomObject">
      <TagData.../>
      ...
    </ObjectDef>
  </Definitions>
  <Configurations.../>
</MDEObjects>
```

At this point we have specified an insertable object. You can insert this object with right-click on the specified parent node. If you want this object will be automatically inserted to the object tree, you have to specify this also in the *SetupConf* section of the corresponding setup:

```
<MDEObjects>
  <Configurations>
    <SetupConf Name="MyCustomSetup"
      <ObjectConf Type="OME:Image" ...>
      <ObjectConf Type="OME:Objective"...>
      <ObjectConf Type="MyCustomObject" Insert="true" InsertPoint="OME:Image" ...>
    </SetupConf>
  </Configurations>
</MDEObjects>
```

Example

Specification like:

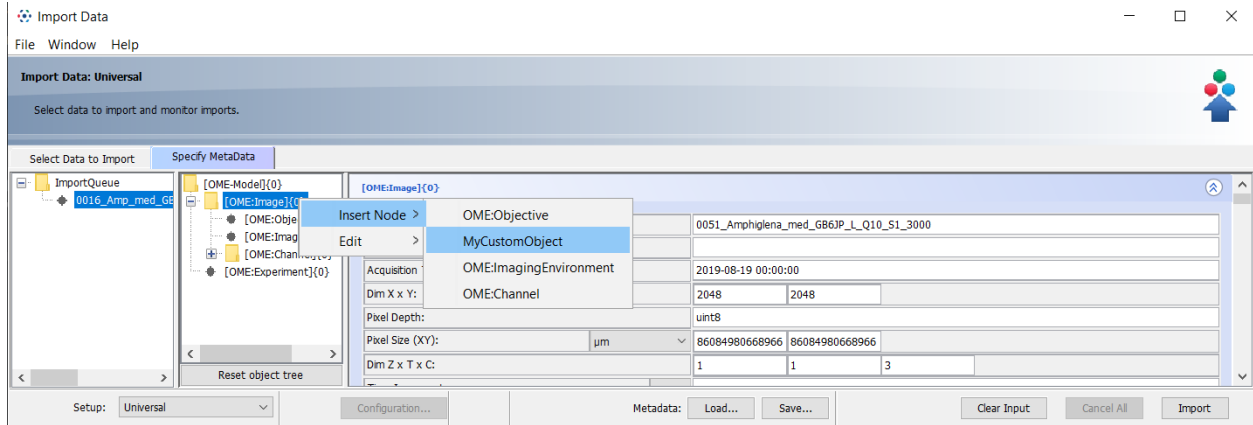
```
<MDEConfiguration>
  <MDEPredefinitions...>
  <MDEObjects>
    <Definitions>
      ...
      <ObjectDef Type="MyCustomObject">
        <TagData DefaultValues="" Name="ExampleKey_1" Type="TextField"
          Unit="" Value="" Visible="true" />
        <TagData DefaultValues="" Name="ExampleKey_2" Type="TextField"
          Unit="" Value="" Visible="true" />
        <Parents Values="OME:Image" />
      </ObjectDef>
    </Definitions>
  </MDEObjects>
</MDEConfiguration>
```

(continues on next page)

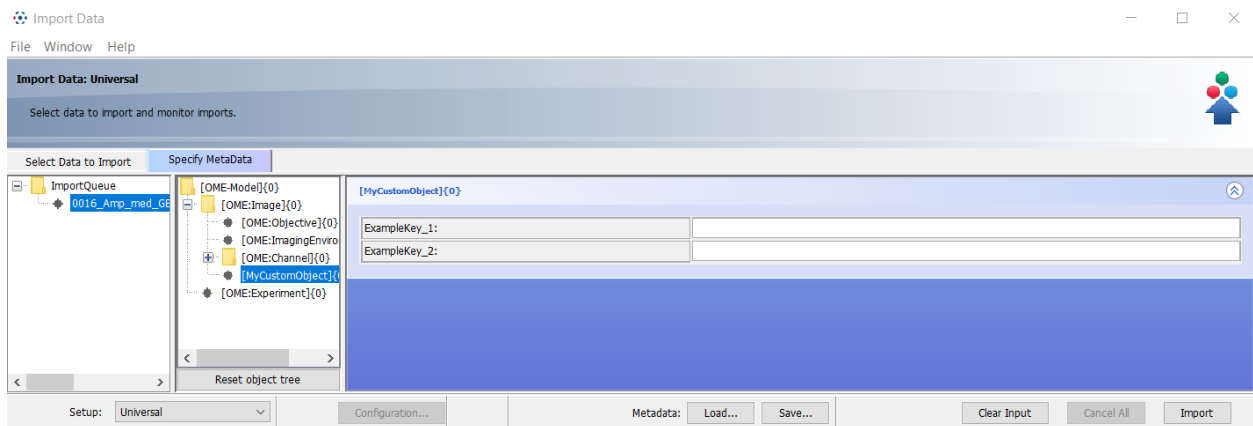
(continued from previous page)

```
</MDEObjects>
</MDEConfiguration>
```

will create an insertable object at OME:Image object.



that has input form for two specified key-value pairs



With the additional specification in *Configuration*:

```
<MDEConfiguration>
  <MDEPredefinitions...>
  <MDEObjects>
    <Definitions>
      ...
      <ObjectDef Type="MyCustomObject">
        <TagData DefaultValues="" Name="ExampleKey_1" Type="TextField"
          Unit="" Value="" Visible="true" />
        <TagData DefaultValues="" Name="ExampleKey_2" Type="TextField"
          Unit="" Value="" Visible="true" />
        <Parents Values="OME:Image" />
      </ObjectDef>
    </Definitions>
    <Configuration>
      <SetupConf Name="MyCustomSetup">
        <ObjectConf Type="OME:Image"...>
```

(continues on next page)

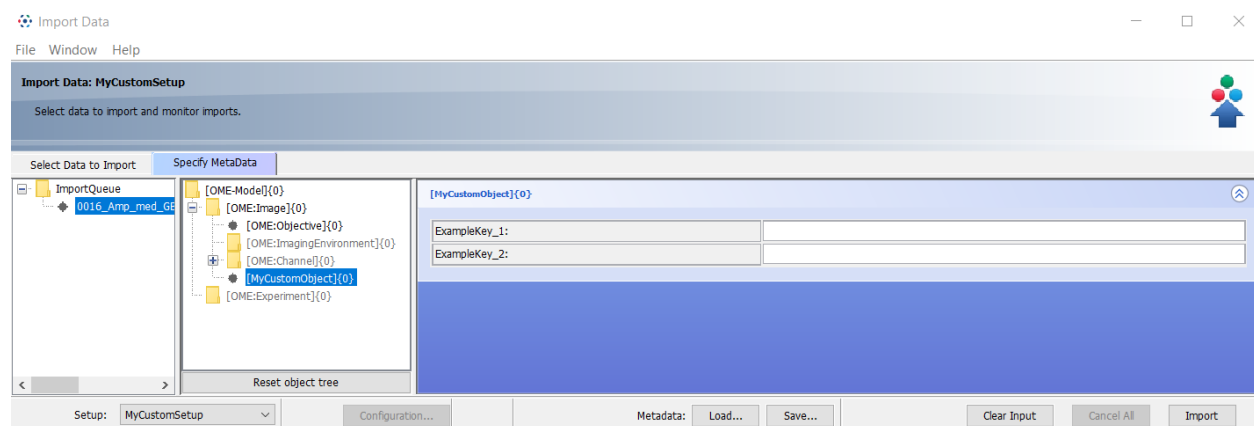
(continued from previous page)

```

    <ObjectConf Type="OME:Objective"/>
    <ObjectConf Type="MyCustomObject" Insert="true" InsertPoint="OME:Image"/>
  </SetupConf>
</Configurations>
</MDEObjects>
</MDEConfiguration>

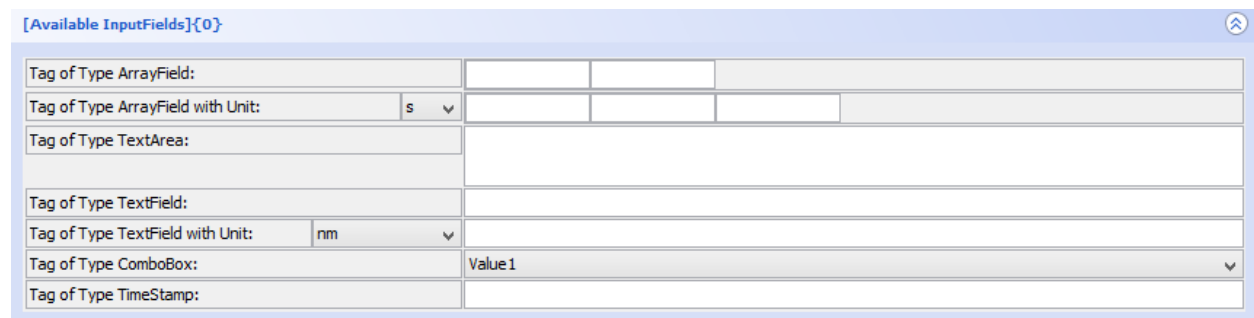
```

will lead into following object tree if you select the setup *MyCustomSetup*



Type of input fields

There are different editor input field types for the element `<TagData>`. You can find this example by using the example `mdeConfiguration.xml` and insert a *Available InputFields* object by right-clicking on OME-Model node or it is inserted automatically when you select the setup Example Setup:Fields.



You can specify the different types like:

TextField define like:

```

<TagData DefaultValues=""
  Name="Tag of Type TextField"
  Type="TextField"
  Unit=""
  Value=""
  Visible="true" />

```

TextField with unit define like:

```
<TagData DefaultValues=""
  Name="Tag of Type TextField with unit"
  Type="TextField"
  Unit="nm"
  Value=""
  Visible="true" />
```

TextArea define like:

```
<TagData DefaultValues=""
  Name="Tag of Type TextArea"
  Type="TextArea"
  Unit=""
  Value=""
  Visible="true" />
```

ArrayField define like (for an array of 2 elements):

```
<TagData DefaultValues="2"
  Name="Tag of Type ArrayField"
  Type="ArrayField"
  Unit=""
  Value=""
  Visible="true" />
```

ArrayField with unit define like (for an array of 3 elements):

```
<TagData DefaultValues="3"
  Name="Tag of Type ArrayField with unit"
  Type="ArrayField"
  Unit="s"
  Value=""
  Visible="true" />
```

ComboBox define like:

```
<TagData DefaultValues="Value1,Value2,Value3"
  Name="Tag of Type ComboBox"
  Type="ComboBox"
  Unit=""
  Value=""
  Visible="true" />
```

ComboBox with values from ontology (OMERO.insight version >= 5.5.15):

```
<TagData DefaultValues=""
  Name="Tag of Type ComboBox val from ontology href"
  Type="ComboBox"
  Unit=""
  Value=""
  Visible="true">
  <Ontology URL_restapi="http://data.bioontology.org" Acronym="BAO" ID_href=
  ↪ "http://www.bioassayontology.org/bao#BAO_0150008" />
</TagData>
```

CheckComboBox for multiple selection (OMERO.insight version >= 5.5.15):

```
<TagData DefaultValues="Value1,Value2,Value3"
  Name="Tag of Type CheckComboBox"
  Type="CheckComboBox"
  Unit=""
  Value="Value1"
  Visible="true" /
```

TimeStamp define like:

```
<TagData DefaultValues=""
  Name="Tag of Type TimeStamp"
  Type="TimeStamp"
  Unit=""
  Value=""
  Visible="true" />
```

Predefine object values in OMERO.mde

Description

By using OMERO.mde you can generate a set of predefined values for the available objects. You can predefine different values for same objects by group it in setups.

Step-by-Step

1. Open the `mdeConfiguration.xml` file under `<userHome>/omero/`.
2. Insert a new `<SetupPre>` element under `<MDEPredefinitions>` with the name of the setup for that you want to specify the values:

```
<MDEConfiguration>
  <MDEPredefinitions>
    <SetupPre Name="MyCustomSetup">
  </MDEPredefinitions>
</MDEConfiguration>
```

3. Insert under this `<SetupPre>` element an `<ObjectPre>` element with the name of the object for that you want to specify the values:

```
<MDEConfiguration>
  <MDEPredefinitions>
    <SetupPre Name="MyCustomSetup">
      <ObjectPre ID="" Type="MyCustomObject"/>
    </SetupPre>
  </MDEPredefinitions>
</MDEConfiguration>
```

4. Now you can copy - paste the object properties defined under `<ObjectDef>` for this object and then specify the Values.

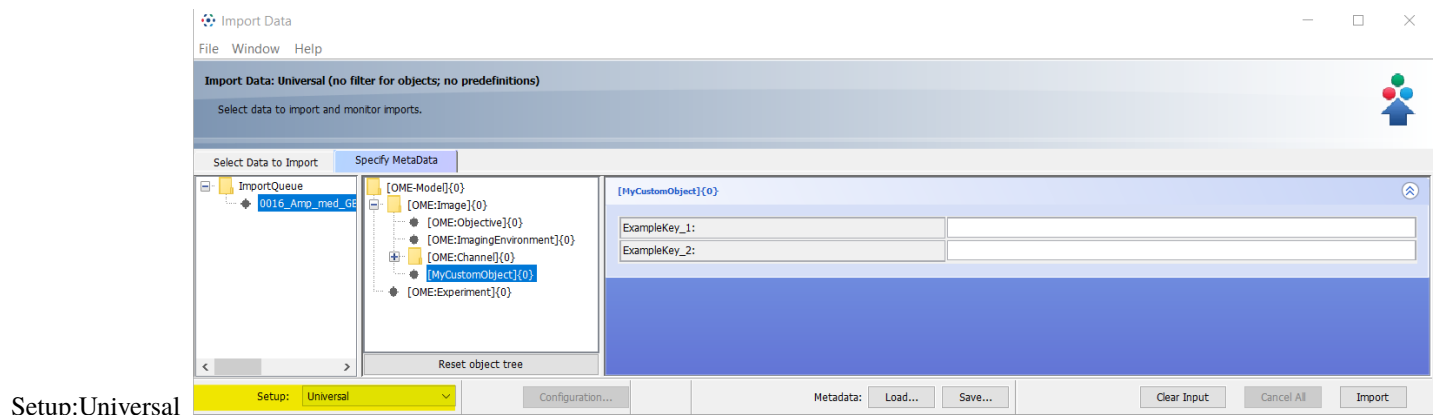
NOTE: You can specify different predefined values for the same object by generate multi <ObjectPre> elements for this object. OMERO.mde than shows you a table of available Elements Predefinitions) for this object.

Example

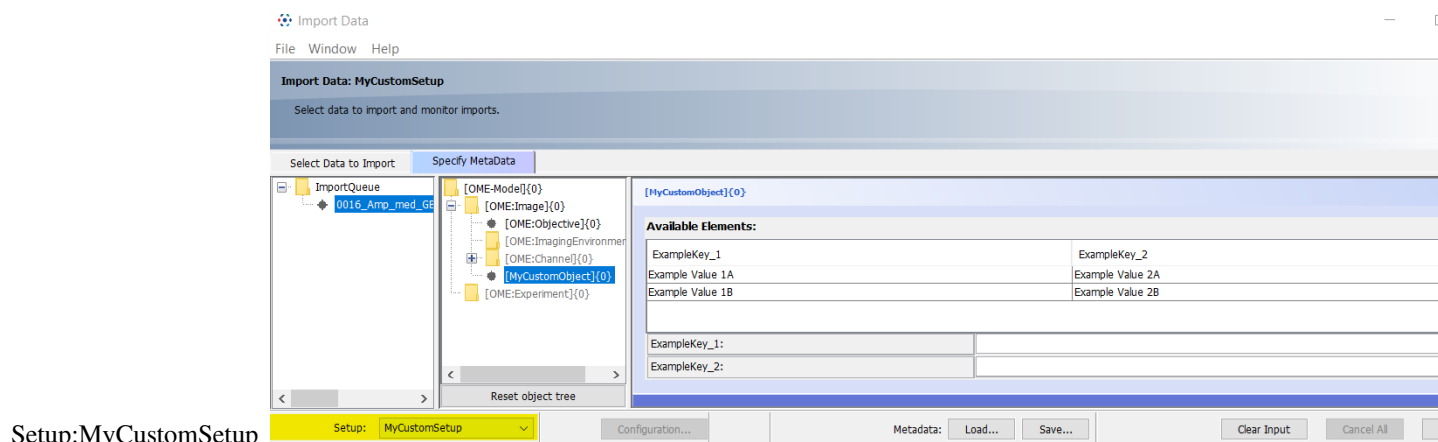
Specification like:

```
<MDEConfiguration>
  <MDEPredefinitions>
    <SetupPre Name="MyCustomSetup">
      <ObjectPre Type="MyCustomObject">
        <TagData DefaultValues="" Name="ExampleKey_1" Type="TextField"
          Unit="" Value="Example Value 1A" Visible="true" />
        <TagData DefaultValues="" Name="ExampleKey_2" Type="TextField"
          Unit="" Value="Example Value 2A" Visible="true" />
      </ObjectPre>
      <ObjectPre Type="MyCustomObject">
        <TagData DefaultValues="" Name="ExampleKey_1" Type="TextField"
          Unit="" Value="Example Value 1B" Visible="true" />
        <TagData DefaultValues="" Name="ExampleKey_2" Type="TextField"
          Unit="" Value="Example Value 2B" Visible="true" />
      </ObjectPre>
    </SetupPre>
  </MDEPredefinitions>
  <MDEObjects>
    <Definitions>
      ...
      <ObjectDef Type="MyCustomObject">
        <TagData DefaultValues="" Name="ExampleKey_1" Type="TextField"
          Unit="" Value="" Visible="true" />
        <TagData DefaultValues="" Name="ExampleKey_2" Type="TextField"
          Unit="" Value="" Visible="true" />
        <Parents Values="OME:Image" />
      </ObjectDef>
    </Definitions>
    <Configuration>
      <SetupConf Name="MyCustomSetup">
        <ObjectConf Type="OME:Image"/>
        <ObjectConf Type="MyCustomObject"/>
      </SetupConf>
    </Configuration>
  </MDEObjects>
</MDEConfiguration>
```

will create input form like



Setup:Universal



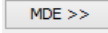
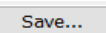
Setup:MyCustomSetup

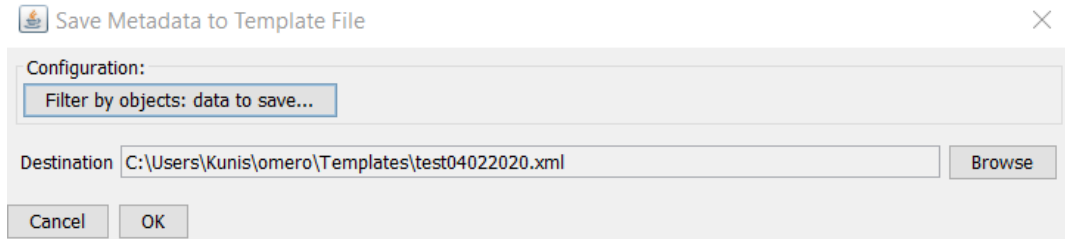
Use input templates for OMERO.mde

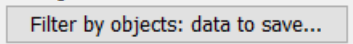
**** only available from OMERO.insight v5.5.11**** You can save your inputs for MDE objects in a template file for reused the inputs for another import.

Step-by-Step

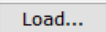
Save input:

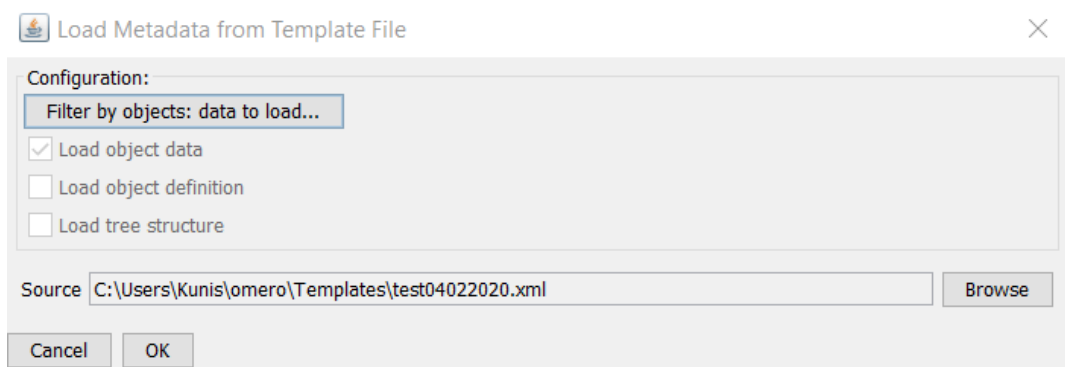
1. Start OMERO.importer and login.
2. Select your files for import and add it to the queue.
3. Click on the MDE Button  at the right bottom pane.
4. Add your inputs.
5. Select in the middle bottom pane  to get the *Save Template* dialog

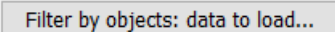


6. Select the object types for which the input should be saved. To do this press the button  and select the types.
7. Choose the location and file name for your template file.
8. Press ok to save the file.

Loading the template file for another import:

1. In the MDE Pane select the dir or file for which the input should load.
2. Select in the middle bottom pane  to get the *Load Template* dialog



3. Select the object types for which the input should be loaded. To do this press the button  and select the types.
4. Choose the location and file name of your template file.
5. Press ok to load the values.

3.5 OMERO.web extensions to view data

OMERO.web is a flexible Django-based Web platform offering extension points. Those extensions are not installed by default when deploying OMERO. They require some additional setup steps.

All the Web extensions can be installed using `pip`. Check each app for configuration details.

3.5.1 OMERO.figure

OMERO.figure is a popular tool for creating figures from Images in OMERO. Image metadata can be used to facilitate figure creation. For more information, see <https://github.com/ome/omero-figure>.

3.5.2 OMERO.FPBioimage

OMERO.FPBioimage is a volumetric visualization tool. For more information, see <https://github.com/ome/omero-fpbioimage>.

3.5.3 OMERO.iviewer

We introduce OMERO.iviewer, a 2D viewer which can open and browse multi-t, multi-z and multi-channel images and allows to draw and edit Regions of Interest (ROIs) and perform some rudimentary image analysis. It also offers the ability to view several images at the same time and synchronize the view. For more information, see <https://github.com/ome/omero-iviewer>.

3.5.4 OMERO.parade

OMERO.parade is a data mining tool. For more information, see <https://github.com/ome/omero-parade>.

OMERO.figure

OMERO.figure is a popular tool for creating figures from Images in OMERO. Image metadata can be used to facilitate figure creation. For more information, see <https://github.com/ome/omero-figure>.

Contents:

Create figures using OMERO.figure

OMERO.figure is a web-based tool for creating figures from Images in OMERO. Image metadata can be used to facilitate figure creation.

Description

This guide covers:

- Opening images in OMERO.figure to create a new figure
- How to add additional images to an existing figure
- Arranging panels in the desired figure layout
- How to synchronize the rendering settings between panels
- How to add scalebars, labels and ROIs to panels
- How to save and export figures as PDF or TIFF

Setup

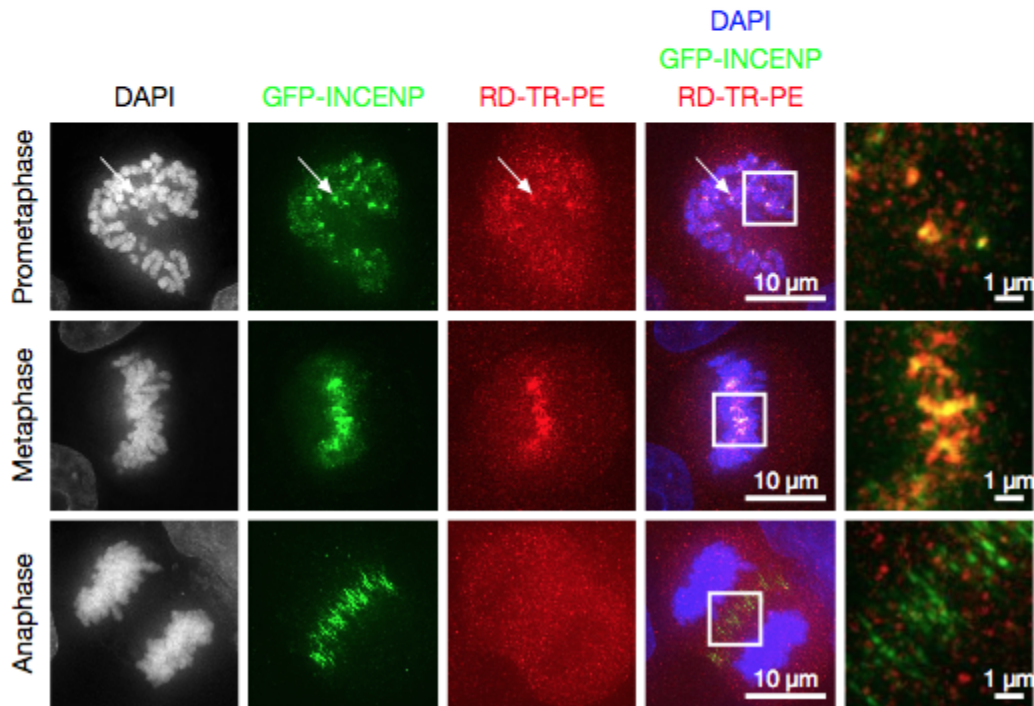
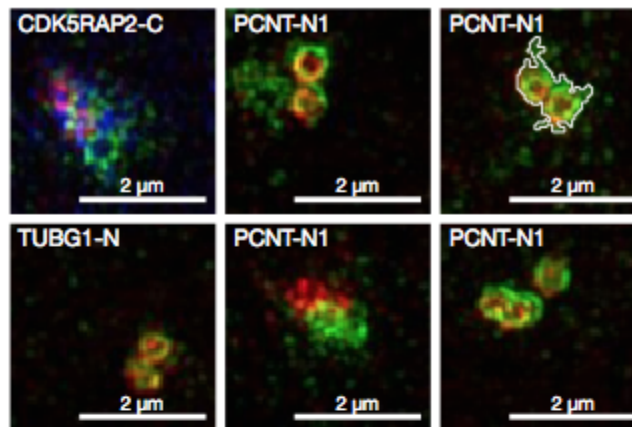
- Install the OMERO.figure web app as described at <https://pypi.org/project/omero-figure/>

Resources

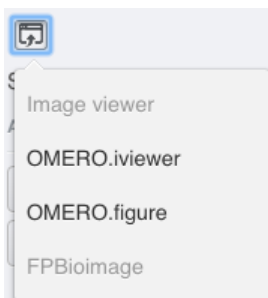
- Sample images from the Image Data Resource (IDR) [idr0021](#). See [idr0021-data-prep.md](#) for download and import instructions.
- DV images from [siRNAi-HeLa](#).
- SVS ‘big’ pathology images from [SVS](#).

Step-by-Step

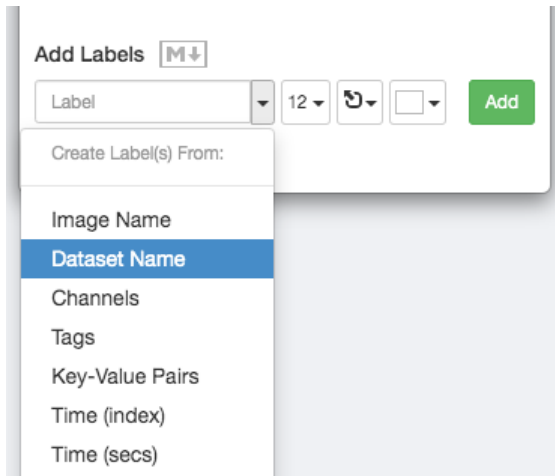
Using the sample images above, we will create a figure like this, but you can use any multi-channel images.



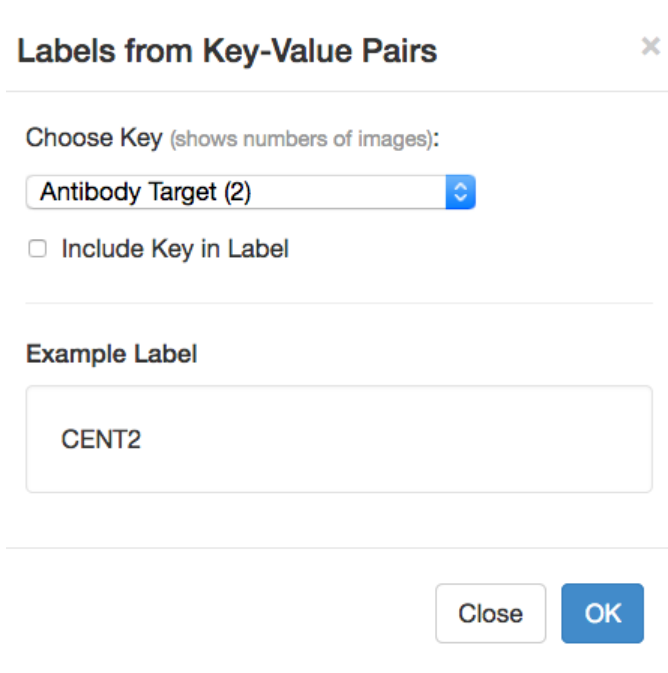
1. In the webclient, select 6 images from the **idr0021** Project.
2. In the right-hand panel, click the *Open with...* button and choose *OMERO.figure*:



3. This will open these images in OMERO.figure in a new browser tab.
4. Drag to arrange the Images approximately into two rows, select all (use Ctrl-A or drag to select) and click the *snap to grid* button  at the top of the page.
5. Select all Images and Zoom in around ~300%, using the Zoom slider in the right *Preview* tab.
6. Go to the *Labels* tab, select all Images and add a Scalebar: Click the *Show* button, choose a length of 2 m, click the *Label* checkbox and adjust the size of the Label to 12.
7. Add labels: choose *Dataset Name* in the label input dropdown list, choose *color white* and *position=top-left*.



8. Click *Add* to create the new label.
9. Add labels: choose *Key-Value Pairs* in the label input dropdown list, and in the following popup choose the Key which you want to add the value of from the dropdown menu.



10. Click *OK* in that dialog to create the new label.
11. Select one image. In the *Labels* tab, click the *Edit* button for ROIs.

12. If the image has ROIs in OMERO, click *Load ROIs*.
13. Mouse over the list to show each ROI on the Image and click to add it to the Image.
14. Click *OK* to close the dialog.
15. Return to the webclient tab, select the **siRNAi-HeLa** Dataset. N.B.: You may wish to filter the images when selecting those to add to your figure, e.g. Filter by Rating.
16. Select 3 images and in the right-hand panel click the link icon  then copy the link.
17. Return to the OMERO.figure tab, click *Add Image* button and paste the link into dialog. Click *OK*.
18. Arrange the 3 images into a vertical column, select all and click *snap to grid* button.
19. Copy the 3 images and paste (keyboard shortcut or *Edit > Copy/Paste*) 3 times to create 3 more columns.
20. Select the panels in the first column and adjust the rendering settings: Turn only the first channel on and set the color to white.
21. Repeat for the next 2 columns: 1 channel turned on for each column, adjusting the levels if desired, leaving the 4th column as **merged** with multiple channels on.
22. Select all panels and zoom a little. Then select all the panels from one row and drag the image in the *Preview* tab to pan the selected images to the same point.
23. Copy and paste the **merged** column again to create a 5th column. Zoom in to approx 500%.
24. Select the *Labels* tab, select the **merged** and **zoomed** columns and click *Show Scalebar* button.
25. Click the *Label* checkbox to add a label to the scalebar. Select only the zoomed-in panels and change the scalebar to 1 micron.
26. Select the top-left panel and enter a label text in the *Add Labels* form. “Prometaphase” in the example above.
27. Set the label size (14), position (left vertical) and color (black) and click *Add* to create a label.
28. If we have Tags on the images, we can use these to create labels:
29. Select the first column of panels and choose *Tags* from the label text-field drop-down options. Click *Add*.
30. Select the first row of panels and create the labels in the *top* position using the *Channels* option to add Labels for active channels in each image.
31. Edit the created labels located at the bottom of the *Info* tab to rename the green labels to **GFP-INCENP**.
32. Select just the first **merged** Image and click the ROIs *Edit* button in the *Labels* tab.
33. Draw arrows or other shapes on the Image, or load ROIs from OMERO. Click *OK* to close the dialog.
34. Click *Copy ROIs* in the *Labels* tab, select the other panels in the same row and click *Paste* to add ROIs to these panels.
35. To create a Rectangle ROI indicating the region of the zoomed-in image, select the zoomed-in image on the first row and click *Copy* of the cropped region at the bottom of the *Preview* tab.
36. Now select the zoomed-out ‘**merged**’ panel, and paste this region as an ROI by clicking *Paste* under ROIs section of the *Labels* tab.
37. Repeat for other rows of the figure. At this point we have created the figure in the screenshot above.

Saving and exporting figures

1. Go to *File > Paper Setup...* and in the dialog that pops up choose *Pages: 2*. Click *OK*.
2. Finally return to the webclient, select ‘Big’ images from the **svs** Dataset, copy the link to them and paste it into the *Add Image* dialog in OMERO.figure.
3. Move the big images to the 2nd page.
4. In the header, click on the *Save* button to save the Figure as “Figure 1”.
5. The URL will update. You can bookmark this URL or share with collaborators to view your figure.
6. To open other saved files, go *File > Open...*
7. We can view figures from our collaborators here and filter by name or Owner.
8. Choose a figure to Open. For example the **Aurora-B figure 2** from trainer-2.
9. Select a panel and click on the *Webclient* link in the *Info* tab to show the image in the webclient.
10. Back in OMERO.figure, go to *File > Open...* to choose the “Figure 1” file saved above.
11. Click on *Export PDF* to export it as PDF.
12. Download the PDF and open it. If opened in a suitable application e.g. Illustrator, the elements on the page will still be editable.

OMERO.figure scripting

Description

We can use Python scripts on the OMERO.server or JavaScript in the browser Console to create or modify OMERO.figures. These are experimental features and not documented elsewhere.

Setup

- Install the OMERO.figure web app as described at <https://pypi.org/project/omero-figure/>
- Upload the script `Split_View_Figure.py` to the OMERO scripting service
- For the *Shapes heatmap from OMERO.table* example, you’ll need to draw ROIs on an Image and create an OMERO.table on the Image with a Shape column as described in the example.

Resources

- For *Figure creation in Python*, any multi-channel images, e.g. from siRNAi-HeLa
- For the *Labels from Map Annotations* example, any time-lapse images, e.g. FRAP.
- For the other sections, any images can be used.

Figure creation in Python

OMERO.figure files are simply JSON data, stored in OMERO File Annotations with a specific namespace of `omero.web.figure.json`. We can create these files using Python scripts, uploaded to the OMERO.scripting service to make them available to all OMERO users.

The format of the JSON is described in the [Format](#) page. We will use the example [Split_View_Figure.py](#) script.

1. Select a few multi-channel Images in the webclient.
2. Click on the Script button  in the top-right of the webclient and choose the `Split_View_Figure.py` script (e.g. under Workshop Scripts).
3. Choose a value for *Row Labels* input. The default is label each row with the *Image Name*, but you can also choose *Tags* as in the example below.
4. Run the script. When complete, open the OMERO.figure app and File > Open.
5. Choose the most recent figure, called “Split View Figure”.

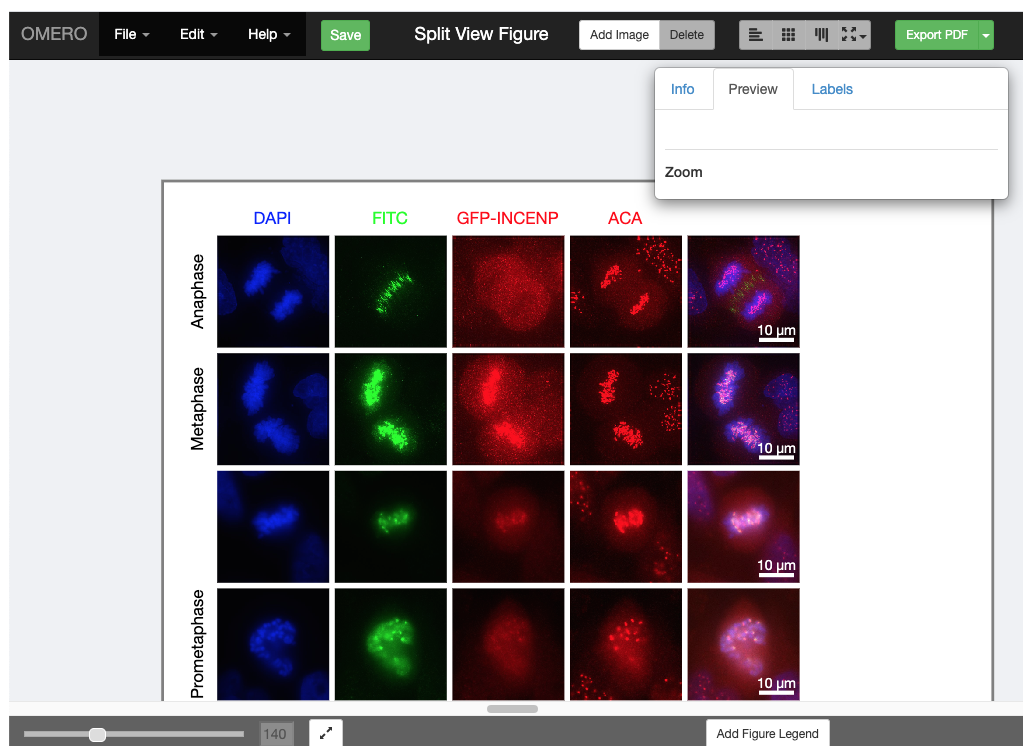


Figure editing in JavaScript

1. To see the data model for any current file in OMERO.figure, go to *File > Export as JSON...*
2. You will see that the `panels` list defines the panels and each panel has attributes. For example, a panel with a single white label might include the following attributes:

```
"name": "image1.tiff",
"labels": [{ "text": "label text", "size": 12, "position": "topleft", "color": "FFFFFF" } ],
"x": 200, "y": 200, "width": 100, "height": 100,
...many other attributes not shown...
```

3. The `figureModel` variable is accessible in the *Console* of the browser *Developer Tools*. The easiest way to open the developer tools in most browsers is to open the context menu (right-click) anywhere on the page and choose *Inspect*.
4. We can use `figureModel.getSelected()` to get selected panels and for each panel we can call `p.set()` to change an attribute.
5. For example, to set the `height` of several panels to 200, we can select the panels in the figure UI and paste this code snippet in the *Console*:

```
figureModel.getSelected().forEach(function(p){
  p.set('height', 200)
});
```

6. There are several JavaScript examples in the `scripts` folder. Many of these are quite simple and self-explanatory. Below are some more complex examples that require specific set-up steps.

Example 1: Labels from Map Annotations

We will use the time-lapse images listed above to create a FRAP figure but you can use any time-lapse images.

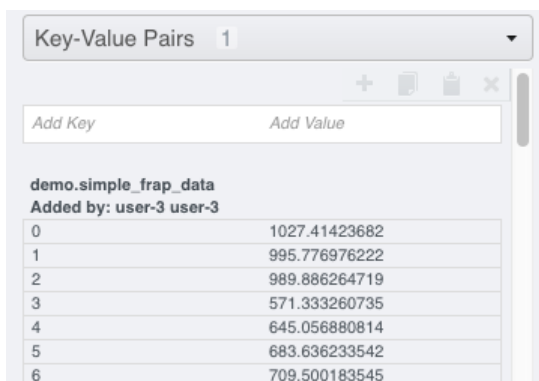
The screenshot displays the OMERO web interface. At the top, there's a menu bar with 'File', 'Edit', 'Help', and a 'Save' button. Below the menu, there's a toolbar with 'Add Image', 'Delete', and 'Export PDF' buttons. The main area shows a figure with 12 panels arranged in a 2x6 grid. Each panel has a label in the top-left corner. The top row labels are 1027, 989, 683, 766, 820, and 827. The bottom row labels are 1000, 500, 750, 800, 850, and 900. A right-hand panel shows a zoomed-in view of the selected panel (1000) with a green fluorescence signal. Below the figure, a console window shows JavaScript code that uses AJAX to load JSON data and `p.add_labels()` to create labels.

```
> figureModel.getSelected().forEach(function(p){
  var image_id = p.get('imageId');
  var ns = 'demo.simple_frap_data';
  var url = WEBINDEX_URL + 'api/annotations?type=map&image=' + image_id;
  var theT = p.get('theT');

  $.getJSON(url, function(data){
    // Use only the values from the first annotation
    var values = data.annotations[0].values;
    var labels = [{
      'text': "" + parseInt(values[theT][1]),
      'size': 12,
      'position': "topleft",
      'color': "ffffff"
    }];
    p.add_labels(labels);
  });
});
< undefined
>
```

1. We can use AJAX to load JSON data and we will use `p.add_labels()` to create labels.

- In this example we will load the FRAP intensities from the Map Annotations on these images.
- Select 2 FRAP images that have previously been analysed to create a Map Annotation with the namespace `demo.simple_frap_data`.

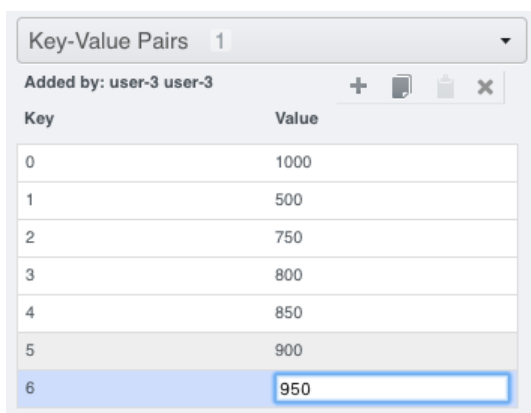


Key-Value Pairs 1

Added by: user-3 user-3

0	1027.41423682
1	995.776976222
2	989.886264719
3	571.333260735
4	645.056880814
5	683.636233542
6	709.500183545

- Alternatively, you can add your own Map Annotation with each *Key* being a *T-index* (start at 0), and the *Value* will be a FRAP intensity (number).



Key-Value Pairs 1

Added by: user-3 user-3

Key	Value
0	1000
1	500
2	750
3	800
4	850
5	900
6	950

- Create a Figure with 2 images.
- Copy and paste each image several times and increment T-index in the Preview panel to show multiple time-points per image.
- Open the browser console by *right-click > Inspect Element (Firefox)* or *right-click > Inspect (Chrome)* and click on the *Console* tab.
- Copy the code from `figure_frap_mapannotation_label.js` <https://github.com/ome/omero-guide-figure/tree/master/scripts/figure_frap_mapannotation_label.js>.
- Drag to select the FRAP movie images in the figure.
- Paste the code into the console. **Do not hit enter yet.**
- Inspect the code. It will iterate through each of the **selected** panels, an AJAX call is made to load the Map Annotations with the namespace that we created from FRAP values above.
- NB: If you manually created your own Map Annotation above, you can remove the line `url += '&ns=' + ns;` to avoid filtering by namespace.
- The FRAP values are a list of `[key, value]` pairs and we can get the value for the current T index of the panel with `values[theT][1]` and use this to create a label.
- Hit Enter to run the code on selected panels.

15. The labels should be added. Note that you can undo and redo these changes in the UI as normal.

Example 2: Shapes heatmap from OMERO.table

This example uses an OMERO.table linked to each Image to generate a heatmap of colors applied to Shapes on the figure panel.

Setup:

1. If you wish to use Images and table data from *idr0079*, see the setup steps at [idr0079-data-prep](#).
2. Alternatively, perform the following steps:
3. For the Image you wish to use, add some ROIs to the Image. You can see the ROI and Shape IDs in the iviewer ROI table.
4. To setup the OMERO.table, create a CSV file with an `Roi` column and a `Shape` column containing the corresponding IDs and 1 or more number columns. The `#header` defines the column types: `l` (long) for *integers* and `d` (double) for *floats*. For example:

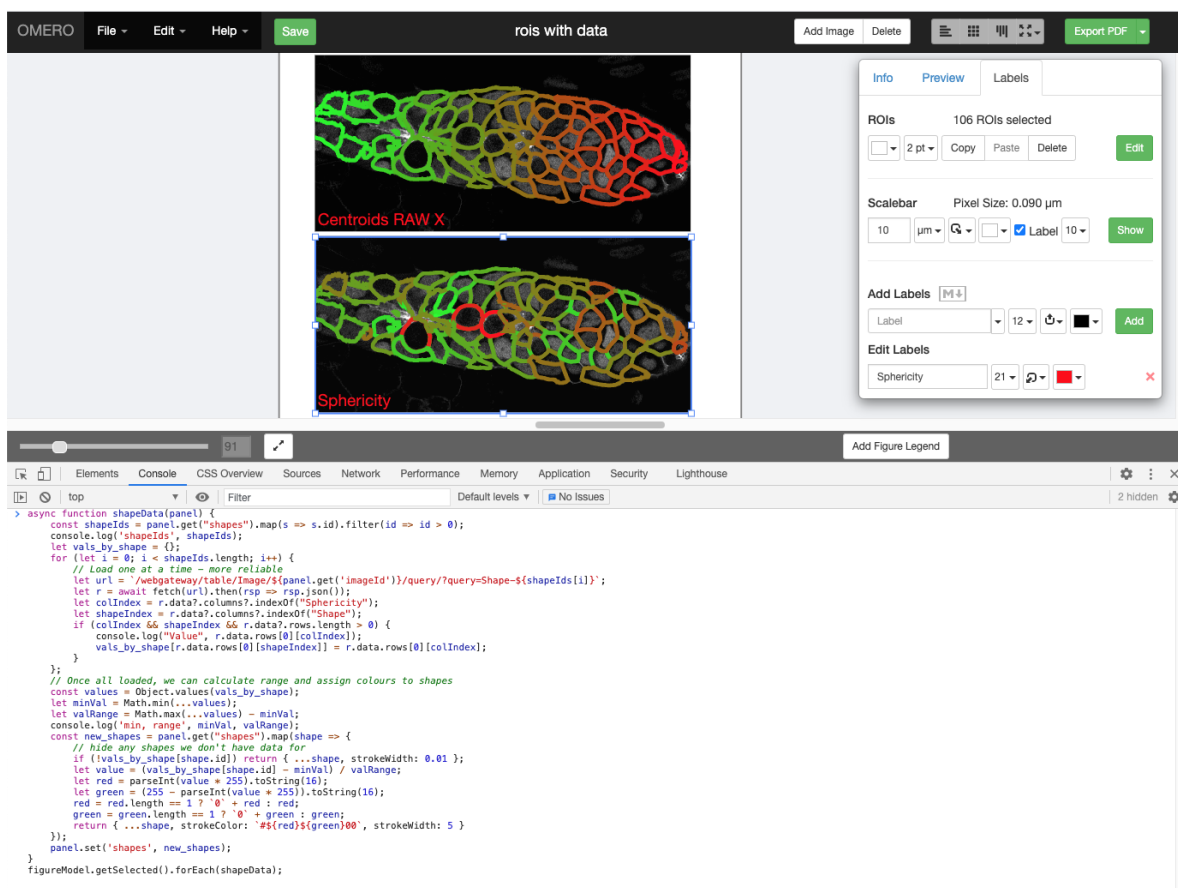
```
# header roi,l,d,d,l
Roi,Shape,Area,Sphericity,Pixels
1,10,34.5,0.5,110
2,11,18.2,0.6,55
2,12,44.1,0.9,210
```

5. Save the edited csv as `data.csv`.
6. With `omero-metadata` installed on the command-line, we can create an OMERO.table on the Image, using the Image ID:

```
$ omero metadata populate Image:123 --file data.csv
```

OMERO.figure steps:

1. In OMERO.figure, add the Image to the figure, then in the ROIs dialog, load the Shapes from OMERO and add them to the panel. The JSON data for each Shape will have an `id` that corresponds to the Shape in OMERO.
2. View the JavaScript snippet at [figure_table_data_shapes.js](#). This uses the ID of each Shape of the panel to query the most recent OMERO.table on the Image using the endpoint: `/webgateway/table/Image/{imageId}/query/?query=Shape-{shapeId}`, which returns all table rows for that Shape ID. From the JSON returned, we find the column index for the data we want, e.g. `Sphericity`, and then get the value for that column. Once the values for all Shapes on the panel are loaded, the code calculates the range and generates a heatmap color for each value in that range. This is set as the color on each Shape.
3. Select the panel in the figure, then paste the JavaScript code into the browser *Console* and hit Enter
4. In the screenshot below, Shapes in the first panel are colored according to the `Centroids_RAW_X` column and Shapes on the lower panel are colored according to the `Sphericity` column. Images in this example are from *idr0079*.



Move Figures between Groups

Moving Figures between Groups in OMERO might be necessary for example in cases where a scientist decides to move their data into another Group to enable cooperation and viewing by other colleagues or to make the data public.

As with all data in OMERO, Figures belong to a particular Group and are visible only to members of that Group. Figures can contain Images from the same Group as the Figure or from any other Group. Figures and the Images they contain can be independently moved from one Group to another but should ideally be kept in the same Group. This avoids the situation when a user can view a Figure, but not the Images within it.

This walkthrough offers two alternatives for getting Figures into another Group, either with or without duplicating the Figures and the Images contained in them.

Description

This guide covers:

- Moving of Figures and Images between Groups.
- Duplicating of Figures and Images into a new Group.

Setup

- Install the scripts `Figure_Images_To_Dataset.py` and `Dataset_Images_To_New_Figure.py` on your OMERO.server.
- We suggest in this workflow the installation under a folder `Figure scripts` but you can install the script in any folder.
- See [server-side script guides](#) for further details.

Resources

- Any Images and Figures created from these Images.

Step-by-Step

Moving of Figures and Images between Groups

This workflow assumes that you have an OMERO.figure already created and want to move this Figure into another Group.

1. Log in to OMERO.web.
2. Click on `Figure` link above the central pane to get into OMERO.figure.
3. Inside OMERO.figure, click `File > Open` and open the Figure you wish to move to another Group. Copy the last part of the url from the address bar of your browser, which contains the Figure ID, e.g.:
<https://your-omero-server.org/web/figure/file/79828>
has the Figure ID 79828. The Figure ID will be needed later in case you also want to collect and move the Images contained in the Figure as described below.
4. Click `File > Move Figure to Group...` In the following dialog, select the Group you wish to move the Figure to and click OK.
5. Observe a dialog reporting a success of the move. Verify that when you now again click `File > Move Figure to Group...` the Figure is now reported to be in your intended Group.

Note: You might want to move the Images contained in the Figure to the other Group as well. This step is optional, if you want to use your Figures by yourself only, but you should consider it in case the users you want to share your Figure with do not have permissions to see your Images in the original Group (for example they are members of the target Group only).

1. In the following steps, we first collect all the Images in the Figure into a single Dataset to enable moving them to the target Group in a single step.
2. Go back to the tab with OMERO.web, create a new Dataset and select it.
3. Click on the Scripts icon  above the central pane of OMERO.web. Select `Figure scripts > Figure Images to Dataset`. Start the script and in the dialog, enter the Figure ID into the Figure IDs field. If you wish to work with multiple Figures, IDs can be separated with commas, e.g. 79828, 79830, 71228. Click Run.

4. When the script finishes, refresh the page and find the Images contained in your Figure linked to the Dataset. Note that the Images are linked to the chosen Dataset without removing them from any existing Dataset. They will be doubly-linked but not duplicated.
5. The above means that when you execute the Move into another Group, the Images in your Dataset will no longer be available in the original Group.
6. Move the Images into the Group you have moved the Figure to. Be sure to select the Images in OMERO.web when moving, not the Dataset. If the Dataset is selected, the Images are left in the original Group, and only the empty Dataset is moved. This is because the Images are linked to the original Dataset which is left in the original Group. Follow [the Move workflow](#) to execute the Move. The Move will enable any member of the target Group to view both the Figure and the Images within it.

Duplicating of Figures and Images into a new Group

If you wish to keep your Figures and the contained Images in their original Group, while also making them available to users in another Group, you can duplicate them as described below.

In this workflow, we first duplicate the Images within one or more Figures. Then move these Images to the target Group.

Finally, using a script, we copy the Figure into the target Group, replacing the Images within it with the duplicated Images.

Note that this workflow involves usage of Command Line.

1. Collect the Images contained in the Figures into a single Dataset using the workflow [Moving of Figures...](#) For this, you can find out the Figure IDs either manually as described in the [Moving of Figures...](#) or by running hql queries on the command line, such as:

```
$ omero hql --all --limit 1000 --style plain --ids-only "select f.id from
↪FileAnnotation f where (f.details.group.name = 'Lab1' and f.details.owner.id =
↪454)" | sed -e 's/^.*,//g' | paste -s -d, -
```

which will retrieve all the Figure IDs of user with ID 454 in a Group Lab1 in a format which you can immediately copy and paste into the Figure Images to Dataset script.

2. Start your command line terminal and duplicate the Dataset with the Images contained in the Figures as described in [the Duplicate workflow](#).
3. Go to OMERO.web, select the duplicate Dataset and Move it to the target Group. For that, follow [the Move workflow](#).
4. Find the Dataset which you have just moved and select it.
5. Click on the Scripts icon  above the central pane of OMERO.web. Select Figure scripts > Dataset Images To New Figure.
6. Start the script and in the dialog, enter the Figure ID into the Figure IDs field. If you wish to work with multiple Figures, IDs can be separated with commas, e.g. 79828, 79830, 71228. Click Run. This will copy each specified Figure, update the Images within it to those in the duplicate Dataset (using the Image name to match the replacement Images) and save the Figure to the new Group.
7. Click on Figure link above the central pane to get into OMERO.figure.
8. Inside OMERO.figure, click File > Open. In the top-right corner of the new dialog, click on the Group drop-down and select your target Group name. Verify that the list contains the newly created Figures.

Contribute

Changes to the documentation or the materials should be made directly in the [omero-guide-figure repository](#).

OMERO.FPBioimage

OMERO.FPBioimage is a volumetric visualization tool. For more information, see <https://github.com/ome/omero-fpbioimage>.

Contents:

OMERO.FPBioimage

In this document, we introduce OMERO.FPBioimage, a 3D volume viewer for OMERO.web.

Description:

We will show here:

- How to open a multi-z image in OMERO.FPBioimage

Resources:

Example files used

- <https://downloads.openmicroscopy.org/images/DV/siRNAi-HeLa/>

Note: Only some of the images in this dataset are z-stacks, for example

- https://downloads.openmicroscopy.org/images/DV/siRNAi-HeLa/IN_03.r3d_D3D.dv

Setup:

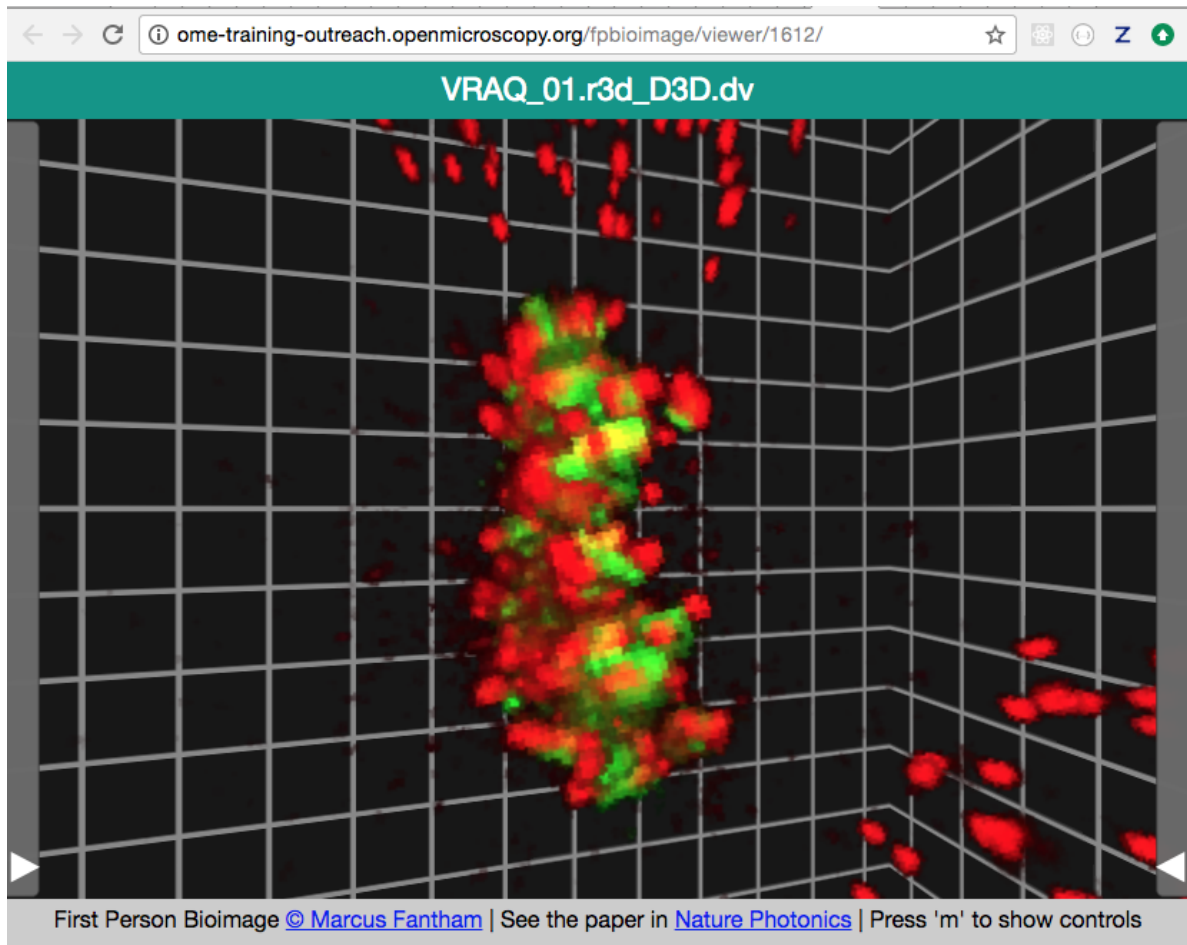
OMERO.FPBioimage installation

OMERO.FPBioimage is a pip installable application for OMERO.web. Follow the steps described in <https://pypi.org/project/omero-fpbioimage/> to install it and configure the OMERO.web accordingly.

Step-by-Step:

1. Login to OMERO.web and open an image from the Dataset siRNA-HeLa with multiple Z-sections e.g. *VRAQ_01.r3d_D3D.dv* in a 3D viewer: OMERO.FPBioimage.
 - a. First select the Image.
 - b. In the Preview tab, switch off all channels except FITC and the GFP-INCENP channel.
 - c. Save the new rendering settings.
 - d. Use right-click menu on the image in the left panel, or the Open with... icon  on top of the right-hand pane to open the image with FPBioimage.

- e. Click Start in the new viewer window.
- f. We can see that the centromeres are well aligned on the metaphase plate on the selected Image, whereas the centromeres are located in and around the spheroid on the *IN_02.r3d* Image for example.



OMERO.iviewer

We introduce OMERO.iviewer, a 2D viewer which can open and browse multi-t, multi-z and multi-channel images and allows to draw and edit Regions of Interest (ROIs) and perform some rudimentary image analysis. It also offers the ability to view several images at the same time and synchronize the view. For more information, see <https://github.com/ome/omero-iviewer>.

Contents:

View images in OMERO.iviewer

We introduce OMERO.iviewer, a 2D viewer which can open and browse multi-t, multi-z and multi-channel images and allows to draw and edit Regions of Interest. It also offers the ability to view several images at the same time and synchronize the view.

Description

We will show here:

- How to open multidimensional images in OMERO.iviewer
- How to change rendering settings in OMERO.iviewer and scroll through z and t
- How to change the Lookup table and invert the displayed intensities for a particular channel
- How to use the Histogram feature
- How to zoom and rotate the image
- How to create a maximum intensity Projection and save it
- How to save a current viewport as png in OMERO.iviewer
- How to use and synchronize multiple viewer windows inside OMERO.iviewer

Resources

Example files used

- <https://downloads.openmicroscopy.org/images/DV/siRNAi-HeLa/>

Note: Only some of the images in this dataset are z-stacks, for example

- https://downloads.openmicroscopy.org/images/DV/siRNAi-HeLa/IN_03.r3d_D3D.dv

Setup

OMERO.iviewer installation

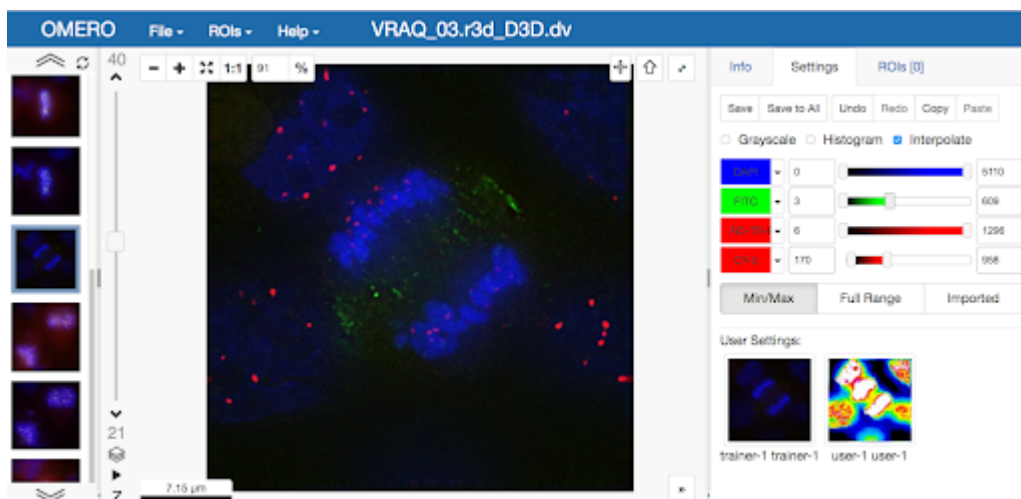
OMERO.iviewer is a pip installable application for OMERO.web. Follow the steps described in <https://pypi.org/project/omero-iviewer/> to install it and configure the OMERO.web accordingly.

The walkthrough assumes that OMERO.iviewer has been set up as the default viewer for OMERO.web.

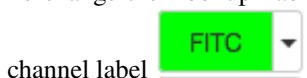
Step-by-Step

1. Double-click on an Image thumbnail, or click the Full Viewer  button in the right-hand pane to open an Image in a larger viewer called OMERO.iviewer. It is a Web app developed and released independently from the webclient. The OMERO.web framework can be extended with multiple apps to view data in different ways.
2. Zoom in and out of the image by using the + and - buttons in top-left corner or by scrolling the mousewheel.
3. Rotate the image by holding Shift on your keyboard and then drag the image using the mouse.

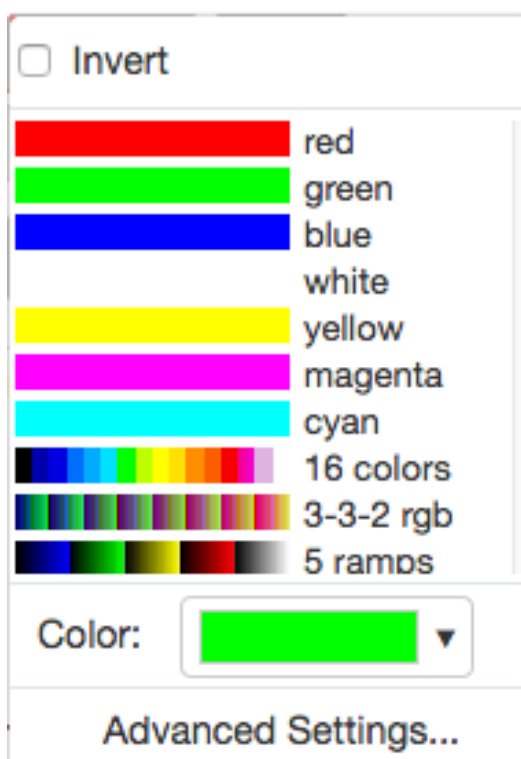
4. We can adjust the rendering settings and scroll through Z or T.



5. To change the Lookup Table (LUT) of a particular channel, click on the downward facing arrow next to the

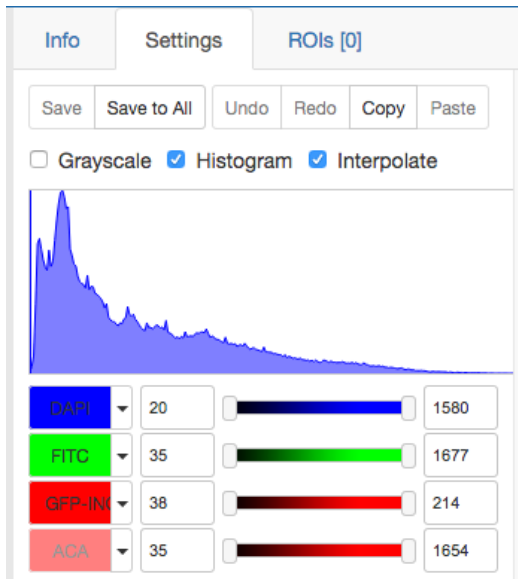


6. In the menu, select the LUT you wish to use. You can find here all the LUTs which are supported in ImageJ.

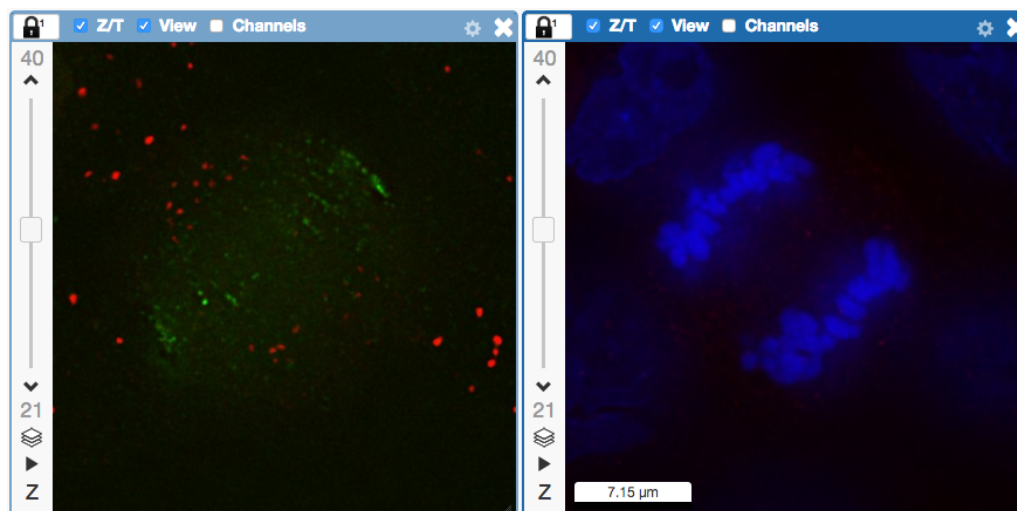


7. In the same menu, you can also invert the intensity of the channel by checking the checkbox on the top.
8. Still staying in the Settings tab in the right-hand side of OMERO.iviewer, click on the checkbox next to the Histogram word ☒ Histogram near the top.
9. A histogram of the first channel which is on will appear. Note that when you move the rendering settings sliders

beneath the histogram, the histogram updates to showing the channel you just manipulated.



10. The **Interpolate** option provides a smoother rendering of the image in the viewer. Unchecking this will show the pixels at the same resolution as the original data. This option has no effect on the underlying pixel intensity values.
11. Make a maximum intensity Projection of a z-stack by clicking the stack icon  in the bottom-left corner of the central pane, just under the z-slider.
12. Note that on the vertical z-slider, there are now two knobs. Adjust the knobs to select the desired portion of the z-stack to be projected. The central pane will show a preview of your Projection.
13. Save the Projection by selecting **File > Save Projection as new image**. In the following dialog, click either **Navigate to Image in Webclient** or **Open Image in iviewer**.
14. Save the viewport by selecting in the top-left corner **File > Save Viewport as PNG**.
15. Compare the two channels of one of the Images in multi-image view:
 - Double-click on the thumbnail of the already opened Image in the left-hand pane of the viewer. This will open the Image again in a new window.
 - Select one of the windows and in the right-hand pane switch the green channel off.
 - On the other window, switch the blue channel off.



16. Work with images in the multi-view mode. There are several ways to get full views of the images from the thumbnails in the left-hand column.

- Click on a thumbnail to open the corresponding image in the currently active central pane viewer window, replacing the image in that viewer window.
- Double-click on a thumbnail to open a new central pane viewer window with the double-clicked thumbnail image.
- Drag and drop a thumbnail into an existing central pane viewer window to replace the image in that window with the image corresponding to the thumbnail which you just dropped into it.
- Drag and drop a thumbnail onto the white canvas area in the central pane to create a new viewer window.

17. We can synchronize the central pane viewers by adding them both to the same sync Group.

- Click the sync icon  in the top-left corner and select the first option in the dropdown menu.
- Repeat for the other viewer.
- Now both Images have the Z/T and View (zoom and position of viewport) synchronized.



18. Click on Info tab in iviewer now, and find, in the right-hand pane, the link to Dataset:

Dataset: [siRNAi-HeLa](#)

. Click on that link. This will bring you back to the webclient.

Work with ROIs in OMERO.iviewer

In this section, we cover the ability of OMERO.iviewer to work with ROIs, to draw, edit, annotate and evaluate ROIs in the images is shown. In this way, a simple image analysis can be achieved, such as getting the intensity measurements inside the pixels of the ROIs and sizes of the ROIs, such as areas for polygons and lengths for lines and polylines.

Description

We will show:

- How to inspect the intensities
- How to draw ROIs on the image
- How to view ROIs associated with different z or t planes
- How to use the **Planes** tab in OMERO.iviewer to see the spread of ROIs in multi-z, t images
- How to create comments on ROIs and enable and disable the ROI popups
- How to produce simple analysis results from the ROIs and export the results locally

Resources

New features video (ROI popups, ROIs on multi-z or multi-t images)

- <https://youtu.be/yDW5Sg0IIGU>

Example files used

- <https://downloads.openmicroscopy.org/images/DV/siRNAi-HeLa/>

Note: Only some of the images in this dataset are z-stacks, for example

- https://downloads.openmicroscopy.org/images/DV/siRNAi-HeLa/IN_03.r3d_D3D.dv

Setup

OMERO.iviewer installation

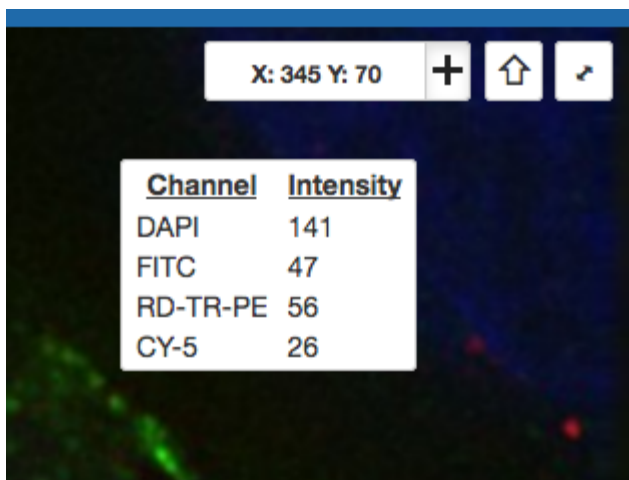
OMERO.iviewer is a pip installable application for OMERO.web. Follow the steps described in <https://pypi.org/project/omero-iviewer/> to install it and configure the OMERO.web accordingly.

The walkthrough assumes that OMERO.iviewer has been set up as the default viewer for OMERO.web.

Step-by-Step

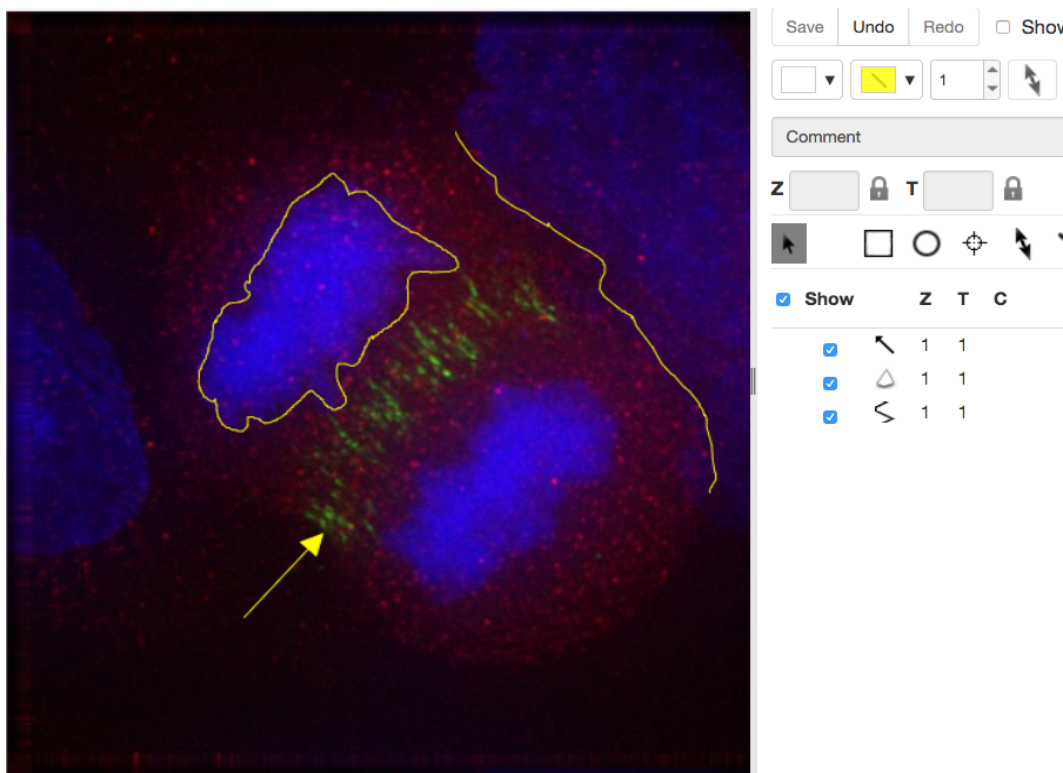
1. Double-click on an Image thumbnail, or click the **Full Viewer**  button in the right-hand pane to open an Image in a larger viewer called OMERO.iviewer. It is a Web app developed and released independently from the webclient. The OMERO.web framework can be extended with multiple apps to view data in different ways.

2. Click the Crosshairs icon  at the top-right of the viewer to enable the pixel intensity display for the mouse pointer. Then mouse over the Image to see the pixel intensities for the channels turned on.



3. The ROIs tab **ROIs [0]** includes tools for viewing and drawing ROIs on the Image. These are saved back to the OMERO server. Select the arrow tool in the right-hand pane  and draw an arrow on the Image, using Click-Move-Click (not drag), pointing to a feature in the Image and save it using the Save button located in the upper part of the right-hand pane. Draw also a couple of other ROIs on different Z planes.

- Dragging on the Image is used for panning the image.
- Shift-drag rotates the image (using selection tool) or draws freehand (polygon, polyline, rectangle and ellipse).



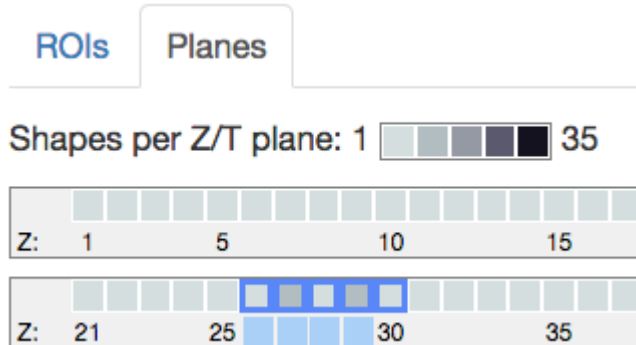
4. Clicking onto the ROIs in the right-hand pane table brings the viewport to the position where the selected ROI is in the middle of it and navigates the viewport to the timepoint the ROI is associated with.

☒ Show		Z	T	C	Comment	
☒		1	1			
☒		1	1			
☒		1	8			
☒		1	11			
☒		1	15			

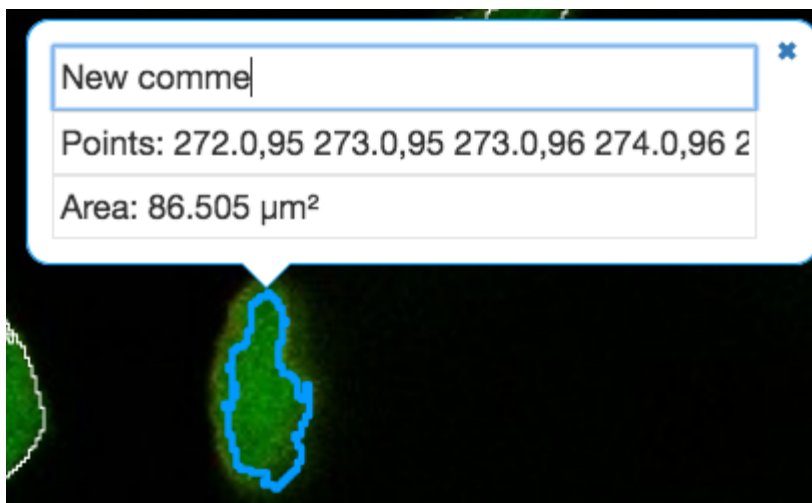
5. Select the Planes tab above the ROI table.

ROIs		Planes			
☒ Show		Z	T	C	Comments
☒		20	1		
☒		20	1		

6. The Planes tab gives you an overview of the ROIs distribution over the z or t sections. The boxes in the schema in the Planes tab represent single z or t planes and are selectable.



7. You can select one box, this will navigate to the corresponding plane in the image viewer. Alternatively, select a range of boxes, which will display a projection of the range of the selected planes (in case of multi-z image) in the main viewer.
8. Click onto a ROI in the image. A popup will appear. You can write a comment to this ROI directly into the popup.



9. The popups can be disabled for all ROIs on the image by clicking onto the cross of one popup and closing it, and re-enabled by using a context menu accessed by right-clicking onto the image.
10. You can copy the values in the popup such as area (select and `Ctrl + C`) and paste them into your local documents.
11. Select several ROIs from different z planes. You can select in the table (`Shift+click` to select a range) but also in the image itself using `Ctrl+drag` (`Cmd-drag` on Mac) to select multiple ROIs.
12. Export the Intensities, areas and line lengths into Excel. Select, in the top-left corner `ROIs > Export (Excel)`. This will export the values from the selected ROIs.
13. Note: ROI intensities and coordinates from the whole image or dataset can be exported using the script `Batch ROI Export`. Draw ROIs on the image as instructed above, go to OMERO.web and select the image or dataset in the left-hand side tree. Find the cog icon above central pane. Then, select the `Export scripts > Batch ROI Export` and run the script.

Contribute

Changes to the documentation or the materials should be made directly in the [omero-guide-iviewer repository](#).

OMERO.parade

OMERO.parade is a data mining tool. For more information, see <https://github.com/ome/omero-parade>.

Contents:

Analyze metadata using OMERO.parade

Description:

OMERO.parade is a metadata-mining plugin for OMERO.web. It enables access to the metadata of images in OMERO for plotting, display and filtering of images. Supported metadata includes number of ROIs, Key-Value pairs, and data stored in OMERO.tables.

Setup:

- Install the OMERO.parade web app as described at <https://pypi.org/project/omero-parade/>

Resources:

- Sample images from the Image Data Resource (IDR) [idr0021](#). See [idr0021-data-prep.md](#) for download and import instructions.
- Sample plate data from Plate with 422 on IDR from [idr0002](#). Use the script [idr_copy_plate.py](#) to copy Plate ID 422. Run the script with `$ python idr_copy_plate.py username password 422 --server your.server.org`

Step-by-Step:**Filter Images by Annotations and ROI Count**

1. Select the Project **idr0021**.
2. Choose the *Parade* option in the centre panel dropdown menu.
3. Expand all Datasets by clicking on the *Open All* button.
 - All the Datasets will be expanded in the left-hand tree.
 - The Thumbnails will be loaded in the centre panel. This allows to browse a full Project.
 - Note that if you collapse a Dataset in the tree, the Thumbnails will be removed from the centre panel.
4. In the *Add filter...* selection box, select the *Key_Value* item.
 - When the Map Annotations are loaded, pick the Key *Gene Symbol* and enter the Value *CEP* to show all *CEP* genes and then *CEP120* to show only images with that gene.
 - To remove this last filter, hover over the filter and click the *X* button that shows on hover.
5. In the *Add filter...* selection box, select the *ROI_Count* item.
 - Enter a Value > 20 . When you hover over the area used to enter the value, the range is indicated in the tooltip.
 - Then enter < 3 or 4 .
6. Remove all filters by clicking the *X* button showing on hover.
7. Select several images in the Dataset, and in the right-hand pane add a comment to them saying “poor staining”.
 Refresh, using the refresh button  above the left-hand pane.
8. In the *Add filter...* selection box, select the *Comment* item.
 - Enter a text “poor staining”
9. Note that you can use the selected images in right panel to annotate or *Open with...*
10. For example, *Open with Figure...*

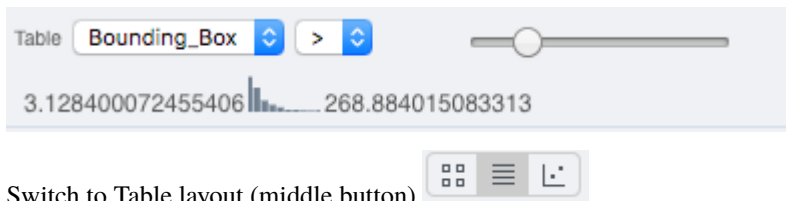
Analyze OMERO.table data using OMERO.parade

In case the parameter values on the images are coming from OMERO.tables (stored in OMERO.tables as numerical values), OMERO.parade also enables to compare parameters in metadata, using heatmaps ordering and plotting of the parameters against each other.

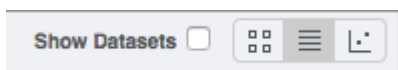
For how to create an OMERO.table from a csv file see <https://github.com/ome/omero-metadata/blob/master/README.rst>

For how to create an OMERO.table using a script, see workflow (link to Fiji workflow).

1. Select the Project **idr0021**.
2. Choose the *Parade* option in the centre panel dropdown menu.
3. Expand all Datasets by clicking on the *Open All* button.
4. In the *Add filter...* selection box, select the *Table* item so we can find using the analytical results generated previously:. Choose the *Bounding_Box* item and drag the slider to filter the Images. Note that *PCNT* has the largest number of Images with large ROIs.



5. Switch to Table layout (middle button).
6. In the selection box *Add table data...*, select
 - *Table_Bounding_Box*
 - *Table_Total_Area*
 - *Table_Image*
 - Note that it is currently not possible to remove a column.
7. Click on the name of a column to sort it.
8. Uncheck *Show Datasets* to sort all Images together e.g. by ROI count.



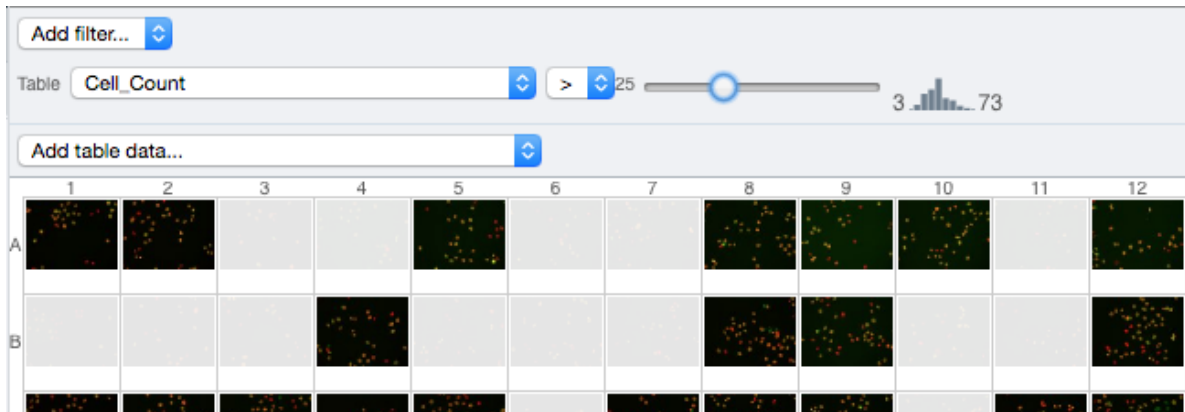
9. Check the checkbox in each column to show the *Heatmap*. Note the corresponding pattern in the Heatmap.
10. Switch now to the Plot Layout (third button).
11. It takes the first 2 columns of table data loaded and plots the values.
12. Filters can be added to plot the relevant results.
13. Try plotting by different Axis values.
14. Closing a Dataset in the left-hand tree removes the values from the plot.
15. Drag to select several outliers.
16. Note that you can use the selected images in right panel to annotate or *Open with...*
17. For example, *Open with Figure...*

Filter Plate Wells using OMERO.table data

Let's now look at the results generated by CellProfiler

1. Return to the webclient and select the Plate named **plate1_1_013**.
2. Select a Well and open the *Tables* pane in the General tab in the right-hand panel. This will show all the CellProfiler values for this Well.
3. In the *Thumbnails* chooser at the top-right of the centre panel, select the *Parade* plugin.

4. At the top-left of the centre panel choose *Add filter...* -> *Table* to filter Wells by the data from CellProfiler.
5. Change the filter from *ImageNumber* to *Cell_Count* (at the bottom of the list).
6. Now you can use a slider to filter Wells by Cell Count.



3.6 How to write a guide

The OMERO guide is hosted on [readthedocs](https://readthedocs.org/). This section describes on to create a new guide.

To create a new guide:

- Create a new repository on GitHub using the [template guide](#).
- Add the new repository as a new project on [readthedocs](https://readthedocs.org/).
- Add the new project as a subproject of [omero-guides](#) using an alias e.g. `omero-guide-figure` has been added as `figure`,
- Add the new repository to the [omero-guides repository](#) as a submodule using the same alias.
- Insert the guide into one of the existing files or in a new file if it is a new section.
- If you intend that the new repository contains a `binder` folder to generate an environment for analysis, create two new YAML workflow files from the templates [binder_badge.yml](#) and [repo2docker.yml](#), then place these new files to `.github/workflows`.

3.7 OMERO walkthrough example

In this document, we go through a common workflow that a scientist wishing to use OMERO might follow:

- Import data using the Desktop client.
- View images using the OMERO.iviewer plugin.
- Analyze images using a 3rd party tool, e.g. Fiji.
- Generate a figure using OMERO.figure.

3.7.1 Import

Import data using the Desktop Client

3.7.2 Description

In the first part, we first show how to import data by yourself and for yourself into OMERO using various import strategies. This will be mainly done using the OMERO.insight desktop client.

Second part of this import section will show how to import data for another user, using OMERO.insight. The user importing the data needs to have some admin or restricted-admin privileges. More information about restricted privileges can be found at <https://docs.openmicroscopy.org/latest/omero/sysadmins/restricted-admins.html>

The import for another user requires that the user doing the import has specific privileges. We will use the user `importer1`, this could be, for example, a facility manager.

We will show:

- How to install the OMERO.insight desktop client on Windows, Mac and Linux.
- How to import data for the user logged in using OMERO.insight.
- How to select a target Project and Dataset or create a new ones in OMERO for the imports.
- How to add Tags to imported images at the import stage, to facilitate the management of these images later in OMERO.server.
- How to import data for other users in OMERO.insight.

3.7.3 View

View images in OMERO.iviewer

We introduce OMERO.iviewer, a 2D viewer which can open and browse multi-t, multi-z and multi-channel images and allows to draw and edit Regions of Interest. It also offers the ability to view several images at the same time and synchronize the view.

3.7.4 Description

We will show here:

- How to open multidimensional images in OMERO.iviewer
- How to change rendering settings in OMERO.iviewer and scroll through z and t
- How to change the Lookup table and invert the displayed intensities for a particular channel
- How to use the Histogram feature
- How to zoom and rotate the image
- How to create a maximum intensity Projection and save it
- How to save a current viewport as png in OMERO.iviewer
- How to use and synchronize multiple viewer windows inside OMERO.iviewer

3.7.5 Inspect intensities, draw and evaluate ROIs

Work with ROIs in OMERO.iviewer

In this section, we cover the ability of OMERO.iviewer to work with ROIs, to draw, edit, annotate and evaluate ROIs in the images is shown. In this way, a simple image analysis can be achieved, such as getting the intensity measurements inside the pixels of the ROIs and sizes of the ROIs, such as areas for polygons and lengths for lines and polylines.

3.7.6 Description

We will show:

- How to inspect the intensities
- How to draw ROIs on the image
- How to view ROIs associated with different z or t planes
- How to use the Planes tab in OMERO.iviewer to see the spread of ROIs in multi-z, t images
- How to create comments on ROIs and enable and disable the ROI popups
- How to produce simple analysis results from the ROIs and export the results locally

3.7.7 Analyze

Analyze, save ROIs and measurements

3.7.8 Description

The following workflows should work both with ImageJ and Fiji, after these have been correctly set up with the OMERO plugin for Fiji/ImageJ.

Using the User Interface of the OMERO plugin, we will show:

- How to connect to OMERO using the OMERO plugin for ImageJ/Fiji.
- How to open an image from OMERO.server into Fiji/ImageJ.
- How to manually save ROIs and measurements as CSV back to the original image in OMERO.server.
- How to import a newly created image from Fiji/ImageJ into OMERO.

3.7.9 Present or publish data

Create figures using OMERO.figure

OMERO.figure is a web-based tool for creating figures from Images in OMERO. Image metadata can be used to facilitate figure creation.

3.7.10 Description

This guide covers:

- Opening images in OMERO.figure to create a new figure
- How to add additional images to an existing figure
- Arranging panels in the desired figure layout
- How to synchronize the rendering settings between panels
- How to add scalebars, labels and ROIs to panels
- How to save and export figures as PDF or TIFF

3.8 OMERO walkthrough for facility managers

In this document, we go through a common workflow that an imaging facility manager who facilitates OMERO usage for others might follow:

- Create and manage Groups and Users.
- Import data for others.
- Import large or heterogeneous data for others.
- Manage data for others.
- Set up OMERO for publication of data.
- Get users started.

3.8.1 Administrate Groups and Users

Click on the link below to get to the full content of this chapter. The excerpts of the chapter are highlighted below the link.

Administrate Groups and Users

This chapter will show how to manage groups and users using the graphical interface in OMERO.web and the command-line interface. Most of the following tasks below can only be done by users with some administrator privileges. We will show:

- How to manage groups, creating and editing a new/existing group.
- How to manage users, creating and editing a new/existing user.
- How to set up the OMERO server to be able to email all users.

3.8.2 Resources

- Documentation:
 - <https://docs.openmicroscopy.org/latest/omero/sysadmins/server-permissions.html>
 - <https://docs.openmicroscopy.org/latest/omero/sysadmins/restricted-admins.html>
 - <https://docs.openmicroscopy.org/omero/latest/sysadmins/cli/usergroup.html>
- Script for Command Line User/Group management
 - `create_groups_users.sh`
- File defining the User/Group setup used by the script
 - `create_groups_users_setup`

3.8.3 Import for others using the Desktop application

Click on the link below to get to the full content of this chapter. The excerpts of the chapter are highlighted below the link.

Import data using the Desktop Client

Second part of this import section will show how to import data for another user, using OMERO.insight. The user importing the data needs to have some admin or restricted-admin privileges. More information about restricted privileges can be found at <https://docs.openmicroscopy.org/latest/omero/sysadmins/restricted-admins.html>

The import for another user requires that the user doing the import has specific privileges. We will use the user importer1, this could be, for example, a facility manager.

We will show:

- How to install the OMERO.insight desktop client on Windows, Mac and Linux.
- How to import data for the user logged in using OMERO.insight.
- How to select a target Project and Dataset or create a new ones in OMERO for the imports.
- How to add Tags to imported images at the import stage, to facilitate the management of these images later in OMERO.server.
- How to import data for other users in OMERO.insight.

3.8.4 Import large data using Command Line Interface

Click on the link below to get to the full content of this chapter. The excerpts of the chapter are highlighted below the link.

Import data using the Command Line Interface (CLI)

3.8.5 Description

This chapter will show how to import data for another user, using Command Line Interface (CLI).

The import can be done by any user as long as they import the data for themselves.

In case of the import for others, the user importing the data needs to have some admin (or restricted-admin) privileges. More information about restricted privileges can be found at [restricted-admins](#).

In the example workflow below, a user with restricted administrator privileges is used with login name `importer1`. This could be in real life e.g. a facility manager.

We will show:

- How to import data using the CLI for myself and for others into a remote OMERO.server
- How to import data using the CLI “in-place”, which means not copying the imported data into OMERO. Instead, OMERO will point to the original location of “in-place” imported files, thus preventing data duplication.
- How to deal with imports of large amounts of data in CLI, using the `–bulk` option and helper `csv` and `yml` files which define what is to be imported and how.

3.8.6 Resources

- Documentation:
 - <https://docs.openmicroscopy.org/latest/omero/users/cli/installation.html>
 - <https://docs.openmicroscopy.org/omero/latest/users/cli/index.html>
 - <https://docs.openmicroscopy.org/omero/latest/users/cli/import-target.html>
 - <https://docs.openmicroscopy.org/omero/latest/sysadmins/in-place-import.html>
 - <https://docs.openmicroscopy.org/omero/latest/users/cli/import-bulk.html>
- Data: example images from
 - <https://downloads.openmicroscopy.org/images/DV/will/FRAP/>
- Bash script for performing in-place imports:
 - https://github.com/ome/training-scripts/blob/master/maintenance/scripts/in_place_import_as.sh
- Example files for bulk import
 - `bulk.yml`

3.8.7 Manage data for others

Click on the link below to get to the full content of this chapter. The excerpts of the chapter are highlighted below the link.

Data management and cooperation

In this document, we introduce the basic concepts of data management, such as browsing, navigating to others’ data, and changing the display of the images in OMERO. The example here uses OMERO.web, but majority of the features described here are also present in OMERO.insight.

Further, we show how to use the Command Line Interface (CLI) for data management, introducing mainly the features which are not present in the OMERO.web.

3.8.8 Description

We will show:

- How to browse data in OMERO.web, navigating to yours and other users' Images.
- How to use the basic layout of OMERO.web for Images organized in Projects and Datasets.
- How to use OMERO.web for viewing of High-Content Screening (HCS) data.
- How to use the Preview panel.
- How to adjust the rendering settings of your and other users' images from the Preview panel.
- How to organize Images in Projects and Datasets.
- How to *move the data between groups if you are data owner*.
- How to move the data between groups if you are an administrator working on behalf of others.
- How to *change the ownership of objects*.
- How to use *Command Line for duplicating objects* such as Images, Datasets or Projects.

3.8.9 Set up OMERO for publication of data

Click on the link below to get to the full content of this chapter. The excerpts of the chapter are highlighted below the link.

Prepare data for publication using OMERO

Users can publish Image data to the world using OMERO. Here we describe some steps to facilitate that. The steps are to be done by an administrator or restricted administrator in OMERO.

3.8.10 Description

We will show:

- How to set up configurations in OMERO.web for establishing of a `public` user.
- How to set up a `public` group.
- How to get the data into the `public` group.
- How to open the `public` group to the outer world.

3.8.11 Get users started

Here we describe how to get users started. This is a walkthrough for a facility manager in charge of introducing new users to OMERO.

3.8.12 Step-by-Step

1. Create the new user in OMERO and add them to the appropriate group(s) in OMERO. Consult the user and group creation sections of *Administrate Groups and Users*. In case your OMERO.server is set up for synchronization of logins with your institutional login system (LDAP), look for Managing LDAP Users in *Administrate Groups and Users*.
2. Explain to the user the basics of Group and User system in OMERO, drawing their attention to the importance of logging into the correct group when importing data. The user sets then their default group in OMERO correctly, as explained in *Administrate Groups and Users*, or you do it for the user.
3. Point the new user to the section explaining how to install and import data into OMERO using a Desktop client *Import data using the Desktop Client*.
4. Point the new user to the basic workflow example *OMERO walkthrough example*. This will give them hints about the basic steps usually done when working with OMERO and how to go through them.
5. Explain to the user the data management and annotation possibilities in OMERO, as highlighted in *Data management and cooperation* and *Annotate Data and Filter using Annotations*.
6. Consult with the user the analysis possibilities when using 3rd party tools with OMERO, as listed in *External Software and OMERO*.
7. Mention *OMERO.figure*. This will help to raise the usage of OMERO in your institution.

3.9 Prepare an OMERO server for training

This chapter describes a setup of a training OMERO.server at your institution. Further, it gives an overview of the other training resources as well as hints for trainers about how to present OMERO during a training session.

3.9.1 Resources

- Community contributions and setups: [Github issue](#).
- [Useful scripts](#) for setting up the training server.

3.9.2 Download of scripts and OMERO.cli environment setup

- This guide assumes that you have installed an OMERO.server for training purposes.
- Clone the [training-scripts](#) repository:

```
$ git clone https://github.com/ome/training-scripts.git
```

- Set up a OMERO.cli as specified under [CLI installation](#). Typically, this environment will be used on your local machine. Alternatively, you can use the OMERO.cli environment of the OMERO.server.

3.9.3 Setup of groups and users

Prepare groups and users as listed in the [provided template sheet](#). Assuming you have the [template sheet](#) in the same directory as the script `create_groups_users.sh`, you can run:

```
$ cd training-scripts/maintenance/scripts
$ HOST=$YOUR_SERVER_ADDRESS PASSWORD=$PASSWORD_FOR_ROOT bash create_groups_users.sh
```

which will create 50 users in your database. The users are members of four main OMERO.groups, which cover the group permissions allowed in OMERO. The Read-annotate group Lab 1 is the main group used in the trainings, which contains most of the data in a typical training. The setup in the [template sheet](#) sets this Lab 1 group as a default group in OMERO for all users.

Rename users to have first and last names of real people (the list of famous scientist names is used) running [the renaming script](#):

```
$ python rename_users.py trainer-1 $PASSWORD --server $YOUR_SERVER_ADDRESS
```

3.9.4 Import images

We like to use [in-place import](#) because it avoids duplicating the data on disk, but if you are only importing a small amount of data you may find non-in-place import (described below) is more convenient. In-place import is achieved by using [import bash script](#). Use the [publicly available data provided](#) or use your own data or [data downloaded from IDR](#). Shell into the machine where you have installed your OMERO.server (necessary for in-place import) and, assuming that you have for example your data in a folder `my-folder` which is visible from that machine, run:

```
$ HOST=localhost SUDOER=trainer-1 PASSWORD=$PASSWORD_FOR_trainer-1 IMPORTTYPE=normal_
↪NUMBER=15 FOLDER=/path/to/my-folder bash in_place_import_as.sh
```

This will in-place import the images from `my-folder` into a new Dataset named `/path/to/my-folder` for user-1 through user-15.

To achieve a better reproducibility of your imports, you can use a [bulk file](#) pointing to a list of paths of the image files. Assuming you have for example the bulk file `idr0021-experimentA-bulk.yml` located on the filesystem you are working on, together with the corresponding `idr0021-experimentA-filePaths.tsv` file and the images as specified in the `idr0021-experimentA-filePaths.tsv` which you have edited accordingly, you can run:

```
$ HOST=localhost SUDOER=trainer-1 PASSWORD=$PASSWORD_FOR_trainer-1 OMEUSER=trainer_
↪NUMBER=2 BULKFILE=/path/to/idr0021-experimentA-bulk.yml bash in_place_import_as.sh
```

to import the images for trainer-1 and trainer-2.

The [script](#) assumes by default [in-place imports](#). You can adjust it for “classsical”, non-in-place import by deleting the `--transfer=ln_s` from the script lines or, if using the [bulk file](#) workflow, comment out the `transfer` line from the bulk file.

Image data can also be populated by a [Python script](#) which copies images as `pixeldata` (i.e. not the original images) from [IDR](#) by default. You can adjust the three lines beginning with `idr_client = omero.client(host="idr.openmicroscopy.org", port=4064)` to copy from other OMERO.servers. Note that the script is only creating [images of single Z and T](#), and thus reducing the original images dimensions in case these are multi-Z or T images.:

```
$ python idr_copy_plate.py trainer-1 --server $YOUR_SERVER_ADDRESS $PASSWORD $PLATE_ID
```

This will copy the Plate with `$PLATE_ID` from IDR as trainer-1 into your server, but note that this `idr_copy_plate.py` script only works on Plates which have a single Field (Image) per Well.

3.9.5 Import metadata

Import metadata from a CSV via [OMERO.web](#) or [Command Line Interface](#).

Annotate images using training-scripts:

- **Copy Key-Value pairs** from IDR or other OMERO server. This script will copy MapAnnotations from Images in IDR to Images with the same names in your server (Datasets names must also match). The command below copies from *Project:1* in IDR to *Project:2* in your server. The Project in your server is owned and the action is executed by *trainer-1*:

```
$ python idr_get_map_annotation.py trainer-1 $PASSWORD_FOR_trainer-1 Project:1 ↵
↵Project:2 --server $YOUR_SERVER_ADDRESS
```

- **Add new Key-Value pairs**. The command below will add Key-Value pairs defined inside the script randomly to the images inside Datasets with name *big-dataset* for all 50 users in your server. The \$PASSWORD for all the users must be the same.:

```
$ python key_value_pairs.py $PASSWORD big-dataset --server $YOUR_SERVER_ADDRESS
```

- **Calibrate images**. The command below will calibrate all the images within Dataset named *western-blot*s to *0.33 micrometers per pixel* for all 50 users in your server. The \$PASSWORD for all the users must be the same.:

```
$ python calibrate_images.py $PASSWORD western-blot --server $YOUR_SERVER_ADDRESS
```

- **Add timestamps**. The command below will set timestamps on the timelapse images within Dataset named *time-stamps* with *delta T of 300 seconds* for all 50 users in your server. The \$PASSWORD for all the users must be the same.:

```
$ python set_timestamps.py $PASSWORD timestamp --server $YOUR_SERVER_ADDRESS
```

- **Propagate tags and ratings to all users**. Supposing that the *trainer-1* has a Dataset *to-tag* with Tags and Ratings on the images in the Dataset. Further, each user, such as *user-1*, *user-2* has the same-named Dataset with equivalent images in it, but with no Tags and Ratings (a typical situation after a fresh import of images). The command below will link the Tags of *trainer-1* which are linked to the images in the *to-tag* Dataset to the corresponding images in the *to-tag* Datasets of the users. The links between the Tags and the Images will belong to each user. Also, the Ratings which are on the Images of the *to-tag* Dataset of *trainer-1* will be re-created for the corresponding Images of the users and will belong to those users.:

```
$ python copy_tags_ratings.py to-tag $PASSWORD --server $YOUR_SERVER_ADDRESS
```

3.9.6 Add analytical metadata

Create an analysis results table using a script run from a 3rd party tool. For example, you can run the [segmentation script](#) in the [scripting editor of Fiji](#) on a Project in OMERO containing Datasets with Images which creates an OMERO.table and a CSV file with results and attaches these to that Project in OMERO.

These analytical results can be used to [showcase OMERO.parade](#).

3.9.7 Cleanup scripts

It might be of great advantage to be able to clean up in batches, but still selectively, metadata added to the images on your training server.

- **Delete ROIs** on all Images inside Datasets of specified name for all users on the server who have such Datasets. In the example below, the Dataset's name is `with-rois`. The `$PASSWORD` is the password of the user deleting the ROIs. The deleting user is `trainer-1` by default.:

```
$ python delete_ROIs.py --datasetname with-rois --server $YOUR_SERVER_ADDRESS  
↪ $PASSWORD
```

- **Delete Annotations** on all Images inside Datasets of specified name for all users on the server who have such Datasets. In the example below, the Dataset's name is `western-bLOTS` and the deleted annotation type is `FileAnnotation`. All other annotation types such as `TagAnnotation` etc. will be preserved. The `$PASSWORD` is the password of the user deleting the ROIs. The deleting user is `trainer-1` by default.:

```
$ python delete_annotations.py --anntype file --namespace none --server $YOUR_  
↪ SERVER_ADDRESS $PASSWORD western-bLOTS
```